Kalyanmoy Deb et al. (Eds.)

Genetic and Evolutionary Computation – GECCO 2004

Genetic and Evolutionary Computation Conference
Seattle, WA, USA, June 2004
Proceedings, Part II

LNCS 3103

2 Part II

Springer

Lecture Notes in Computer Science 3103

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

Springer
Berlin
Heidelberg
New York
Hong Kong
London
Milan
Paris
Tokyo

Kalyanmoy Deb Riccardo Poli
Wolfgang Banzhaf Hans-Georg Beyer
Edmund Burke Paul Darwen
Dipankar Dasgupta Dario Floreano
James Foster Mark Harman
Owen Holland Pier Luca Lanzi
Lee Spector Andrea Tettamanzi
Dirk Thierens Andy Tyrrell (Eds.)

Genetic and Evolutionary Computation - GECCO 2004

Genetic and Evolutionary Computation Conference
Seattle, WA, USA, June 26-30, 2004
Proceedings, Part II

Springer

eBook ISBN: 3-540-24855-2
Print ISBN: 3-540-22343-6

Created in the United States of America

Visit Springer's eBookstore at: http://ebooks.springerlink.com
and the Springer Global Website Online at: http://www.springeronline.com

Volume Editors

Riccardo Poli
Owen Holland
University of Essex
Department of Computer Science
Wivenhoe Park, Colchester, CO4 3SQ, UK
E-mail: {rpoli,owen } @ essex .ac.uk

Wolfganz Banzhaf
Department of Computer Science
Memorial University of Newfoundland
St. John's, NL, A1B 3X5, Canada
E-mail: banzhaf @cs.mun.ca

Hans-Georg Beyer
University of Dortmund
Systems Analysis Research Group
Joseph-von-Fraunhoferstr. 20
44221 Dortmund, Germany
hans-georg.beyer@cs.uni-dortmund.de

Edmund Burke
University of Nottingham
School of Computer Science
and Information Technology
Jubilee Campus, Nottingham NG8 2BB, UK
E-mail: ekb@cs.nott.ac.uk

Paul Darwen
Anadare Pty Ltd
14 Annie Street, Brisbane
Queensland 4066, Australia
E-mail: darwen@ieee.org

Dipankar Dasgupta
University of Memphis
Division of Computer Science
Memphis, TN 38152, USA
E-mail: dasgupta@memphis.edu

Dario Floreano
Swiss Federal Institute of Technology
Autonomous Systems Laboratory
1015 Lausanne, Switzerland
E-mail: dario.floreano@epfl.ch

James Foster
University of Idaho
Department of Computer Science
P.O. Box 441010, Moscow
ID 83844-1010, USA
E-mail: foster@uidaho.edu

Mark Harman
Brunel University
Department of Information Systems
and Computing
Uxbridge, Middlesex, 11B8 3PH, UK
E-mail: Mark.Harman@brunel.ac.uk

Pier Luca Lanzi
Politecnico di Milano
Dipartimento di Elettronica e Informazione
Piazza Leonardo da Vinci, 32
20133 Milan, Italy
E-mail: lanzi@elet.polimi.it

Lee Spector
Hampshire College, Cognitive Science
Amherst, MA 01002, USA
E-mail: lspector@hampshire.edu

Andrea G.B. Tettamanzi
University of Milan
Department of Information Technology
Via Bramante 65, 26013 Crema, Italy
E-mail: andrea.tettamanzi@unimi.it

Dirk Thierens
Utrecht University
Department of Information
and Computing Sciences
P.O. Box 80.089, 3508 TB Utrecht
The Netherlands
E-mail: dirk.thierens@cs.uu.nl

Andrew M. Tyrrell
The University of York
The Department of Electronics
Heslington, York YO10 5DD, UK
E-mail: amt@ohm.york.ac.uk

This page intentionally left blank

Preface

These proceedings contain the papers presented at the sixth annual Genetic and Evolutionary Computation Conference (GECCO 2004). The conference was held in Seattle, during June 26–30, 2004.

A total of 460 papers were submitted to GECCO 2004. After a rigorous double-blind reviewing process, 230 papers were accepted for full publication and oral presentation at the conference, resulting in an acceptance rate of 50%. An additional 104 papers were accepted as posters with two-page extended abstracts included in these proceedings.

This year's GECCO constituted the union of the Ninth Annual Genetic Programming Conference (which has met annually since 1996) and the Thirteenth International Conference on Genetic Algorithms (which, with its first meeting in 1985, is the longest running conference in the field). Since 1999, these conferences have merged to produce a single large meeting that welcomes an increasingly wide array of topics related to genetic and evolutionary computation.

Since the fifth annual GECCO conference, the proceedings have been published by Springer-Verlag as part of their Lecture Notes in Computer Science (LNCS) series. This makes the proceedings available in many libraries as well as online, widening the dissemination of the research presented at the conference. In addition to these proceedings volumes, each participant of the GECCO 2004 conference received a CD containing electronic versions of the papers presented.

A new track entitled 'Biological Applications' was introduced this year to emphasize the use of evolutionary computing methods to various biological applications, such as bioinformatics and others.

In addition to the presentation of the papers contained in the proceedings, the conference included 16 workshops, 32 tutorials by leading specialists, the Evolutionary Computation in Industry special track, and presentation of late-breaking papers.

GECCO is sponsored by the International Society for Genetic and Evolutionary Computation (ISGEC). The ISGEC by-laws contain explicit guidance on the organization of the conference, including the following principles:

(i) GECCO should be a broad-based conference encompassing the whole field of genetic and evolutionary computation.
(ii) Papers will be published and presented as part of the main conference proceedings only after being peer reviewed. No invited papers shall be published (except for those of up to three invited plenary speakers).
(iii) The peer review process shall be conducted consistently with the principle of division of powers performed by a multiplicity of independent program committees, each with expertise in the area of the paper being reviewed.
(iv) The determination of the policy for the peer review process for each of the conference's independent program committees and the reviewing of papers for each program committee shall be performed by persons who occupy their

positions by virtue of meeting objective and explicitly stated qualifications based on their previous research activity.

(v) Emerging areas within the field of genetic and evolutionary computation shall be actively encouraged and incorporated in the activities of the conference by providing a semi-automatic method for their inclusion (with some procedural flexibility extended to such emerging new areas).

(vi) The percentage of submitted papers that are accepted as regular full-length papers (i.e., not posters) shall not exceed 50%.

These principles help ensure that GECCO maintains high quality across the diverse range of topics it includes.

Besides sponsoring the conference, the ISGEC supports the field in other ways. ISGEC sponsors the biennial "Foundations of Genetic Algorithms" workshop on theoretical aspects of all evolutionary algorithms. The journals *Evolutionary Computation* and *Genetic Programming and Evolvable Machines* are also supported by ISGEC. All ISGEC members (including students) receive subscriptions to these journals as part of their membership. ISGEC membership also includes discounts on GECCO registration rates as well as discounts on other journals. More details on ISGEC can be found online at http://www.isgec.org.

Many people volunteered their time and energy to make this conference a success. The following people in particular deserve the gratitude of the entire community for their outstanding contributions to GECCO 2004:

- Riccardo Poli, the General Chair of GECCO 2004 for his tireless efforts in organizing every aspect of the conference, which started well before GECCO 2003 took place in Chicago in July 2003
- David E. Goldberg, John Koza and Riccardo Poli, members of the Business Committee, for their guidance and financial oversight
- Stefano Cagnoni, for coordinating the workshops
- Maarten Keijzer, for editing the late breaking papers
- Past conference organizers, James Foster and Erick Cantú-Paz, for their constant help and advice
- John Koza, for his efforts as publicity chair
- Simon Lucas, for arranging competitions during GECCO 2004
- Mike Cattolico, for local arrangements
- Pat Cattolico, for her help in the local organization of the conference
- Carol Hamilton, Ann Stolberg, and the rest of the AAAI staff for their outstanding efforts administering the conference
- Thomas Preuss, for maintaining the ConfMaster Web-based paper review system
- Gerardo Valencia, for Web programming and design
- Jennifer Ballentine, Lee Ballentine and the staff of Professional Book Center, for assisting in the production of the proceedings
- Alfred Hofmann and Ursula Barth of Springer-Verlag for the production of the GECCO 2004 proceedings; and

the sponsors who made generous contributions to support student travel grants:

- Air Force Office of Scientific Research
- New Light Industries
- Philips Research
- Tiger Mountain Scientic
- Unilever.

The track chairs deserve special thanks. Their efforts in recruiting program committees, assigning papers to reviewers, and making difficult acceptance decisions in relatively short times, were critical to the success of the conference:

- Owen Holland, A-Life, Adaptive Behavior, Agents, and Ant Colony Optimization,
- Dipankar Dasgupta, Artificial Immune Systems
- James Foster and Wolfgang Banzhaf, Biological Applications
- Paul Darwen, Coevolution
- Hans-Georg Beyer, Evolution Strategies, Evolutionary Programming
- Dario Floreano, Evolutionary Robotics
- Edmund Burke, Evolutionary Scheduling and Routing
- Andy Tyrrell, Evolvable Hardware
- Dirk Thierens, Genetic Algorithms
- Lee Spector, Genetic Programming
- Pier Luca Lanzi, Learning Classifier Systems
- Andrea Tettamanzi, Real World Applications
- Mark Harman, Search Based Software Engineering

The conference was held in cooperation and/or affiliation with:

- The American Association for Artificial Intelligence (AAAI)
- The 2004 NASA/DoD Conference on Evolvable Hardware
- *Evolutionary Computation*
- *Genetic Programming and Evolvable Machines*
- *Journal of Scheduling*
- *Journal of Hydroinformatics*
- *Applied Soft Computing*

Above all, our special thanks are due to the numerous researchers and practitioners who submitted their best work to GECCO 2004, reviewed the work of others, presented tutorials, organized workshops, or volunteered their time in any other way. I am sure that the contributors to this proceedings will be proud of the results of their efforts and readers will get a glimpse of the current activities in the field of genetic and evolutionary computation.

April 2004

Kalyanmoy Deb
Editor-in-Chief, GECCO 2004

This page intentionally left blank

GECCO 2004 Conference Organization

Conference Committee

General Chair: Riccardo Poli
Proceedings Editor-in-Chief: Kalyanmoy Deb
Business Committee: David E. Goldberg, John Koza, Riccardo Poli
Chairs of Program Policy Committees:
Owen Holland, A-Life, Adaptive Behavior, Agents, and Ant Colony Optimization
Dipankar Dasgupta, Artificial Immune Systems
James Foster and Wolfgang Banzhaf, Biological Applications
Paul Darwen, Coevolution
Hans-Georg Beyer, Evolution Strategies, Evolutionary Programming
Dario Floreano, Evolutionary Robotics
Edmund Burke, Evolutionary Scheduling and Routing
Andy Tyrrell, Evolvable Hardware
Dirk Thierens, Genetic Algorithms
Lee Spector, Genetic Programming
Pier Luca Lanzi, Learning Classifier Systems
Andrea Tettamanzi, Real World Applications
Mark Harman, Search Based Software Engineering

Late Breaking Papers Chair: Maarten Keijzer
Workshops Chair: Stefano Cagnoni

Workshop Organizers

E. Costa, F. Pereira , G. Raidl, Application of Hybrid Evolutionary Algorithms to Complex Optimization Problems
S.C. Upton and D.E. Goldberg, Military and Security Applications of Evolutionary Computation
H. Lipson, E. De Jong and J. Koza, Modularity, Regularity and Hierarchy in Open-Ended Evolutionary Computation
H. Suzuki and H. Sawai, Evolvability in Evolutionary Computation (EEC)
I. Parmee, Interactive Evolutionary Computing
M. Pelikan, K. Sastry and D. Thierens, Optimization by Building and Using Probabilistic Models (OBUPM 2004)
W. Stolzmann, P.L. Lanzi, S.W. Wilson, International Workshop on Learning Classifier Systems (IWLCS)
S. Mueller, S. Kern, N. Hansen and P. Koumoutsakos, Learning, Adaptation, and Approximation in EC, Jiri Ocenasek
M. O'Neill and C. Ryan, Grammatical Evolution (GEWS 2004)
T. Yu, Neutral Evolution in Evolutionary Computation
J.F. Miller, Regeneration and Learning in Developmental Systems (WORLDS)

I. Garibay, G. Holifield and A.S. Wu, Self-Organization on Representations for Genetic and Evolutionary Algorithms
A. Wright and N. Richter, Evolutionary Computation Theory
Jason H. Moore and Marylyn D. Ritchie, Biological Applications of Genetic and Evolutionary Computation (BioGEC 2004)
T. Riopka, Graduate Student Workshop
M.M. Meysenburg, Undergraduate Student Workshop

Tutorial Speakers

Erik Goodman, Genetic Algorithms
John Koza, Genetic Programming
Thomas Bäck, Evolution Strategies
Kenneth De Jong, A Unified Approach to EC
Tim Kovacs, Learning Classifier Systems
Martin Pelikan, Probabilistic Model-Building GAs
Russ Eberhart, Particle Swarm Optimization
Steffen Christensen and Mark Wineberg, Introductory Statistics for Evolutionary Computation
W.B. Langdon, Genetic Programming Theory
Jonathan Rowe, Genetic Algorithm Theory
J. Foster and W. Banzhaf, Biological Applications
Chris Stephens, Taxonomy and Coarse Graining in EC
Darrell Whitley, No Free Lunch
Kalyanmoy Deb, Multiobjective Optimization with EC
Ingo Wegener, Computational Complexity and EC
Julian Miller, Evolvable Physical Media
Tetsuya Higuchi, Evolvable Hardware Applications
Franz Rothlauf, Representations
Lee Altenberg, Theoretical Population Genetics
Ingo Rechenberg, Bionik: Building on Biological Evolution
Marco Tomassini, Spatially Structured EAs
Hideyuki Takagi, Interactive Evolutionary Computation
Garry Greenwood, Evolutionary Fault Tolerant Systems
Maarten Keijzer, GP for Symbolic Regression
Conor Ryan, Grammatical Evolution
Dario Floreano, Evolutionary Robotics
Al Biles, Evolutionary Music
Peter Ross, EAs for Combinatorial Optimization
Jürgen Branke, Optimization in Dynamic Environments
Ian Parmee, Evolutionary Algorithms for Design
Xin Yao, Evolving Neural Networks
Arthur Kordon, Guido Smits and Mark Kotanchek, Industrial Evolutionary Computing

Keynote Speakers

Leroy Hood, President, Institute for Systems Biology, Seattle
François Baneyx, Professor of Chemical Engineering and adjunct Professor of Bioengineering, Center for Nanotechnology at University of Washington, Seattle

Members of the Program Committee

Hussein Abbass
Andrew Adamatzky
Adam Adamopoulos
Alexandru Agapie
Jose Aguilar
Jesus Aguilar-Ruiz
Hernan Aguirre
Uwe Aickelin
Javier Alcaraz Soria
Lee Altenberg
Giuliano Antoniol
Shawki Areibi
Tughrul Arslan
Dan Ashlock
Anne Auger
R. Muhammad Atif Azad
B.V. Babu
Thomas Bäck
Karthik Balakrishnan
Gianluca Baldassarre
Julio Banga
Ranieri Baraglia
Alwyn Barry
Thomas Bartz-Beielstein
Cem Baydar
Theodore Belding
Fevzi Bell
Michael Bender
Peter Bentley
Aviv Bergman
Ester Bernado-Mansilla
Tim Blackwell
Jacek Blazewicz
Lashon Booker
Peter Bosman
Klaus Bothe
Leonardo Bottaci
Jürgen Branke
Wilker Bruce
Peter Brucker
Anthony Bucci
Bill P. Buckles
Dirk Bueche
Larry Bull
Martin Butz
Stefano Cagnoni
Xiaoqiang Cai
Alexandre Caminada
Erick Cantú-Paz
Nachol Chaiyaratana
Uday Chakraborty
Partha Chakroborty
Weng Tat Chan
Alastair Channon
Kumar Chellapilla
Shu-Heng Chen
Ying-ping Chen
Prabhas Chongstitvatana
John Clark
Maurice Clerc
André Coelho
Carlos Coello Coello
Myra Cohen
David Coley
Philippe Collard
Pierre Collet
Clare Congdon
David Corne
Luis Correia
Ernesto Costa
Carlos Cotta
Peter Cowling
Bart Craenen
Keshav Dahal
Rajarshi Das
Leandro de Castro
Patrick De Causmaecker
Ivanoe De Falco
Hugo de Garis
Edwin de Jong
David de la Fuente
Anthony Deakin
Kalyanmoy Deb
Myriam Delgado
Medha Dhurandhar
Ezequiel Di Paolo
Jose Javier Dolado Cosin
Keith Downing
Kath Dowsland
Gerry Dozier
Rolf Drechsler
Stefan Droste
Tim Edwards
Aniko Ekart
Mark Embrechts
Michael Emmerich
Maria Fasli
Francisco Fernandez
Bogdan Filipic
Peter Fleming
Stuart Flockton
Carlos Fonseca
James Foster
Alex Freitas
Clemens Frey
Christian Gagné
Luca Gambardella
Josep Maria Garrell-Guiu
Michel Gendreau
Pierre Gerard
Andreas Geyer-Schulz
Robert Ghanea-Hercock

Marco César Goldbarg
Faustino Gomez
Jonatan Gomez
Fabio Gonzalez
Tim Gosling
Jens Gottlieb
Buster Greene
Garrison Greenwood
Gary Greenwood
Michael Gribskov
Hans-Gerhard Gross
Steven Gustafson
Charlie Guthrie
Walter Gutjahr
Pauline Haddow
Hani Hagras
Hisashi Handa
Nikolaus Hansen
Dave Harris
Emma Hart
Inman Harvey
Jun He
Robert Heckendorn
Jeffrey Herrmann
Rob Hierons
David Hillis
Steven Hofmeyr
John Holmes
Jeffrey Horn
Daniel Howard
Jianjun Hu
Phil Husbands
Hitoshi Iba
Christian Igel
Auke Jan Ijspeert
Akio Ishiguro
Christian Jacob
Thomas Jansen
Yaochu Jin
Colin Johnson
Bryan Jones
Bryant Julstrom
Mahmoud Kaboudan
Sanza Kazadi
Maarten Keijzer
Douglas Kell
Graham Kendall
Mathias Kern
Didier Keymeulen
Joshua Knowles
Arthur Kordon
Bogdan Korel
Erkan Korkmaz
Petros Koumoutsakos
Tim Kovacs
Natalio Krasnogor
Krzysztof Krawiec
Kalmanje Krishnakumar
Renato Krohling
Gabriella Kûkai
Rajeev Kumar
Raymond Kwan
Sam Kwong
Han La Poutre
Shyong Lam
Gary Lamont
W. B. Langdon
Pedro Larranaga
Jesper Larsen
Claude Lattaud
Marco Laumanns
Claude Le Pape
Martin Lefley
Tom Lenaerts
K. S. Leung
Lukas Lichtensteiger
Anthony Liekens
Hod Lipson
Fernando Lobo
Jason Lohn
Michael Lones
Sushil Louis
Jose Lozano
Evelyne Lutton
Bob MacCallum
Nicholas Macias
Ana Madureira
Spiros Mancoridis
Vittorio Maniezzo
Elena Marchiori
Peter Martin
Andrew Martin
Alcherio Martinoli
Iwata Masaya
Shouichi Matsui
Dirk Mattfeld
Barry McCollum
Nic McPhee
Jörn Mehnen
Karlheinz Meier
Lawrence Merkle
Jean-Arcady Meyer
Christoph Michael
Zbigniew Michalewicz
Olivier Michel
Martin Middendorf
Stuart Middleton
Orazio Miglino
Julian Miller
Brian Mitchell
Chilukuri Mohan
Francesco Mondada
David Montana
Byung-Ro Moon
Frank Moore
Jason Moore
Alberto Moraglio
J. Manuel Moreno
Masaharu Munetomo
Hajime Murao
Kazuyuki Murase
Olfa Nasraoui
Bart Naudts
Norberto Eiji Nawa
Chrystopher Nehaniv
Miguel Nicolau
Fernando Nino
Stefano Nolfi
Peter Nordin
Bryan Norman
Cedric Notredame
Wim Nuijten
Una-May O'Reilly
Markus Olhofer
Sigaud Olivier
Michael O'Neill
Ender Ozcan
Anil Patel
Shail Patel
Martin Pelikan

Carlos-Andrés Pena-Reyes
Francisco Pereira
Sanja Petrovic
Hartmut Pohlheim
Daniel Polani
Marie-Claude Portmann
Jean-Yves Potvin
Alexander Pretschner
Thomas Preuss
Mike Preuss
Adam Prugel-Bennett
Joao Pujol
Günther Raidl
Khaled Rasheed
Al Rashid
Thomas Ray
Tapabrata Ray
Victor Rayward-Smith
Patrick Reed
Richard Reeve
Colin Reeves
Marek Reformat
Andreas Reinholz
Rick Riolo
Jose Riquelme Santos
Marc Roper
Franz Rothlauf
Rajkumar Roy
Guenter Rudolph
Kazuhiro Saitou
Arthur Sanderson
Eugene Santos
Kumara Sastry
Yuji Sato
Thorsten Schnier
Marc Schoenauer
Sonia Schulenburg
Alan Schultz
Hans-Paul Schwefel
Mikhail Semenov
Sandip Sen
Bernhard Sendhoff
Kisung Seo
Martin Shepperd
Alaa Sheta
Richard Skalsky
Jim Smith
Don Sofge
Terry Soule
Pieter Spronck
Peter Stadler
Kenneth Stanley
Chris Stephens
Harmen Sthamer
Christopher Stone
Matthew Streeter
Thomas Stuetzle
Raj Subbu
Keiki Takadama
Kiyoshi Tanaka
Uwe Tangen
Alexander Tarakanov
Gianluca Tempesti
Sam Thangiah
Scott Thayer
Lothar Thiele
Jonathan Thompson
Jonathan Timmis
Jon Timmis
Ashutosh Tiwari
Marco Tomassini
Jim Torresen
Paolo Toth
Edward Tsang
Shigeyoshi Tsutsui
Supiya Ujjin
Steven van Dijk
Jano van Hemert
Frederik Vandecasteele
Greet Vanden Berghe
Leonardo Vanneschi
Robert Vanyi
Oswaldo Velez-Langs
J. L. Verdegay
Fernando Von Zuben
Roger Wainwright
Matthew Wall
Harold Wareham
Jean-Paul Watson
Everett Weber
Ingo Wegener
Karsten Weicker
Peter Whigham
Shimon Whiteson
Darrell Whitley
R. Wiegand
Stewart Wilson
Mark Wineberg
Alden Wright
Annie Wu
Zheng Wu
Jinn-Moon Yang
Tina Yu
Hongnian Yu
Ricardo Zebulum
Andreas Zell
Byoung-Tak Zhang
Gengui Zhou
Fan Zhun
Tom Ziemke
Lyudmilla Zinchenko
Eckart Zitzler

This page intentionally left blank

A Word from the Chair of ISGEC

You may have just picked up your proceedings, in hard copy and CD-ROM, at GECCO 2004. We've chosen once again to work with Springer-Verlag, including our proceedings as part of their Lecture Notes in Computer Science (LNCS) series, which makes them available in many libraries, broadening the impact of the GECCO conference.

If you're now at GECCO 2004, we, the organizers, hope your experience is memorable and productive, and you will find the proceedings to be of continuing value. The opportunity for first-hand interaction among authors and other participants at GECCO is a big part of what makes it exciting, and we all hope you come away with many new insights and ideas.

If you were unable to come to GECCO 2004 in person, I hope you'll find many stimulating ideas from the world's leading researchers in evolutionary computation reported in the proceedings, and that you'll be able to participate in future GECCO conferences, for example, next year, in the Washington, DC area!

The International Society for Genetic and Evolutionary Computation, sponsoring organization of the annual GECCO conferences, is a young organization, formed through merger of the International Society for Genetic Algorithms (sponsor of the ICGA conferences) and the organization responsible for the annual Genetic Programming Conferences. It depends strongly on the voluntary efforts of many of its members. It is designed to promote not only exchange of ideas among innovators and practitioners of well-known methods such as genetic algorithms, genetic programming, evolution strategies, evolutionary programming, learning classifier systems, etc., but also the growth of newer areas such as artificial immune systems, evolvable hardware, agent-based search, and others. One of the founding principles is that ISGEC operates as a confederation of groups with related but distinct approaches and interests, and their mutual prosperity is assured by their representation in the program committees, editorial boards, etc., of the conferences and journals with which ISGEC is associated. This also insures that ISGEC and its functions continue to improve and evolve with the diversity of innovation that has characterized our field.

The ISGEC saw many changes last year, in addition to its growth in membership. We anticipate yet more advances in the next year. A second round of Fellows and Senior Fellows will be added to our society this year, after last year's inaugural group. GECCO continues to be subject to dynamic development – the many new tutorials, workshop topics, and tracks will evolve again next year, seeking to follow and encourage the developments of the many fields represented at GECCO. The best paper awards will be presented for the third time at this GECCO, and we hope many of you will participate in the balloting. This year, most presentations at GECCO will once again be made electronically, displayed with the LCD projectors that ISGEC purchased last year. Our journals, Evolutionary Computation and Genetic Programming and Evolvable Machines, continue to prosper, and we are exploring ways to make them even more widely available.

The ISGEC is your society, and we urge you to become involved or continue your involvement in its activities, to the mutual benefit of the whole evolutionary computation community. Three members were re-elected to five-year terms on the Executive Board at GECCO 2003 – Ken De Jong, David Goldberg, and Erik Goodman.

Since that time, the ISGEC has been active on many issues, through actions of the Board and our two Councils – the Council of Authors and the Council of Conferences. Last year, the Board voted to combine the Council of Authors and Council of Editors into a single body, the Council of Authors.

The organizers of GECCO 2004 are shown in this front matter, but special thanks are due to Riccardo Poli, General Chair, and Kalyanmoy Deb, Editor-in-Chief of the proceedings, as well as to John Koza and Dave Goldberg, the Business Committee. Each year has seen many new features in GECCO, and it is the outstanding efforts of this group that "make GECCO come together."

Of course, we all owe a great debt to those who chaired or served on the various Core and Special Program Committees that reviewed all of the papers for GECCO 2004. Without their effort, it would not be possible to put on a meeting of this quality.

Another group also deserves the thanks of GECCO participants and ISGEC members – the members of the ISGEC Executive Board and Councils, who are listed on the next page. I am particularly indebted to them for their thoughtful contributions to the organization and their continuing demonstrations of concern for the welfare of the ISGEC.

I invite you to communicate with me (goodman@egr.msu.edu) if you have questions or suggestions for ways ISGEC can be of greater service to its members, or if you would like to get more involved in ISGEC and its functions.

Don't forget about the eighth Foundations of Genetic Algorithms (FOGA) workshop, also sponsored by ISGEC, the biennial event that brings together the world's leading theorists on evolutionary computation. FOGA will be held January 5–9, 2005 at the University of Aizu, Japan, which will be a fascinating place to visit for those of us who haven't spent much time in Japan. I hope you'll join many of your fellow ISGEC members there!

Finally, I hope to see you at GECCO 2005 in the Washington, DC area. Get your ideas for new things for GECCO 2005 to Una-May O'Reilly, the General Chair of GECCO 2005, when you see her at GECCO 2004, and please check the ISGEC Web site, www.isgec.org, regularly for details as the planning for GECCO 2005 continues.

Erik D. Goodman
ISGEC Chair

ISGEC Executive Board

Erik D. Goodman (chair), Michigan State University
David Andre, BodyMedia, Inc, Pittsburgh
Wolfgang Banzhaf, Memorial University of Newfoundland
Kalyanmoy Deb, Indian Institute of Technology Kanpur
Kenneth De Jong, George Mason University
Marco Dorigo, Université Libre de Bruxelles
David E. Goldberg, University of Illinois at Urbana-Champaign
John H. Holland, University of Michigan & Sante Fe Institute
John R. Koza, Stanford University
Una-May O'Reilly, Massachusetts Institute of Technology
Ingo Rechenberg, Technical University of Berlin
Marc Schoenauer, INRIA Futurs
Lee Spector, Hampshire College
Darrell Whitley, Colorado State University
Annie S. Wu, University of Central Florida

Council of Authors

Kalyanmoy Deb (chair), Indian Institute of Technology Kanpur
David Andre, University of California at Berkeley
Plamen P. Angelov, Loughborough University
Vladan Babovic, Danish Hydraulic Institute
Karthik Balakrishnan, Fireman's Fund Insurance Company
Wolfgang Banzhaf, University of Dortmund
Forrest H. Bennett III, FX Palo Alto Laboratory, Inc.
Peter Bentley, University College, London
Hans-Georg Beyer, University of Dortmund
Jürgen Branke, University of Karlsruhe
Martin Butz, University of Illinois at Urbana-Champaign
Erick Cantú-Paz, Lawrence Livermore National Laboratory
Lance D. Chambers, Western Australian Department of Transport
Runwei Cheng, Ashikaga Institute of Technology
Carlos A. Coello Coello, CINVESTAV-IPN
David A. Coley, University of Exeter
Dipankar Dasgupta, University of Memphis
Kenneth De Jong, George Mason University
Marco Dorigo, IRIDIA, Université Libre de Bruxelles
Rolf Drechsler, University of Freiburg
Agoston E. Eiben, Vrije Universiteit Amsterdam
Emanuel Falkenauer, Optimal Design and Brussels University ULB
Stephanie Forrest, University of New Mexico
James Foster, University of Idaho
Mitsuo Gen, Ashikaga Institute of Technology
Andreas Geyer-Schulz, University of Karlsruhe

David E. Goldberg, University of Illinois at Urbana-Champaign
Jens Gottlieb, SAP AG
Wolfgang A. Halang, FernUniversität, Hagen
John H. Holland, University of Michigan & Sante Fe Institute
Hitoshi Iba, University of Tokyo
Christian Jacob, University of Calgary
Francisco Herrera, University of Granada
Yaochu Jin, Honda Research Institute Europe
Robert E. Keller, University of Dortmund
Dimitri Knjazew, SAP AG
John R. Koza, Stanford University
Sam Kwong, City University of Hong Kong
W.B. Langdon, University College, London
Dirk C. Mattfeld, University of Bremen
Pinaki Mazumder, University of Michigan
Zbigniew Michalewicz, University of North Carolina at Charlotte
Eric Michielssen, University of Illinois at Urbana-Champaign
Melanie Mitchell, Oregon Health and Science University
Byung-Ro Moon, Seoul National University
Michael O'Neill, University of Limerick
Ian Parmee, University of North Carolina at Charlotte
Witold Pedrycz, University of Alberta
Frederick E. Petry, University of North Carolina at Charlotte
Riccardo Poli, University of Essex
Rajkumar Roy, Cranfield University
Elizabeth M. Rudnick, University of Illinois at Urbana-Champaign
Conor Ryan, University of Limerick
Marc Schoenauer, INRIA Futurs
Moshe Sipper, Swiss Federal Institute of Technology
James E. Smith, University of the West of England
Terence Soule, University of Idaho
William M. Spears, University of Wyoming
Lee Spector, Hampshire College
Wallace K.S. Tang, Swiss Federal Institute of Technology
Adrian Thompson, University of Sussex
Jose L. Verdegay, University of Granada
Michael D. Vose, University of Tennessee
Darrell Whitley, Colorado State University
Man Leung Wong, Lingnan University

Council of Conferences, Una-May O'Reilly (chair)

The purpose of the Council of Conferences is to provide information about the numerous conferences that are available to researchers in the field of Genetic and Evolutionary Computation, and to encourage them to coordinate their meetings to maximize our collective impact on science.

ACDM, Adaptive Computing in Design and Manufacture, Bristol, UK, April 2004, Ian Parmee, Ian.Parmee@uwe.ac.uk

EuroGP, European Conference on Genetic Programming, Coimbra, Portugal, April 2004, Ernesto Costa, ernesto@dei.uc.pt

EvoCOP, European Conference on Evolutionary Computation in Combinatorial Optimization, Coimbra, Portugal, April 2004, Günther Raidl, raidl@ads.tuwien.ac.at and Jens Gottlieb, jens.gottlieb@sap.com

EvoWorkshops, European Evolutionary Computing Workshops, Portugal, Coimbra, Portugal, April 2004, Stefano Cagnoni, cagnoni@ce.unipr.it

FOGA, Foundations of Genetic Algorithms Workshop, Fukushima, Japan, January 2005, Lothar M. Schmitt, info@foga05.org

GECCO2004, Genetic and Evolutionary Computation Conference, Seattle, USA, June 2004, Riccardo Poli, rpoli@essex.ac.uk

PATAT 2004, 5th International Conference on the Practice and Theory of Automated Timetabling, Pittsburgh, USA, August 2004, Edmund Burke, ekb@cs.nott.ac.uk

PPSN-VIII, Parallel Problem Solving from Nature, Birmingham, UK, September 2004, Xin Yao, xin@cs.bham.ac.uk

SAB, 8th international conference on Simulation of Adaptive Behavior, Los Angeles, USA, July 2004, John Hallam, john@mip.sdu.dk and Jean-Arcady Meyer, jean-arcady.meyer@lip6.fr

EMO 2005, 3rd Evolutionary Multi-Criterion Optimization, Guanajuato, Mexico, March 2005, Carlos Coello Coello, coello@cs.cinvestav.mx

An up-to-date roster of the Council of Conferences is available online at http://www.isgec.org/conferences.html.

Papers Nominated for Best Paper Awards

In 2002, ISGEC created a best paper award for GECCO. As part of the double blind peer review, the reviewers were asked to nominate papers for best paper awards. The Chairs of Core and Special Program Committees selected the papers that received the most nominations for consideration by the conference. One winner for each program track was chosen by secret ballot of the GECCO attendees after the papers had been presented in Chicago. The titles and authors of all 32 papers nominated for the best paper award for GECCO 2004 are given below:

Robot Trajectory Planner Using Multi-objective Genetic Algorithm Optimization: E.J. Solteiro Pires, J.A. Tenreiro Machado, and P.B. de Moura Oliveira I-615

Evolved Motor Primitives and Sequences in a Hierarchical Recurrent Neural Network: Rainer Paine and Jun Tani I-603

Actuator Noise in Recombinant Evolution Strategies on General Quadratic Fitness Models: Hans-Georg Beyer I-654

An Analysis of the $(\mu + 1)$ EA on Simple Pseudo-Boolean Functions: Carsten Witt I-761

On the Choice of the Population Size: Tobias Storch I-748

Gradient-Based Learning Updates Improve XCS Performance in Multistep Problems: Martin Butz, David E. Goldberg, and Pier Luca Lanzi II-751
High Classification Accuracy Does Not Imply Effective Genetic Search: Tim Kovacs and Manfred Kerber II-785
Mixed Decision Trees: Minimizing Knowledge Representation Bias in LCS: Xavier Llorà and Stewart Wilson II-797
Genetic Programming Neural Networks as a Bioinformatics Tool for Human Genetics: Marylyn Ritchie, Christopher Coffey, and Jason Moore I-438
Fuzzy Dominance Based Multi-objective GA-Simplex Hybrid Algorithms Applied to Gene Network Models: Praveen Koduru, Sanjoy Das, Stephen Welch, and Judith L. Roe I-356
Evaluating Evolutionary Testability with Software-Measurements: Frank Lammermann, Andre Baresel, and Joachim Wegener II-1350
Hybridizing Evolutionary Testing with the Chaining Approach: Phil McMinn and Mike Holcombe II-1363
Vulnerability Analysis of Immunity-Based Intrusion Detection Systems Using Evolutionary Hackers: Gerry Dozier, Douglas Brown, John Hurley, and Krystal Cain I-263
πGrammatical Evolution: Michael O'Neill, Anthony Brabazon, Miguel Nicolau, Sean McGarraghy, and Peter Keenan II-617
Evolving En-Route Caching Strategies for the Internet: Jürgen Branke, Pablo Funes, and Frederik Thiele II-434
A Descriptive Encoding Language for Evolving Modular Neural Networks: Jae-Yoon Jung and James A. Reggia II-519
Shortcomings with Tree-Structured Edge Encodings for Neural Networks: Gregory Hornby II-495
Evolving Quantum Circuits and Programs Through Genetic Programming: Paul Massey, John Clark, and Susan Stepney II-569
Adaptive and Evolvable Network Services: Tadashi Nakano and Tatsuya Suda I-151
Using Clustering Techniques to Improve the Performance of a Multi-objective Particle Swarm Optimizer: Gregorio Toscano Pulido and Carlos Coello Coello I-225
An Interactive Artificial Ant Approach to Non-photorealistic Rendering: Yann Semet, Una-May O'Reilly, and Frédo Durand I-189
A Broad and Narrow Approach to Interactive Evolutionary Design – An Aircraft Design Example: Oliver Bandte and Sergey Malinchik II-883
Evolutionary Drug Scheduling Model for Cancer Chemotherapy: Yong Liang, Kwong-Sak Leung, and Tony Shu Kam Mok II-1166
An Enhanced Genetic Algorithm for DNA Sequencing with Positive and Negative Errors: Thang Bui and Waleed Youssef II-908
Efficient Clustering-Based Genetic Algorithms in Chemical Kinetic Modelling: Lionel Elliott, Derek Ingham, Adrian Kyne, Nicolae Mera, Mohamed Pourkashanian, and Sean Whittaker II-932
Automated Extraction of Problem Structure: Anthony Bucci, Jordan Pollack, and Edwin de Jong I-501

Similarities between Co-evolution and Learning Classifier Systems and Their Applications: Ramón Alfonsos Palacios-Durazo and Manuel Valenzuela-Rendón I-561

Feature Subset Selection, Class Separability, and Genetic Algorithms: Erick Cantú-Paz I-959

What Basis for Genetic Dynamics?: Chryssomalis Chryssomalakos and Christopher R. Stephens I-1018

Dependency Structure Matrix Analysis: Offline Utility of the Dependency Structure Matrix Genetic Algorithm: Tian-Li Yu and David Goldberg II-355

Distributed Constraint Satisfaction, Restricted Recombination, and Hybrid Genetic Search: Gerry Dozier, Hurley Cunningham, Winard Britt, and Funing Zhang I-1078

Mating Scheme for Controlling the Diversity-Convergence Balance for Multiobjective Optimization: Hisao Ishibuchi and Yohei Shibata I-1259

This page intentionally left blank

Table of Contents – Part II

Volume II

Genetic Algorithms (Continued)

PID Controller Tuning for Stable and Unstable Processes Applying GA 1
Marco Antonio Paz-Ramos, Jose Torres-Jimenez, Enrique Quintero-Marmol-Marquez, Hugo Estrada-Esquivel

Dynamic Uniform Scaling for Multiobjective Genetic Algorithms 11
Gerulf K.M. Pedersen, David E. Goldberg

Parameter-Less Hierarchical BOA 24
Martin Pelikan, Tz-Kai Lin

Computational Complexity and Simulation of Rare Events of Ising Spin Glasses 36
Martin Pelikan, Jiri Ocenasek, Simon Trebst, Matthias Troyer, Fabien Alet

Fitness Inheritance in the Bayesian Optimization Algorithm 48
Martin Pelikan, Kumara Sastry

Limit Cycle Prediction in Multivariable Nonlinear Systems Using Genetic Algorithms 60
Farzan Rashidi, Mehran Rashidi

Evolving Reusable Neural Modules 69
Joseph Reisinger, Kenneth O. Stanley, Risto Miikkulainen

How Are We Doing? Predicting Evolutionary Algorithm Performance 82
Mark A. Renslow, Brenda Hinkemeyer, Bryant A. Julstrom

Introduction of a New Selection Parameter in Genetic Algorithm for Constrained Reliability Design Problems 90
Laure Rigal, Bruno Castanier, Philippe Castagliola

Improving the Performance of a Genetic Algorithm Using a Variable-Reordering Algorithm 102
Eduardo Rodriguez-Tello, Jose Torres-Jimenez

Designing Competent Mutation Operators Via Probabilistic Model Building of Neighborhoods 114
Kumara Sastry, David E. Goldberg

Let's Get Ready to Rumble:
Crossover Versus Mutation Head to Head 126
Kumara Sastry, David E. Goldberg

Classification with Scaled Genetic Algorithms
in a Coevolutionary Setting 138
Lothar M. Schmitt

New Epistasis Measures for Detecting Independently Optimizable
Partitions of Variables 150
Dong-Il Seo, Sung-Soon Choi, Byung-Ro Moon

Clustering with Niching Genetic K-means Algorithm 162
Weiguo Sheng, Allan Tucker, Xiaohui Liu

A Comparison of Genetic Programming and Genetic Algorithms
in the Design of a Robust, Saturated Control System 174
Andrea Soltoggio

Upper Bounds on the Time and Space Complexity of Optimizing
Additively Separable Functions 186
Matthew J. Streeter

Winnowing Wheat from Chaff: The Chunking GA 198
Hal Stringer, Annie S. Wu

An Effective Chromosome Representation for Evolving
Flexible Job Shop Schedules 210
Joc Cing Tay, Djoko Wibowo

Linkage Identification by Nonlinearity Check
for Real-Coded Genetic Algorithms 222
Masaru Tezuka, Masaharu Munetomo, Kiyoshi Akama

Population-Based Iterated Local Search:
Restricting Neighborhood Search by Crossover 234
Dirk Thierens

Modeling Dependencies of Loci with String Classification
According to Fitness Differences 246
Miwako Tsuji, Masaharu Munetomo, Kiyoshi Akama

The Edge-Set Encoding Revisited:
On the Bias of a Direct Representation for Trees 258
Carsten Tzschoppe, Franz Rothlauf, Hans-Josef Pesch

A Gene Based Adaptive Mutation Strategy for Genetic Algorithms 271
Sima Uyar, Sanem Sariel, Gulsen Eryigit

Subthreshold-Seeking Behavior and Robust Local Search 282
Darrell Whitley, Keith Bush, Jonathan Rowe

Ruffled by Ridges: How Evolutionary Algorithms Can Fail 294
Darrell Whitley, Monte Lunacek, James Knight

Non-stationary Subtasks Can Improve Diversity
in Stationary Tasks 307
Christopher Willis-Ford, Terence Soule

The Shifting Balance Genetic Algorithm as More
than Just Another Island Model GA 318
Mark Wineberg, Jun Chen

Bistability of the Needle Function in the Presence
of Truncation Selection 330
Alden Wright, Greg Cripe

An Estimation of Distribution Algorithm
Based on Maximum Entropy 343
Alden Wright, Riccardo Poli, Chris Stephens, W.B. Langdon, Sandeep Pulavarty

Dependency Structure Matrix Analysis: Offline Utility
of the Dependency Structure Matrix Genetic Algorithm 355
Tian-Li Yu, David E. Goldberg

Toward an Understanding of the Quality and Efficiency
of Model Building for Genetic Algorithms 367
Tian-Li Yu, David E. Goldberg

Genetic Algorithms – Posters

Sexual and Asexual Paradigms in Evolution:
The Implications for Genetic Algorithms 379
Mark W. Andrews, Christopher Salzberg

Mutation Rates in the Context of Hybrid Genetic Algorithms 381
Seung-Hee Bae, Byung-Ro Moon

Systematic Integration of Parameterized Local Search Techniques
in Evolutionary Algorithms 383
Neal K. Bambha, Shuvra S. Bhattacharyya, Jürgen Teich, Eckart Zitzler

Comparative Molecular Binding Energy Analysis
of HIV-1 Protease Inhibitors Using Genetic Algorithm-Based
Partial Least Squares Method 385
Yen-Chih Chen, Jinn-Moon Yang, Chi-Hung Tsai, Cheng-Yan Kao

Controlled Content Crossover: A New Crossover Scheme
and Its Application to Optical Network
Component Allocation Problem 387
Mohammad Amin Dallaali, Malin Premaratne

Efficient and Reliable Evolutionary Multiobjective Optimization Using ε-Dominance Archiving and Adaptive Population Sizing 390
Venkat Devireddy, Patrick Reed

Heuristic Methods for Solving Euclidean Non-uniform Steiner Tree Problems 392
Ian Frommer, Bruce Golden, Guruprasad Pundoor

Automating Evolutionary Art in the Style of Mondrian 394
Andrés Gómez de Silva Garza, Aram Zamora Lores

Mutation Can Improve the Search Capability of Estimation of Distribution Algorithms 396
Hisashi Handa

Neural Network Normalization for Genetic Search 398
Jung-Hwan Kim, Sung-Soon Choi, Byung-Ro Moon

Distance Measures in Genetic Algorithms 400
Yong-Hyuk Kim, Byung-Ro Moon

Analysis of a Parallel MOEA Solving the Multi-objective Quadratic Assignment Problem 402
Mark P. Kleeman, Richard O. Day, Gary B. Lamont

Evolving Features in Neural Networks for System Identification 404
Yung-Keun Kwon, Byung-Ro Moon

A Bio-inspired Genetic Algorithm with a Self-Organizing Genome: The RBF-Gene Model 406
Virginie Lefort, Carole Knibbe, Guillaume Beslon, Joël Favrel

Evolving Spike-Train Processors 408
Juan Liu, Andrzej Buller

A Philosophical Essay on Life and Its Connections with Genetic Algorithms 410
Fernando G. Lobo

An Architecture for Massive Parallelization of the Compact Genetic Algorithm 412
Fernando G. Lobo, Cláudia F. Lima, Hugo Mártires

An Evolutionary Technique for Multicriterial Optimization Based on Endocrine Paradigm 414
Corina Rotar

Evolving Golomb Rulers 416
Jorge Tavares, Francisco B. Pereira, Ernesto Costa

Populating Genomes in a Dynamic Grid 418
Han Yu, Ning Jiang, Annie S. Wu

Empirical Study of Population Diversity
in Permutation-Based Genetic Algorithm 420
Kenny Q. Zhu, Ziwei Liu

Genetic Programming

A Demonstration of Neural Programming
Applied to Non-Markovian Problems 422
Gabriel Catalin Balan, Sean Luke

Evolving En-Route Caching Strategies for the Internet 434
Jürgen Branke, Pablo Funes, Frederik Thiele

Grammatical Constant Creation 447
Ian Dempsey, Michael O'Neill, Anthony Brabazon

Memetic Crossover for Genetic Programming:
Evolution Through Imitation 459
Brent E. Eskridge, Dean F. Hougen

Virtual Ramping of Genetic Programming Populations 471
Thomas Fernandez

Evolving Local Search Heuristics for SAT
Using Genetic Programming 483
Alex S. Fukunaga

Shortcomings with Tree-Structured Edge Encodings
for Neural Networks 495
Gregory S. Hornby

Adapting Representation in Genetic Programming 507
Cezary Z. Janikow

A Descriptive Encoding Language
for Evolving Modular Neural Networks 519
Jae-Yoon Jung, James A. Reggia

Run Transferable Libraries —
Learning Functional Bias in Problem Domains 531
Maarten Keijzer, Conor Ryan, Mike Cattolico

Using Genetic Programming to Obtain a Closed-Form Approximation
to a Recursive Function 543
Evan Kirshenbaum, Henri J. Suermondt

Comparison of Selection Strategies
for Evolutionary Quantum Circuit Design 557
André Leier, Wolfgang Banzhaf

Evolving Quantum Circuits and Programs
Through Genetic Programming 569
Paul Massey, John A. Clark, Susan Stepney

On Multi-class Classification by Way of Niching 581
A.R. McIntyre, M.I. Heywood

On the Strength of Size Limits in Linear Genetic Programming 593
Nicholas Freitag McPhee, Alex Jarvis, Ellery Fussell Crane

Softening the Structural Difficulty in Genetic Programming
with TAG-Based Representation and Insertion/Deletion Operators 605
Nguyen Xuan Hoai, R.I. McKay

πGrammatical Evolution 617
Michael O 'Neill, Anthony Brabazon, Miguel Nicolau,
Sean Mc Garraghy, Peter Keenan

Alternative Bloat Control Methods 630
Liviu Panait, Sean Luke

Robotic Control Using Hierarchical Genetic Programming 642
Marcin L. Pilat, Franz Oppacher

A Competitive Building Block Hypothesis 654
Conor Ryan, Hammad Majeed, Atif Azad

Dynamic Limits for Bloat Control (Variations on Size and Depth) 666
Sara Silva, Ernesto Costa

On Naïve Crossover Biases with Reproduction
for Simple Solutions to Classification Problems 678
M. David Terrio, Malcolm I. Heywood

Fitness Clouds and Problem Hardness in Genetic Programming 690
Leonardo Vanneschi, Manuel Clergue, Philippe Collard,
Marco Tomassini, Sébastien Vérel

Genetic Programming – Posters

Improving Generalisation Performance Through Multiobjective
Parsimony Enforcement 702
Yaniv Bernstein, Xiaodong Li, Vic Ciesielski, Andy Song

Using GP to Model Contextual Human Behavior 704
Hans Fernlund, Avelino J. Gonzalez

A Comparison of Hybrid Incremental Reuse Strategies for Reinforcement Learning in Genetic Programming 706
Scott Harmon, Edwin Rodríguez, Christopher Zhong, William Hsu

Humanoid Robot Programming Based on CBR Augmented GP 708
Hongwei Liu, Hitoshi Iba

Genetic Network Programming with Reinforcement Learning and Its Performance Evaluation 710
Shingo Mabu, Kotaro Hirasawa, Jinglu Hu

Multi-agent Cooperation Using Genetic Network Programming with Automatically Defined Groups 712
Tadahiko Murata, Takashi Nakamura

Chemical Genetic Programming – Coevolution Between Genotypic Strings and Phenotypic Trees 715
Wojciech Piaseczny, Hideaki Suzuki, Hidefumi Sawai

A Study of the Role of Single Node Mutation in Genetic Programming 717
Wei Quan, Terence Soule

Multi-branches Genetic Programming as a Tool for Function Approximation 719
Katya Rodríguez-Vázquez, Carlos Oliver-Morales

Hierarchical Breeding Control for Efficient Topology/Parameter Evolution 722
Kisung Seo, Jianjun Hu, Zhun Fan, Erik D. Goodman, Ronald C. Rosenberg

Keeping the Diversity with Small Populations Using Logic-Based Genetic Programming 724
Ken Taniguchi, Takao Terano

Learning Classifier Systems

Analysis and Improvements of the Adaptive Discretization Intervals Knowledge Representation 726
Jaume Bacardit, Josep Maria Garrell

Bounding Learning Time in XCS 739
Martin V. Butz, David E. Goldberg, Pier Luca Lanzi

Gradient-Based Learning Updates Improve XCS Performance in Multistep Problems 751
Martin V. Butz, David E. Goldberg, Pier Luca Lanzi

System Level Hardware–Software Design Exploration with XCS 763
Fabrizio Ferrandi, Pier Luca Lanzi, Donatella Sciuto

Parameter Adaptation within Co-adaptive
Learning Classifier Systems 774
Chung-Yuan Huang, Chuen-Tsai Sun

High Classification Accuracy Does Not Imply
Effective Genetic Search 785
Tim Kovacs, Manfred Kerber

Mixed Decision Trees:
Minimizing Knowledge Representation Bias in LCS 797
Xavier Llorà, Stewart W. Wilson

Improving MACS Thanks to a Comparison with 2TBNs 810
Olivier Sigaud, Thierry Gourdin, Pierre-Henri Wuillemin

Classifier Systems for Continuous Payoff Environments 824
Stewart W. Wilson

Learning Classifier Systems – Poster

Confidence and Support Classification Using Genetically
Programmed Neural Logic Networks..... 836
Henry Wai-Kit Chia, Chew-Lim Tan

Real World Applications

An Evolutionary Constraint Satisfaction Solution
for Over the Cell Channel Routing 838
Adnan Acan, Ahmet Unveren

Solution to the Fixed Airbase Problem
for Autonomous URAV Site Visitation Sequencing 850
Amit Agarwal, Meng-Hiot Lim, Chan Yee Chew,
Tong Kiang Poo, Meng Joo Er, Yew Kong Leong

Inflight Rerouting for an Unmanned Aerial Vehicle 859
Amit Agarwal, Meng-Hiot Lim, Maung Ye Win Kyaw,
Meng Joo Er

Memetic Optimization of Video Chain Designs..... 869
Walid Ali, Alexander Topchy

A Broad and Narrow Approach to Interactive Evolutionary Design –
An Aircraft Design Example 883
Oliver Bandte, Sergey Malinchik

Feature Synthesis Using Genetic Programming for Face Expression Recognition 896
Bir Bhanu, Jiangang Yu, Xuejun Tan, Yingqiang Lin

An Enhanced Genetic Algorithm for DNA Sequencing by Hybridization with Positive and Negative Errors 908
Thang N. Bui, Waleed A. Youssef

Unveiling Optimal Operating Conditions for an Epoxy Polymerization Process Using Multi-objective Evolutionary Computation 920
Kalyanmoy Deb, Kishalay Mitra, Rinku Dewri, Saptarshi Majumdar

Efficient Clustering-Based Genetic Algorithms in Chemical Kinetic Modelling 932
Lionel Elliott, Derek B. Ingham, Adrian G. Kyne, Nicolae S. Mera, Mohamed Pourkashanian, Sean Whittaker

An Informed Operator Based Genetic Algorithm for Tuning the Reaction Rate Parameters of Chemical Kinetics Mechanisms 945
Lionel Elliott, Derek B. Ingham, Adrian G. Kyne, Nicolae S. Mera, Mohamed Pourkashanian, Christopher W. Wilson

Transfer of Neuroevolved Controllers in Unstable Domains 957
Faustino J. Gomez, Risto Miikkulainen

Evolving Wavelets Using a Coevolutionary Genetic Algorithm and Lifting 969
Uli Grasemann, Risto Miikkulainen

Optimization of Constructive Solid Geometry Via a Tree-Based Multi-objective Genetic Algorithm 981
Karim Hamza, Kazuhiro Saitou

Co-evolutionary Agent Self-Organization for City Traffic Congestion Modeling 993
Luis Miramontes Hercog

Validating a Model of Colon Colouration Using an Evolution Strategy with Adaptive Approximations 1005
Džena Hidović, Jonathan E. Rowe

Evolution-Based Deliberative Planning for Cooperating Unmanned Ground Vehicles in a Dynamic Environment 1017
Talib Hussain, David Montana, Gordon Vidaver

Optimized Design of MEMS by Evolutionary Multi-objective Optimization with Interactive Evolutionary Computation 1030
Raffi Kamalian, Hideyuki Takagi, Alice M. Agogino

Hybrid Genetic Algorithms for Multi-objective Optimisation of Water Distribution Networks 1042
Edward Keedwell, Soon-Thiam Khu

A Hybrid Genetic Approach for Circuit Bipartitioning 1054
Jong-Pil Kim, Yong-Hyuk Kim, Byung-Ro Moon

Lagrange Multiplier Method for Multi-campaign Assignment Problem 1065
Yong-Hyuk Kim, Byung-Ro Moon

Biomass Inferential Sensor Based on Ensemble of Models Generated by Genetic Programming 1078
Arthur Kordon, Elsa Jordaan, Lawrence Chew, Guido Smits, Torben Bruck, Keith Haney, Annika Jenings

CellNet Co-Ev: Evolving Better Pattern Recognizers Using Competitive Co-evolution 1090
Taras Kowaliw, Nawwaf Kharma, Chris Jensen, Hussein Moghnieh, Jie Yao

Evolutionary Ensemble for Stock Prediction 1102
Yung-Keun Kwon, Byung-Ro Moon

Discovery of Human-Competitive Image Texture Feature Extraction Programs Using Genetic Programming 1114
Brian Lam, Vic Ciesielski

Evolutionary Drug Scheduling Model for Cancer Chemotherapy 1126
Yong Liang, Kwong-Sak Leung, Tony Shu Kam Mok

An Island-Based GA Implementation for VLSI Standard-Cell Placement 1138
Guangfa Lu, Shawki Areibi

Exploratory Data Analysis with Interactive Evolution 1151
Sergey Malinchik, Eric Bonabeau

Designing Multiplicative General Parameter Filters Using Adaptive Genetic Algorithms 1162
Jarno Martikainen, Seppo J. Ovaska

Reducing the Cost of the Hybrid Evolutionary Algorithm with Image Local Response in Electronic Imaging 1177
Igor V. Maslov

The Lens Design Using the CMA-ES Algorithm 1189
Yuichi Nagata

Automatic Synthesis of an 802.11a Wireless LAN Antenna
Using Genetic Programming A Real World Application 1201
Rian Sanderson

A Generic Network Design for a Closed-Loop Supply Chain
Using Genetic Algorithm 1214
Eoksu Sim, Sungwon Jung, Haejoong Kim, Jinwoo Park

Evolving a Roving Eye for Go 1226
Kenneth O. Stanley, Risto Miikkulainen

Comparing Discrete and Continuous Genotypes on the Constrained
Portfolio Selection Problem 1239
Felix Streichert, Holger Ulmer, Andreas Zell

Learning Environment for Life Time Value Calculation
of Customers in Insurance Domain 1251
*Andrea Tettamanzi, Luca Sammartino, Mikhail Simonov,
Massimo Soroldoni, Mauro Beretta*

Multiple Species Weighted Voting –
A Genetics-Based Machine Learning System 1263
Alexander F. Tulai, Franz Oppacher

Object Oriented Design and Implementation
of a General Evolutionary Algorithm 1275
Róbert Ványi

Generating Multiaxis Tool Paths for Die and Mold Making
with Evolutionary Algorithms 1287
Klaus Weinert, Marc Stautner

Real World Applications – Posters

Tackling an Inverse Problem from the Petroleum Industry
with a Genetic Algorithm for Sampling 1299
Pedro J. Ballester, Jonathan N. Carter

A Genetic Approach for Generating Good
Linear Block Error-Correcting Codes 1301
Alan Barbieri, Stefano Cagnoni, Giulio Colavolpe

Genetic Fuzzy Discretization for Classification Problems 1303
Yoon-Seok Choi, Byung-Ro Moon

A Genetic Algorithm for the Shortest Common Superstring Problem 1305
Luis C. González, Heidi J. Romero, Carlos A. Brizuela

A Genetic Algorithm to Improve Agent-Oriented Natural Language Interpreters 1307
Babak Hodjat, Junichi Ito, Makoto Amamiya

Optimization of Gaussian Mixture Model Parameters for Speaker Identification 1310
Q. Y. Hong, Sam Kwong, H.L. Wang

Network Intrusion Detection Using Genetic Clustering 1312
Elizabeth Leon, Olfa Nasraoui, Jonatan Gomez

Enhanced Innovation: A Fusion of Chance Discovery and Evolutionary Computation to Foster Creative Processes and Decision Making 1314
Xavier Llorà, Kei Ohnishi, Ying-ping Chen, David E. Goldberg, Michael E. Welge

Development of a Genetic Algorithm for Optimization of Nanoalloys 1316
Lesley D. Lloyd, Roy L. Johnston, Said Salhi

Empirical Performance Evaluation of a Parameter-Free GA for JSSP 1318
Shouichi Matsui, Isamu Watanabe, Ken-ichi Tokoro

A Caching Genetic Algorithm for Spectral Breakpoint Matching 1320
Jonathan Mohr, Xiaobo Li

Multi-agent Simulation of Airline Travel Markets 1322
Rashad L. Moore, Ashley Williams, John Sheppard

Improved Niching and Encoding Strategies for Clustering Noisy Data Sets 1324
Olfa Nasraoui, Elizabeth Leon

A Multi-objective Approach to Configuring Embedded System Architectures 1326
James Northern, Michael Shanblatt

Achieving Shorter Search Times in Voice Conversion Using Interactive Evolution 1328
Yuji Sato

Predicting Healthcare Costs Using Classifiers 1330
C.R. Stephens, H. Waelbroeck, S. Talley, R. Cruz, A.S. Ash

Generating Compact Rough Cluster Descriptions Using an Evolutionary Algorithm 1332
Kevin Voges, Nigel Pope

An Evolutionary Meta Hierarchical Scheduler
for the Linux Operating System 1334
Horst F. Wedde, Muddassar Farooq, Mario Lischka

An Evolutionary Algorithm for Parameters Identification
in Parabolic Systems 1336
Zhijian Wu, Zhilong Tang, Jun Zou, Lishan Kang, Mingbiao Li

Search-Based Software Engineering

How to Overcome the Equivalent Mutant Problem
and Achieve Tailored Selective Mutation Using Co-evolution 1338
Konstantinos Adamopoulos, Mark Harman, Robert M. Hierons

Evaluating Evolutionary Testability with Software-Measurements 1350
Frank Lammermann, André Baresel, Joachim Wegener

Hybridizing Evolutionary Testing with the Chaining Approach 1363
Phil McMinn, Mike Holcombe

Using Interconnection Style Rules
to Infer Software Architecture Relations 1375
Brian S. Mitchell, Spiros Mancoridis, Martin Traverso

Finding Effective Software Metrics to Classify Maintainability
Using a Parallel Genetic Algorithm 1388
Rodrigo Vivanco, Nicolino Pizzi

Evaluation of Different Fitness Functions
for the Evolutionary Testing of an Autonomous Parking System 1400
Joachim Wegener, Oliver Bühler

Search Based Automatic Test-Data Generation
at an Architectural Level 1413
Yuan Zhan, John Clark

Search-Based Software Engineering – Posters

Search-Based Techniques for Optimizing
Software Project Resource Allocation 1425
G. Antoniol, M. Di Penta, M. Harman

Applying Evolutionary Testing to Search for Critical Defects 1427
André Baresel, Harmen Sthamer, Joachim Wegener

Input Sequence Generation for Testing
of Communicating Finite State Machines (CFSMs) 1429
Karnig Derderian, Robert M. Hierons, Mark Harman, Qiang Guo

TDSGen: An Environment Based on Hybrid Genetic Algorithms for Generation of Test Data 1431
Luciano Petinati Ferreira, Silvia Regina Vergilio

Author Index 1433

Table of Contents – Part I

Volume I

A-Life, Adaptive Behavior, Agents, and Ant Colony Optimization

Efficient Evaluation Functions for Multi-rover Systems 1
Adrian Agogino, Kagan Tumer

A Particle Swarm Model of Organizational Adaptation 12
Anthony Brabazon, Arlindo Silva, Tiago Ferra de Sousa, Michael O'Neill, Robin Matthews, Ernesto Costa

Finding Maximum Cliques with Distributed Ants 24
Thang N. Bui, Joseph R. Rizzo, Jr.

Ant System for the k-Cardinality Tree Problem 36
Thang N. Bui, Gnanasekaran Sundarraj

A Hybrid Ant Colony Optimisation Technique for Dynamic Vehicle Routing 48
Darren M. Chitty, Marcel L. Hernandez

Cooperative Problem Solving Using an Agent-Based Market 60
David Cornforth, Michael Kirley

Cultural Evolution for Sequential Decision Tasks: Evolving Tic–Tac–Toe Players in Multi–agent Systems 72
Dara Curran, Colm O'Riordan

Artificial Life and Natural Intelligence 81
Keith L. Downing

Bluenome: A Novel Developmental Model of Artificial Morphogenesis 93
T. Kowaliw, P. Grogono, N. Kharma

Adaptively Choosing Neighbourhood Bests Using Species in a Particle Swarm Optimizer for Multimodal Function Optimization 105
Xiaodong Li

Better Spread and Convergence: Particle Swarm Multiobjective Optimization Using the Maximin Fitness Function 117
Xiaodong Li

Evolving a Self-Repairing, Self-Regulating, French Flag Organism 129
Julian Francis Miller

The Kalman Swarm (A New Approach to Particle Motion in Swarm Optimization) 140
Christopher K. Monson, Kevin D. Seppi

Adaptive and Evolvable Network Services 151
Tadashi Nakano, Tatsuya Suda

Grammatical Swarm 163
Michael O'Neill, Anthony Brabazon

A New Universal Cellular Automaton Discovered by Evolutionary Algorithms 175
Emmanuel Sapin, Olivier Bailleux, Jean-Jacques Chabrier, Pierre Collet

An Interactive Artificial Ant Approach to Non-photorealistic Rendering 188
Yann Semet, Una-May O'Reilly, Frédo Durand

Automatic Creation of Team-Control Plans Using an Assignment Branch in Genetic Programming 201
Walter A. Talbott

Implications of Epigenetic Learning Via Modification of Histones on Performance of Genetic Programming 213
Ivan Tanev, Kikuo Yuta

Using Clustering Techniques to Improve the Performance of a Multi-objective Particle Swarm Optimizer 225
Gregorio Toscano Pulido, Carlos A. Coello Coello

SWAF: Swarm Algorithm Framework for Numerical Optimization 238
Xiao-Feng Xie, Wen-Jun Zhang

A-Life, Adaptive Behavior, Agents, and Ant Colony Optimization – Posters

Autonomous Agent for Multi-objective Optimization 251
Alain Berro, Stephane Sanchez

An Evolved Autonomous Controller for Satellite Task Scheduling 253
Darren M. Chitty

Multi-agent Foreign Exchange Market Modelling Via GP 255
Stephen Dignum, Riccardo Poli

An Evolutionary Autonomous Agent with Visual Cortex and Recurrent Spiking Columnar Neural Network 257
Rich Drewes, James Maciokas, Sushil J. Louis, Philip Goodman

Arguments for ACO's Success 259
Osvaldo Gómez, Benjamín Barán

Solving Engineering Design Problems by Social Cognitive Optimization 261
Xiao-Feng Xie, Wen-Jun Zhang

Artificial Immune Systems

Vulnerability Analysis of Immunity-Based Intrusion Detection Systems Using Evolutionary Hackers 263
Gerry Dozier, Douglas Brown, John Hurley, Krystal Cain

Constructing Detectors in Schema Complementary Space for Anomaly Detection 275
Xiaoshu Hang, Honghua Dai

Real-Valued Negative Selection Algorithm with Variable-Sized Detectors..... 287
Zhou Ji, Dipankar Dasgupta

An Investigation of R-Chunk Detector Generation on Higher Alphabets 299
Thomas Stibor, Kpatscha M. Bayarou, Claudia Eckert

A Comment on Opt-AiNET: An Immune Network Algorithm for Optimisation 308
Jon Timmis, Camilla Edmonds

Artificial Immune Systems – Posters

A Novel Immune Feedback Control Algorithm and Its Applications..... 318
Zhen-qiang Qi, Shen-min Song, Zhao-hua Yang, Guang-da Hu, Fu-en Zhang

Biological Applications

Computer-Aided Peptide Evolution for Virtual Drug Design..... 321
Ignasi Belda, Xavier Llorà, Marc Martinell, Teresa Tarragó, Ernest Giralt

Automating Genetic Network Inference with Minimal Physical Experimentation Using Coevolution 333
Josh C. Bongard, Hod Lipson

A Genetic Approach for Gene Selection on Microarray Expression Data 346
Yong-Hyuk Kim, Su-Yeon Lee, Byung-Ro Moon

Fuzzy Dominance Based Multi-objective GA-Simplex Hybrid Algorithms Applied to Gene Network Models 356
Praveen Koduru, Sanjoy Das, Stephen Welch, Judith L. Roe

Selection-Insertion Schemes in Genetic Algorithms for the Flexible Ligand Docking Problem 368
Camila S. de Magalhães, Helio J.C. Barbosa, Laurent E. Dardenne

A GA Approach to the Definition of Regulatory Signals in Genomic Sequences 380
Giancarlo Mauri, Roberto Mosca, Giulio Pavesi

Systems Biology Modeling in Human Genetics Using Petri Nets and Grammatical Evolution 392
Jason H. Moore, Lance W. Hahn

Evolutionary Computation Techniques for Optimizing Fuzzy Cognitive Maps in Radiation Therapy Systems 402
K.E. Parsopoulos, E.I. Papageorgiou, P.P. Groumpos, M.N. Vrahatis

Identification of Informative Genes for Molecular Classification Using Probabilistic Model Building Genetic Algorithm 414
Topon Kumar Paul, Hitoshi Iba

GA-Facilitated Knowledge Discovery and Pattern Recognition Optimization Applied to the Biochemistry of Protein Solvation 426
Michael R. Peterson, Travis E. Doom, Michael L. Raymer

Genetic Programming Neural Networks as a Bioinformatics Tool for Human Genetics 438
Marylyn D. Ritchie, Christopher S. Coffey, Jason H. Moore

Evolving Better Multiple Sequence Alignments 449
Luke Sheneman, James A. Foster

Optimizing Topology and Parameters of Gene Regulatory Network Models from Time-Series Experiments 461
Christian Spieth, Felix Streichert, Nora Speer, Andreas Zell

Comparing Genetic Programming and Evolution Strategies on Inferring Gene Regulatory Networks 471
Felix Streichert, Hannes Planatscher, Christian Spieth, Holger Ulmer, Andreas Zell

An Evolutionary Approach with Pharmacophore-Based Scoring Functions for Virtual Database Screening 481
Jinn-Moon Yang, Tsai-Wei Shen, Yen-Fu Chen, Yi-Yuan Chiu

Biological Applications – Posters

Statistical Test-Based Evolutionary Segmentation of Yeast Genome 493
Jesus S. Aguilar–Ruiz, Daniel Mateos, Raul Giraldez, Jose C. Riquelme

Equilibrium and Extinction in a Trisexual Diploid Mating System: An Investigation 495
Erik C. Buehler, Sanjoy Das, Jack F. Cully, Jr.

On Parameterizing Models of Antigen-Antibody Binding Dynamics on Surfaces – A Genetic Algorithm Approach and the Need for Speed ... 497
Daniel J. Burns, Kevin T. May

Is the Predicted ESS in the Sequential Assessment Game Evolvable? 499
Winfried Just, Xiaolu Sun

Coevolution

Automated Extraction of Problem Structure 501
Anthony Bucci, Jordan B. Pollack, Edwin de Jong

Modeling Coevolutionary Genetic Algorithms on Two-Bit Landscapes: Random Partnering 513
Ming Chang, Kazuhiro Ohkura, Kanji Ueda, Masaharu Sugiyama

The Incremental Pareto-Coevolution Archive 525
Edwin D. de Jong

A Cooperative Coevolutionary Multiobjective Algorithm Using Non-dominated Sorting 537
Antony W. Iorio, Xiaodong Li

Predicting Genetic Drift in 2 × 2 Games 549
Anthony M.L. Liekens, Huub M.M. ten Eikelder, Peter A.J. Hilbers

Similarities Between Co-evolution and Learning Classifier Systems and Their Applications 561
Ramón Alfonso Palacios-Durazo, Manuel Valenzuela-Rendón

A Sensitivity Analysis of a Cooperative Coevolutionary Algorithm Biased for Optimization 573
Liviu Panait, R. Paul Wiegand, Sean Luke

Coevolution – Posters

A Population-Differential Method of Monitoring Success and Failure in Coevolution 585
Ari Bader-Natal, Jordan B. Pollack

Cooperative Coevolution Fusion for Moving Object Detection 587
Sohail Nadimi, Bir Bhanu

Evolutionary Robotics

Learning to Acquire Autonomous Behavior
— Cooperation by Humanoid Robots — 590
Yutaka Inoue, Takahiro Tohge, Hitoshi Iba

Evolved Motor Primitives and Sequences
in a Hierarchical Recurrent Neural Network 603
Rainer W. Paine, Jun Tani

Robot Trajectory Planning Using Multi-objective
Genetic Algorithm Optimization 615
E. J. Solteiro Pires, J.A. Tenreiro Machado, P.B. de Moura Oliveira

Evolution, Robustness, and Adaptation of Sidewinding Locomotion
of Simulated Snake-Like Robot 627
Ivan Tanev, Thomas Ray, Andrzej Buller

Evolutionary Robotics – Poster

Evolution Tunes Coevolution: Modelling Robot Cognition Mechanisms... 640
Michail Maniadakis, Panos Trahanias

Evolution Strategies/Evolutionary Programming

On the Complexity to Approach Optimum Solutions
by Inhomogeneous Markov Chains 642
Andreas A. Albrecht

Actuator Noise in Recombinant Evolution Strategies
on General Quadratic Fitness Models..... 654
Hans-Georg Beyer

Convergence Examples of a Filter-Based Evolutionary Algorithm..... 666
Lauren M. Clevenger, William E. Hart

Node-Depth Encoding for Evolutionary Algorithms
Applied to Network Design..... 678
*A.C.B. Delbem, Andre de Carvalho, Claudio A. Policastro,
Adriano K. O. Pinto, Karen Honda, Anderson C. Garcia*

Reducing Fitness Evaluations Using Clustering Techniques
and Neural Network Ensembles..... 688
Yaochu Jin, Bernhard Sendhoff

An Improved Diversity Mechanism for Solving Constrained
Optimization Problems Using a Multimembered Evolution Strategy 700
Efrén Mezura-Montes, Carlos A. Coello Coello

Randomized Local Search, Evolutionary Algorithms, and the Minimum Spanning Tree Problem 713
Frank Neumann, Ingo Wegener

An Evolution Strategy Using a Continuous Version of the Gray-Code Neighbourhood Distribution 725
Jonathan E. Rowe, Džena Hidović

A Novel Multi-objective Orthogonal Simulated Annealing Algorithm for Solving Multi-objective Optimization Problems with a Large Number of Parameters 737
Li-Sun Shu, Shinn-Jang Ho, Shinn-Ying Ho, Jian-Hung Chen, Ming-Hao Hung

On the Choice of the Population Size 748
Tobias Storch

An Analysis of the $(\mu+1)$ EA on Simple Pseudo-Boolean Functions 761
Carsten Witt

Program Evolution by Integrating EDP and GP 774
Kohsuke Yanai, Hitoshi Iba

Evolution Strategies/Evolutionary Programming – Posters

A Step Size Preserving Directed Mutation Operator 786
Stefan Berlik

A Comparison of Several Algorithms and Representations for Single Objective Optimization 788
Crina Grosan

Towards a Generally Applicable Self-Adapting Hybridization of Evolutionary Algorithms 790
Wilfried Jakob, Christian Blume, Georg Bretthauer

Evolvable Hardware

High Temperature Experiments for Circuit Self-Recovery 792
Didier Keymeulen, Ricardo Zebulum, Vu Duong, Xin Guo, Ian Ferguson, Adrian Stoica

The Emergence of Ontogenic Scaffolding in a Stochastic Development Environment 804
John Rieffel, Jordan Pollack

A Reconfigurable Chip for Evolvable Hardware 816
Yann Thoma, Eduardo Sanchez

Genetic Algorithms

Experimental Evaluation of Discretization Schemes for Rule Induction ... 828
Jesus Aguilar–Ruiz, Jaume Bacardit, Federico Divina

Real-Coded Bayesian Optimization Algorithm: Bringing the Strength of BOA into the Continuous World ... 840
Chang Wook Ahn, R.S. Ramakrishna, David E. Goldberg

Training Neural Networks with GA Hybrid Algorithms ... 852
Enrique Alba, J. Francisco Chicano

Growth Curves and Takeover Time in Distributed Evolutionary Algorithms ... 864
Enrique Alba, Gabriel Luque

Simultaneity Matrix for Solving Hierarchically Decomposable Functions ... 877
Chatchawit Aporntewan, Prabhas Chongstitvatana

Metaheuristics for Natural Language Tagging ... 889
Lourdes Araujo, Gabriel Luque, Enrique Alba

An Effective Real-Parameter Genetic Algorithm with Parent Centric Normal Crossover for Multimodal Optimisation ... 901
Pedro J. Ballester, Jonathan N. Carter

Looking Under the EA Hood with Price's Equation ... 914
Jeffrey K. Bassett, Mitchell A. Potter, Kenneth A. De Jong

Distribution of Evolutionary Algorithms in Heterogeneous Networks ... 923
Jürgen Branke, Andreas Kamper, Hartmut Schmeck

A Statistical Model of GA Dynamics for the OneMax Problem ... 935
Bulent Buyukbozkirli, Erik D. Goodman

Adaptive Sampling for Noisy Problems ... 947
Erick Cantú-Paz

Feature Subset Selection, Class Separability, and Genetic Algorithms ... 959
Erick Cantú-Paz

Introducing Subchromosome Representations to the Linkage Learning Genetic Algorithm ... 971
Ying-ping Chen, David E. Goldberg

Interactive One-Max Problem Allows to Compare the Performance of Interactive and Human-Based Genetic Algorithms ... 983
Chihyung Derrick Cheng, Alexander Kosorukoff

Polynomial Approximation of Survival Probabilities Under Multi-point Crossover 994
Sung-Soon Choi, Byung-Ro Moon

Evolving Genotype to Phenotype Mappings with a Multiple-Chromosome Genetic Algorithm 1006
Rick Chow

What Basis for Genetic Dynamics? 1018
Chryssomalis Chryssomalakos, Christopher R. Stephens

Exploiting Modularity, Hierarchy, and Repetition in Variable-Length Problems 1030
Edwin D. de Jong, Dirk Thierens

Optimal Operating Conditions for Overhead Crane Maneuvering Using Multi-objective Evolutionary Algorithms 1042
Kalyanmoy Deb, Naveen Kumar Gupta

Efficiently Solving: A Large-Scale Integer Linear Program Using a Customized Genetic Algorithm 1054
Kalyanmoy Deb, Koushik Pal

Using a Genetic Algorithm to Design and Improve Storage Area Network Architectures 1066
Elizabeth Dicke, Andrew Byde, Paul Layzell, Dave Cliff

Distributed Constraint Satisfaction, Restricted Recombination, and Hybrid Genetic Search 1078
Gerry Dozier, Hurley Cunningham, Winard Britt, Funing Zhang

Analysis of the (1 + 1) EA for a Noisy ONEMAX 1088
Stefan Droste

A Polynomial Upper Bound for a Mutation-Based Algorithm on the Two-Dimensional Ising Model 1100
Simon Fischer

The Ising Model on the Ring: Mutation Versus Recombination 1113
Simon Fischer, Ingo Wegener

Effects of Module Encapsulation in Repetitively Modular Genotypes on the Search Space 1125
Ivan I. Garibay, Ozlem O. Garibay, Annie S. Wu

Modeling Selection Intensity for Toroidal Cellular Evolutionary Algorithms 1138
Mario Giacobini, Enrique Alba, Andrea Tettamanzi, Marco Tomassini

Evolution of Fuzzy Rule Based Classifiers 1150
Jonatan Gomez

Self Adaptation of Operator Rates in Evolutionary Algorithms 1162
Jonatan Gomez

PolyEDA: Combining Estimation of Distribution Algorithms and Linear Inequality Constraints 1174
Jörn Grahl, Franz Rothlauf

Improving the Locality Properties of Binary Representations 1186
Adrian Grajdeanu, Kenneth De Jong

Schema Disruption in Chromosomes That Are Structured as Binary Trees 1197
William A. Greene

The Royal Road Not Taken: A Re-examination of the Reasons for GA Failure on R1 1208
Brian Howard, John Sheppard

Robust and Efficient Genetic Algorithms with Hierarchical Niching and a Sustainable Evolutionary Computation Model 1220
Jianjun Hu, Erik Goodman

A Systematic Study of Genetic Algorithms with Genotype Editing 1233
Chien-Feng Huang, Luis M. Rocha

Some Issues on the Implementation of Local Search in Evolutionary Multiobjective Optimization 1246
Hisao Ishibuchi, Kaname Narukawa

Mating Scheme for Controlling the Diversity-Convergence Balance for Multiobjective Optimization 1259
Hisao Ishibuchi, Youhei Shibata

Encoding Bounded-Diameter Spanning Trees with Permutations and with Random Keys 1272
Bryant A. Julstrom

Three Evolutionary Codings of Rectilinear Steiner Arborescences 1282
Bryant A. Julstrom, Athos Antoniades

Central Point Crossover for Neuro-genetic Hybrids 1292
Soonchul Jung, Byung-Ro Moon

Combining a Memetic Algorithm with Integer Programming to Solve the Prize-Collecting Steiner Tree Problem 1304
Gunnar W. Klau, Ivana Ljubić, Andreas Moser, Petra Mutzel, Philipp Neuner, Ulrich Pferschy, Günther Raidl, René Weiskircher

On the Evolution of Analog Electronic Circuits Using Building Blocks on a CMOS FPTA 1316
Jörg Langeheine, Martin Trefzer, Daniel Brüderle, Karlheinz Meier, Johannes Schemmel

Parameter-Less Optimization with the Extended Compact Genetic Algorithm and Iterated Local Search 1328
Cláudio F. Lima, Fernando G. Lobo

Comparing Search Algorithms for the Temperature Inversion Problem ... 1340
Monte Lunacek, Darrell Whitley, Philip Gabriel, Graeme Stephens

Inequality's Arrow: The Role of Greed and Order in Genetic Algorithms 1352
Anil Menon

Trap Avoidance in Strategic Computer Game Playing with Case Injected Genetic Algorithms 1365
Chris Miles, Sushil J. Louis, Rich Drewes

Topological Interpretation of Crossover 1377
Alberto Moraglio, Riccardo Poli

Simple Population Replacement Strategies for a Steady-State Multi-objective Evolutionary Algorithm 1389
Christine L. Mumford

Dynamic and Scalable Evolutionary Data Mining: An Approach Based on a Self-Adaptive Multiple Expression Mechanism 1401
Olfa Nasraoui, Carlos Rojas, Cesar Cardona

Crossover, Population Dynamics, and Convergence in the GAuGE System 1414
Miguel Nicolau, Conor Ryan

Inducing Sequentiality Using Grammatical Genetic Codes 1426
Kei Ohnishi, Kumara Sastry, Ying-ping Chen, David E. Goldberg

Author Index 1439

PID Controller Tuning for Stable and Unstable Processes Applying GA

Marco Antonio Paz-Ramos[1,2], Jose Torres-Jimenez[3],
Enrique Quintero-Marmol-Marquez[2], and Hugo Estrada-Esquivel[2]

[1] Instituto Tecnologico y de Estudios Superiores de Monterrey
Campus Ciudad de Mexico Calle del Puente 222 Col. Ejidos de Huipulco
Tlalpan, 14380, Mexico, D.F. marco.paz@itesm.mx
[2] Centro Nacional de Investigación y Desarrollo Tecnológico (CENIDET)
Interior Internado Palmira S/N Cuernavaca, Morelos, Mexico
{pazramos,eqm,hestrada}@cenidet.edu.mx
[3] Instituto Tecnologico y de Estudios Superiores de Monterrey
Campus Cuernavaca, Computer Science Department.
Av. Paseo de la Reforma 182-A. Lomas de Cuernavaca 62589 Temixco Morelos, MEXICO
jtj@itesm.mx
Tel. +52-777-3297169, Fax +52-777-3297169

Abstract. During the last years the use of intelligent strategies for tuning Proportional-Integral-Derivative (PID) controllers has been growing. The evolutionary strategies have won an important place thanks to their flexibility. In this paper, the automatic tuning of systems with stable and unstable dynamics, through a genetic approach is presented. The advantages of the proposed approach ere highlighted through the comparison with the Ziegler-Nichols modified closed loop method, and the Visioli genetic approach. The proposed methodology goal is to expand the intelligent tuning application to a wider range of processes (covering systems with oscillatory or unstable modes).

Keywords: PID Controllers, Genetic Algorithms, Ziegler-Nichols method, Visioli method.

1 Introduction

The first attempts to automate the tuning of Proportional-Integral-Derivative (PID) controllers were based on the time response of a process, but this approach had the drawback of requiring a lot of user interaction. A very important advance was made when it was decided to use the frequency response of a process instead of its time response, in this way a bigger degree of automation was attained. The first industrial PID controller based on processes' frequency response, was the Novatune [1], which initialized a PID controller automatically.

Applications based on PID controllers using frequency response were very successful [1] [2] [3], however the range of their applicability is restricted to

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1–10, 2004.

asymptotically stable systems, due to the use of the relay feedback. By using evolutionary meta heuristics, like Genetic Algorithms (GA), it is possible to expand the application range of PID controllers to systems with oscillatory or unstable modes. The application of GA to control applications has been increasing steadily and recently GA were applied: a) to tune Multiple-Input-Multiple-Output (MIMO) systems [4] [5]; and b) to improve significantly the performance of plants [6].

In this paper, a GA was used to automate the tuning of systems with scarce initial information and integrative and unstable dynamics, in order to get insight of the advantages of using a GA, the results were compared against the Ziegler-Nichols modified closed loop method.

The rest of the paper is organized in four more sections: a) in section two the Ziegler-Nichols closed loop control method is presented; b) in section 3 the Visioli genetic method for integral and unstable processes was highlighted; c) in section 4 the Direct Genetic PID proposed method, that is able to complement the Ziegler-Nichols and Visioli methods is presented; and d) finally in section 5 main conclusions will be presented.

2 Ziegler-Nichols Closed Loop Method

With the microprocessor arrival, the industrial field controllers' horizons were expanded, thanks to the capacity to carry out elaborated algorithms. This situation encouraged some controller makers to develop auto tuning PID controllers. One example of this controllers is the Foxboro EXACT [7], whose operation was based on the transient response analysis. An alternative was the use of Ziegler and Nichols closed-loop method [7]. The original method consists of varying the proportional gain until a sustained oscillation is reached. Maintaining the system in this state, the amplitude and frequency of the oscillations are determined by measuring the system output. A PID controller has the standard form depicted in equation 1, where $u(t)$ defines the controller output, $e(t)$ is the system error (difference among the desired set point for the variable under control, and the actual process variable value), K_p indicates the proportional gain, T_d is the derivative time constant, and T_i is the integral time constant.

$$u(t) = K_p \left(e(t) + \frac{1}{T_i} \int_0^t e(t)dt + T_d \frac{de(t)}{dt} \right) \tag{1}$$

The PID controller parameters for equation 1 are calculated and fixed according to the table 1, where K_u is the ultimate gain, and T_u is the ultimate period. The Ziegler and Nichols method for tuning an ISA PID controller, is a thumb rule based on the Nyquist curve. According to the Nyquist theorem a process is stable if the Nyquist curve does not encircle the point (-1,0).

Table 1. Regulator parameters obtained by the Ziegler-Nichols closed-loop method

Controller	K_p	T_i	T_d
P	$0.5K_u$		
PI	$0.4K_u$	$0.8T_u$	
PID	$0.6K_u$	$0.5T_u$	$0.12T_u$

2.1 Using the Relay Feedback System

Hägglund and Åström [1] [9] introduced an alternative to the increment of the proportional gain to achieve the oscillations inside a bigger margin of security that the original free oscillations system. A non linear circuit was introduced. The non linear function has a relay characteristic. To obtain the value of the proportional gain according to the table 1 is necessary to calculate the equation 2 where d is the output relay amplitude and a is the oscillation amplitude, while T_u (necessary to obtain T_i and T_d) is measured directly of the oscillations.

$$K_u = \frac{4d}{a\pi} \tag{2}$$

2.2 Stability of Limit Cycle

The relay feedback method is successfully used in the industry [1], however to measure K_u and T_u is fundamental that the limit cycle were stable and symmetric. Åström developed an analysis consisting in obtaining the Jacobian of the Poincaré map (W) which is related to the relay oscillations; the limit cycle is locally stable if and only if W has all its eigenvalues inside the unit disk [8].The phase portrait is a practical way to know if a process oscillation is stable or not, when the relay feedback is applied.

Generally the integrative and unstable processes do not allow reaching a controlled oscillation. If do not exist a stable oscillation it is not possible to apply the Ziegler and Nichols methods, one of the alternatives to tuning PID controllers for integral and unstable processes is the use of GA. The Visioli method [13] presented in next section uses internally a GA.

3 Visioli Genetic Method for Integral and Unstable Processes

In this section, a brief introduction to GA concepts will be presented first, and then the Visioli method (based on GA) is detailed. John Holland proposed in 1975 the genetic algorithms [11]. The original idea was to achieve that the computing processes could have an evolutionary profile. In the last years there has been an important increase of genetic algorithms applications to the automatic

control field. A genetic algorithm emulates an evolutionary process [12], thanks to a generation of virtual individuals population. The virtual population has a reproduction and death process, which allow space for a new generation. According to the schemata theorem [12] if reproduction possibilities for the better adapted individual are increased, the future generations will be more capable on the average than their predecessors. If we take the population individuals as potential solutions of a problem, and we build the algorithm so that the most capable individuals are the nearest to the solution, then we can hope after a certain number of generations, the solution to our problem.

Visioli presents in [13] a methodology for tuning PID controllers for integral and unstable processes. The method consists on a set of rules for integral and unstable processes, which are similar to the Ziegler and Nichols rules. These rules were obtained using a Genetic Algorithm optimization approach. The method contemplates the structure described in equation 3 for integrative processes (K is the transfer function gain, and L is the system delay), and the structure described in equation 4 for unstable processes (T is a time response rate).

$$G(s) = \frac{K}{s} e^{-Ls} \tag{3}$$

$$G(s) = \frac{K}{Ts - 1} e^{-Ls} \tag{4}$$

In the table 2, the tuning rules for optimal set-point response for unstable processes are shown. The rules presentede in table 2 are optimized are optimized to minimize the integral of the quadratic error (ISE). The complete rule set can be found in [13].

Table 2. Tuning rules for optimal optimal set-point response for unstable processes

PID parameter	ISE
K_p	$1.32/K(L/T)^{-0.92}$
T_i	$4.00(L/T)^{0.47}T$
T_d	$3.78T(1 - 0.84(L/T)^{-0.02})/(L/T)^{-0.95}$

Suppose an unstable plant whose transfer function is given by the equation 5. In accordance with the table 2, the parameter values will be: $K_p = 5.8$, $T_i = 1.88$, and $T_d = 0.11$. In the figure 1 the response of the process described in equation 5 when is controlled by the equation 1 is shown.

$$G(s)_5 = \frac{1}{s - 1} e^{-0.2s} \tag{5}$$

The main virtue of the Visioli method is its simplicity of use; however it has three important disadvantages: a) an identified model of the process is needed;

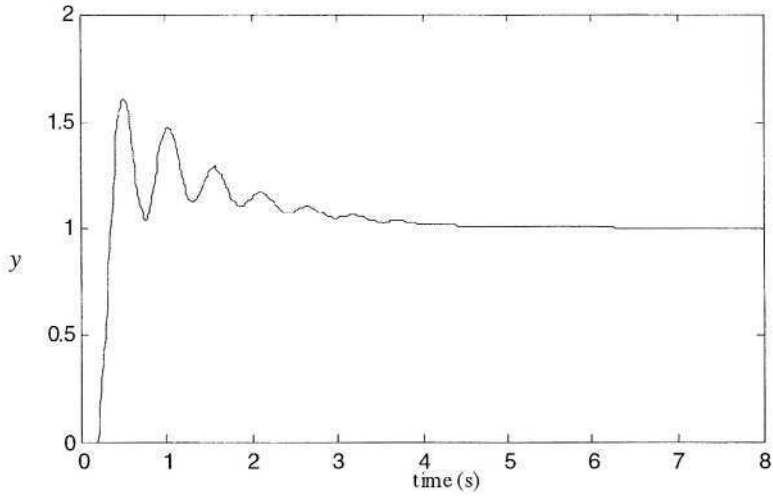


Fig. 1. PID Control of the process described in equation (5) using the Visioli method.

b) the identified model has to be of first order approach; and c) If $L = 0$ (L is the system delay), the method can not be used. In the next section we propose a methodology for tuning a PID control for integral and unstable process which is independent to the transfer function order. The method is a direct optimization of the controller parameters across a genetic algorithm, as it will be shown this method is superior to the Visioli approach.

4 Direct Genetic PID

The use of genetic algorithms for tuning PID controllers has been growing, because is very easy to connect the structure of the controller with the optimization strategy. The genetic algorithm task is to optimize the controller parameters, by means of an evolutionary process. Suppose a chromosome coded as is indicated in equation 6, where C_k is the gain chromosome. C_k represents a simple individual whose genes are the PID controller parameters ($C_k = K_p T_i T_d$, $K_p = p_1, p_2, ..., p_n$, $T_i = i_1, i_2, ..., i_n$, and $T_d = d_1, d_2, ..., d_n$) . Every chromosome is tested and evaluated. The individuals with fitness higher than the average shall have more reproduction opportunities. The gains that allow a better control performance give more chance to inherit the chromosome profile. To optimize the search, the decoding indicated in equation 7 is carried out. In equation 7, A_i is a gene of the chromosome C_k, which is weighted by a decoder weight i, fixed in a heuristic way, and K_i is a gain used by the equation 1. In the figure 2 is shown the optimization process, and it can be observed an on line identification process which use recursive least squares (RLS) [14]. The intern model is used in the

fitness function described in equation 8, where r_i is the reference point, $\widehat{y}$ is the estimated process and n is a value that allows the output process establishment, when it is stable otherwise n will have a default value. In the equation 8, the minimization criterion is the integral of absolute error (IAE).

$$C_k = p_1, p_2, ..., p_n, i_1, i_2, ..., i_n, d_1, d_2, ..., d_n \tag{6}$$

$$K_i = \Delta_i A_i \tag{7}$$

$$FF = \sum_{i=0}^{n} |r_i - \widehat{y}| \tag{8}$$

The higher qualified individuals are those that minimize the equation 8. The controller parameters are modified using a commutation only when an important parameter variation happens.

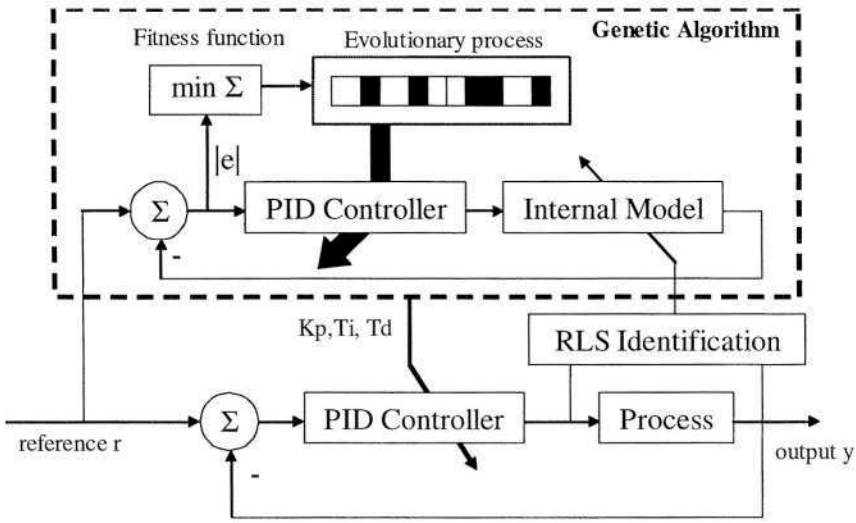


Fig. 2. Optimization structure

The genetic algorithm for the optimization has a population of 100 members. In every reproduction process a tournament selection is performed involving 10% of population. The maximum allele value is $2^{53} - 1$. For maximum focus in the search, decoding considers the value dividing the allele between ten times its maximum value. A mutation process which involves 10% of the population is applied, and the crossover is applied with probability of 1.00.

The equation 9 does not have a stable limit cycle; therefore it is not possible to use the Ziegler and Nichols methods. Applying the Direct Genetic PID, it is

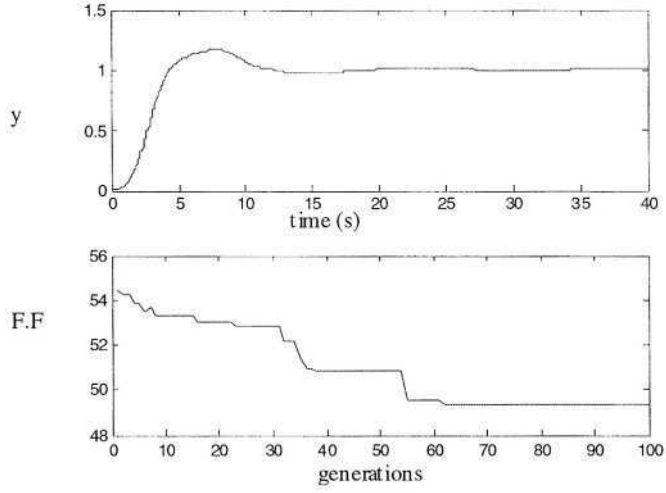


Fig. 3. Genetic PID controller of the process described in equation 9.

possible to control a marginally stable process. In the figure 3, it is shown the plant controlled output and the evolutionary process performance. The resultants gains for the process defined in equation 9 are $K_p = 0.156$, $T_i = 0.312$ and $T_d = 9.96$.

$$G(s)_2 = \frac{1}{s^3 + s^2 + s + 1} \tag{9}$$

The genetic optimization to control the double integrative process described in equation 10 is presented in figure 4. The resultants gains for the process described in equation 10 are $K_p = 4.251$, $T_i = 99.16$ and $T_d = 0.781$, for these processes is evident that the Visioli method can not be used because their structure are inadequate (see the equations 3 and 4). Also is shown the control for the unstable delayed process described by equation 5 in figure 5 (compare with the figure 1). The resultants gains for the process described in equation 5 are $K_p = 2.910$, $T_i = 1.367$ and $T_d = 0.135$.

$$G(s)_3 = \frac{1}{s^2} \tag{10}$$

The proposed method was applied to control a d.c. motor process, that is illustrated in figure 6, and the resultant output of the controlled process is illustrated in figure 7.

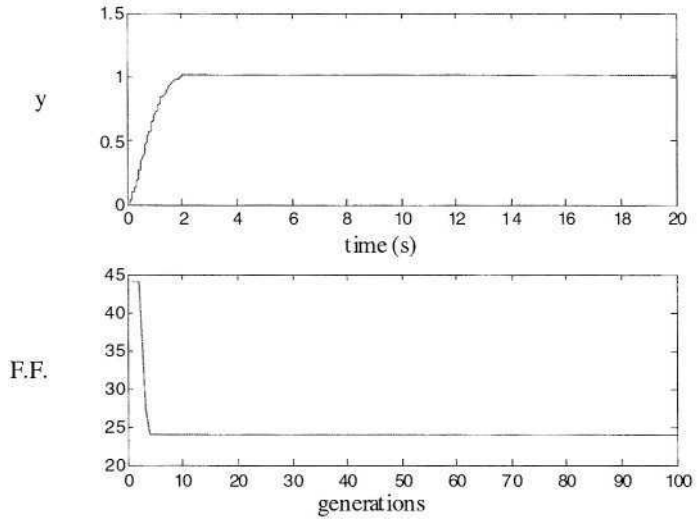


Fig. 4. Genetic PID controller of the process described in equation 10.

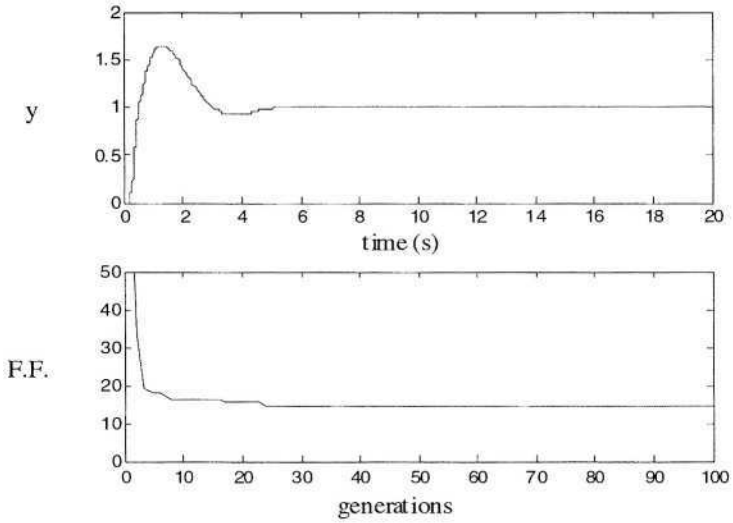


Fig. 5. Genetic PID controller of the process described in equation 6.

5 Conclusions

In this paper a new Genetic PID approach was presented to make control of systems with oscillatory or unstable modes. The main advantages of the pre-

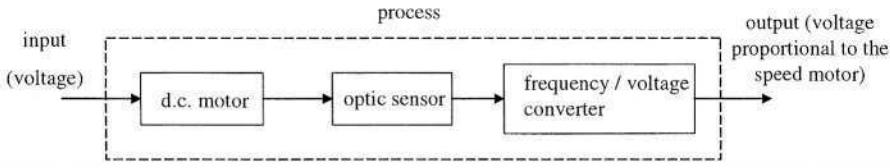


Fig. 6. Diagram of the controlled d.c. motor scheme (indicating the sensors)

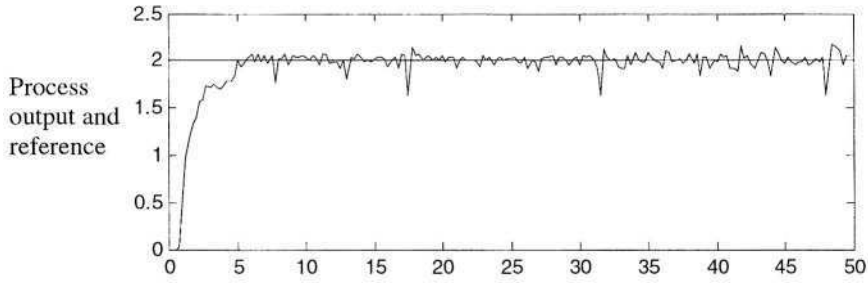


Fig. 7. The controlled speed in the d.c. motor

sented method relate to the possibility of make PID tuning when other methods have failed. Satisfactory results were observed through the exhibited tests in the paper. It is intented that the Genetic PID approach can be connected easily as a backup controller for other adaptive controllers. It is remarkable that the Genetic PID approach proposed does not require to know the process features, because an estimation stage (RLS) can be used.

References

1. Häglund T., and Åström K. J., *Industrial Adaptive Controllers Based on Frequency Response Techniques,* Automatica, Vol. 27, No.4, (1991) 594-609.
2. Hägglund T, and. Åström K.J., *Supervision of Adaptive Control Algorithms,* Automatica. Vol. 36, (2000) 1171-1180.
3. Isermann R., and Lachmann K.-H., *Parameter-adaptive Control with Configuration Aids and Supervision Functions,* Automatica, Vol. 21, No. 6, (1984) 625-638.
4. Vlachos C., Williams D., and Gomm J. B., *Genetic Approach to Decentralized PID Controller Tuning for Multivariable Processes,* IEE Proc. Control theory Appl. Vol 146, No. 1, (1999) 58-64.
5. Paz R. M.A., García B. C. D., and Torres J. J., *Fuzzy-Genetic Controller for a Coupled Drives System,* IEEE International Symposium on Industrial Electronics, Puebla, Mexico (2000) 741-746.
6. Krohling R. A., and Rey UJ. P., *Design of Optimal Disturbance Rejection PID Controllers Using Genetic Algorithms,* IEEE Transactions on evolutionary computation. Vol 5, No. 1, (2001) 78-82.

7. Åström K. J., and Hägglund T., *PID Controllers: Theory, Design, and Tuning* ISA organization 1995, ISBN: 1556175167
8. Åström K. J. et al (ed.), *Adaptive Control, Filtering, and Signal Processing,* Springer-Verlag 1995,ISBN: 0387979883
9. Hägglund T., and Åström K. J., *Method and Apparatus in Tuning a PID-Regulator,* U.S. Patent. Number 4549123, 1995
10. Holmberg V. Relay, *Feedback of Simple Systems,* Ph.D. Thesis, August 1991, Department of Automatic Control Lund Intitute of Technology
11. Holland J.H., *Adaptation in Natural and Artificial Systems,* MIT Press edition, 1992, ISBN: 0262581116.
12. Goldberg D., *Genetic Algorithms in Search, Optimization and Machine Learning,* Addison Wesley 1989,ISBN: 0201157675
13. Visioli A., *Optimal Tuning of PID Controllers for Integral and Unstable Processes,* IEEE Proc. –Control Theory Appl, Vol. 148, No. 2. (2001) 180-184.
14. Åström K. J., and Wittermark B., *Adaptive Control,* Second edition, Addison Wesley 1994, ISBN: 0201097206

Dynamic Uniform Scaling for Multiobjective Genetic Algorithms

Gerulf K.M. Pedersen[1] and David E. Goldberg[2]

[1] Aalborg University, Department of Control Engineering, Fredrik Bajers Vej 7, DK-9220 Aalborg Ø, Denmark
[2] University of Illinois at Urbana-Champaign, Department of General Engineering, 104 S. Mathews Ave., Urbana IL 61801, USA

Abstract. Before Multiobjective Evolutionary Algorithms (MOEAs) can be used as a widespread tool for solving arbitrary real world problems there are some salient issues which require further investigation. One of these issues is how a uniform distribution of solutions along the Pareto non-dominated front can be obtained for badly scaled objective functions. This is especially a problem if the bounds for the objective functions are unknown, which may result in the non-dominated solutions found by the MOEA to be biased towards one objective, thus resulting in a less diverse set of tradeoffs. In this paper, the issue of obtaining a diverse set of solutions for badly scaled objective functions will be investigated and the proposed solutions will be implemented using the NSGA-II algorithm.

1 Introduction

Multiobjective EAs (MOEAs) have been applied in a variety of areas. From design of airframes [1] to economic load dispatch problems in power systems [2] and over evolutionary path planners [3], MOEAs are becoming an important tool in practical optimization and decision making. Moreover, MOEAs themselves have received empirical and theoretical study which is well summarized in two monographs devoted to MOEAs [4,5].

Despite the rise in application, implementation, and theoretical interest, an important consideration in developing broadly capable MOEAs appear to have received scant attention. In particular, many MOEAs use a distance metric in an attempt to ensure a uniform distribution of individuals along the Pareto optimal front, but the individual objective functions may or may not operate over a comparable scale. As a result, it is important to explicitly consider and adapt to widely disparate scalings among different objectives. Here, we will examine the performance of one specific MOEA, NSGA-II, when the objective functions are badly scaled and consider dynamic uniform scaling procedures to solve such difficulty.

First we consider how badly scaled problems might pose a problem for NSGA-II and other MOEAs. Then the crowding mechanism of NSGA-II [6] will be investigated with regard to badly scaled problems, and some alterations for calculating the crowding distance will be suggested. The proposed changes will then be tested and the result will be presented, followed by some concluding remarks.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 11–23, 2004.

2 Background

Most MOEAs use a distance metric or other crowding method in objective space in order to maintain diversity for the non-dominated solutions on the Pareto optimal front. By ensuring diversity among the non-dominated solutions, it is possible to choose from a variety of solutions when attempting to solve a specific problem at hand.

Suppose we have two objective functions $f_1(\boldsymbol{x})$ and $f_2(\boldsymbol{x})$. In this case we can define the distance metric as the Euclidean distance in objective space between two neighboring individuals and we thus obtain a distance given by

$$d_f^2(\boldsymbol{x_1}, \boldsymbol{x_2}) = [f_1(\boldsymbol{x_1}) - f_1(\boldsymbol{x_2})]^2 + [f_2(\boldsymbol{x_1}) - f_2(\boldsymbol{x_2})]^2 \quad . \tag{1}$$

where $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$ are two distinct individuals that are neighboring in objective space. If the functions are badly scaled, e.g. $[\Delta f_1(\boldsymbol{x})]^2 \gg [\Delta f_2(\boldsymbol{x})]^2$, the distance metric can be approximated to

$$d_f^2(\boldsymbol{x_1}, \boldsymbol{x_2}) \approx [f_1(\boldsymbol{x_1}) - f_1(\boldsymbol{x_2})]^2 \quad . \tag{2}$$

In some cases this approximation will result in an acceptable spread of solutions along the Pareto front, especially for small gradual slope changes as shown in the illustrated example in Fig. 1.

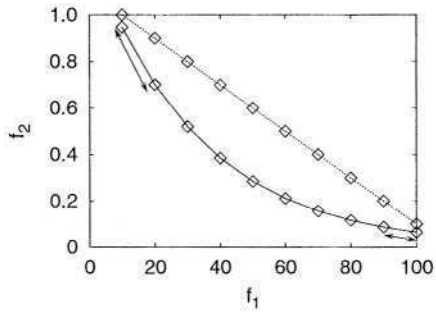
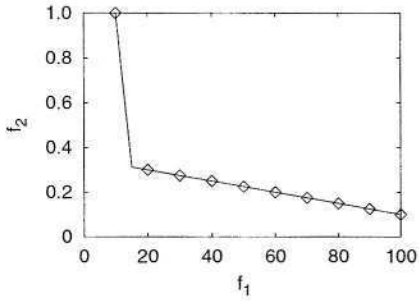


Fig. 1. For fronts with small gradual slope changes an acceptable distribution can be obtained even if one of the objectives (in this case f_2) is neglected from the distance calculations.

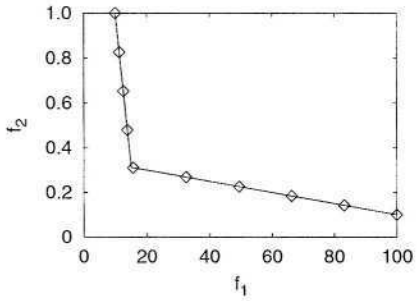
As can be seen in the figure, the distances marked by the arrows are not equal, but the solutions can still be seen to cover the front relatively well.

In other cases, however, the result is that a significant portion of the Pareto front is ignored, especially for large sudden changes of slope. This can be seen clearly in the illustration given in Fig. 2a, where only one objective function is used for calculation of the crowding distance thereby ignoring a large portion of the Pareto front along $f_2(\boldsymbol{x})$.

The method proposed in this paper will try to deal with this issue such that a bad distribution like the one illustrated in Fig. 2a can be avoided when using NSGA-II and instead be replaced by one similar to the one shown in Fig. 2b. First, however, it is necessary to get a better understanding of the distributions themselves.



(a) Neglecting one of the objectives for distance calculation results in a bad distribution.



(b) An ideal uniform distribution.

Fig. 2. Illustration of distributions on a piecewise linear front with bad scaling.

3 Distributions

When finding the Pareto non-dominated front for a multiobjective problem, the desired distribution of solution points on the Pareto front might differ depending on the problem being solved. In this paper it is desired for the solutions on the Pareto front to have a uniform distribution along the entire front. Such a uniform distribution is desirable, because it would provide the user with a good estimate of the tradeoffs between the different objectives. In case a non-uniform distribution is desired, it can be obtained by applying a transformation to the uniformly distributed points, but that will not be discussed further since it is out of the scope of this paper.

3.1 Uniform Distribution

The uniform distribution aims at generating a uniform spacing between each solution point on the Pareto non-dominated front. However, there are two different aspects of uniform distributions which must be considered separately, those with known bounds, which includes those problems with known scaling factors, and those with unknown bounds.

Known Bounds. Throughout the last decade, a lot of research have gone into finding crowding mechanisms, or other methods, that would result in a uniform distribution of non-dominated solutions along the Pareto front when the bounds of the individual objective functions are known or can be approximated [7,8]. For the case of a problem with known bounds, the bounds can be used a priori to normalize the objective functions, resulting in some very efficient crowding methods that ensure a good uniform distribution of points.

These crowding methods with known scaling factors do have some limitations becoming apparent later in this paper. They will give unsatisfactory result for badly scaled problems with unknown bounds. If a MOEA, that only uses fixed scaling, is used to find tradeoffs for a real-world design problem, the probability of obtaining this unsatisfactory result is quite high, since that is the situation in which the unknown bounds are most likely to be encountered.

In the public available version of NSGA-II [6], the crowding distance, d_j, for an individual, j, is calculated using hypercubes. The calculation of the hypercubes is given by sorting the individuals in ascending order, according to the objective value of the ith objective, f_i. The crowding distance for individual j is then calculated using

$$d_j = d_j + \frac{f_{i,j+1} - f_{i,j-1}}{f_{i,max} - f_{i,min}} \ . \tag{3}$$

For the case where the objective value for an individual is either the highest or the lowest amongst the population, the crowding distance is set to ∞. Such a policy ensures that an individual at the edge of the Pareto front will be preferred to an individual fully contained within the edges of the Pareto front. The values for $f_{i,min}$ and $f_{i,max}$ are in some of the literature recommended to be set equal to the maximum obtainable values for the objective function f_i [5], whereas others have tried using a dynamic approach for setting $f_{i,min}$ and $f_{i,max}$ [9].

If the normalization by $f_{i,min}$ and $f_{i,max}$ was not done, the crowding distance would be dominated by the objective functions with the largest values. Hence the distribution of the solutions on the Pareto front would be biased towards those objectives. As long as the bounds of the objective functions are known, then, as earlier mentioned, this normalization can be used to ensure a good uniform distribution of solution points. However, when the bounds are unknown it is necessary to find another way to normalize the crowding distance calculations.

Unknown Bounds. In order to obtain a good distribution we desire to avoid the problem of bad scaling when encountering unknown bounds. By ensuring that each objective can be normalized, even when the bounds are unknown, an equal number of significant tradeoffs can be obtained such that no objectives will be able to dominate the others. The crowding distance for unknown bounded scaled hypercubes can still be calculated using (3). However, the values for $f_{i,min}$ and $f_{i,max}$, since the bounds are unknown, need to be either fixed at a certain value or changed dynamically.

It would be unwise to use a fixed value for $f_{i,min}$ and $f_{i,max}$ when the bounds are unknown, since the lack of information about the objective functions would impair the effort to choose good values. This is what is currently done in NSGA-II [5]. Changing the values dynamically, as suggested by Bentley [9], or using other crowding techniques with normalization, as those seen in PAES [10] and PESA [11] will allow for good distributions to be obtained. Two methods for updating the values will be presented here, where one is similar to the method proposed by Bentley.

The normalization can be done either locally for each front or globally for all fronts[1]. When done locally the maximum and minimum values for each objective in each front are

[1] The use of fronts in NSGA-II corresponds to the use of equivalence classes.

used to normalize the crowding distances for that front. When done globally for all fronts, which is similar to Bentleys approach [9], the maximum and minimum values for each objective in the current population are used for normalization. By using one of those normalization techniques, it should be possible to obtain a good uniform distribution for the solutions along the different fronts when the bounds are unknown. With the normalization schemes in place it is now time to take a closer look at the setup of the experiments.

4 Experimental Setup

The emphasis in this paper is focused on investigating how to successfully normalize the objective functions for the problems with unknown bounds when used in the crowding distance calculations in NSGA-II. The goal is to obtain a uniform distribution along the Pareto non-dominated front for these problems.

Objective Functions. For simplicity this paper will only consider cases consisting of two objectives. The results are also valid for higher dimension cases, but the use of only two objective functions is chosen to ensure a straightforward visualization of the results obtained.

The objective functions are deliberately chosen such that the optimal Pareto-front is known, and such that bad scaling can be easily obtained. This allows for a direct comparison of the optimal solutions with those found by the algorithm, which will provide a help in fully understanding how the points are distributed along the Pareto-front and may also shed light on possible issues that could be further improved.

To illustrate the problems with badly scaled objective functions, the objective functions for this paper are chosen such that the relative scaling of the objective functions can be easily varied. The objective functions presented here does not correspond to any of the benchmark problems usually encountered when testing the performance of MOEAs [10, 12,13], since those problems do not have scaling issues or have already been normalized before they are used in the MOEAs. The objective functions are given by

$$f_1 = x_1^k + |x_2|, \quad k > 0 \qquad \text{and} \qquad f_2 = x_1^{-l} + |x_2|, \qquad x_1 > 0, \; l > 0 \;, \tag{4}$$

where $x_1, x_2 \in \mathbb{R}$ are the optimization variables, and k and l determine the extent of scaling between the functions. The optimal Pareto-front is obtained for

$$f_2 = f_1^{-\gamma} \qquad x_2 = 0 \;, \tag{5}$$

where $\gamma = \frac{l}{k}$. Thus, for small values of γ ($\gamma < 1$), f_1 will have bigger scaling than f_2 and vice versa. Optimal scaling between objective functions will be obtained for $\gamma = 1$.

Variables. The number of variables is two, as shown in (4), and the encoding of both variables is binary. A real-valued encoding should be able to produce similar results for this simple problem, but that is out of the scope of this paper, since it depends on the operators and their implementation.

The representation for variable x_1 is chosen to be 16 bit long belonging to [0.1,10], so x_1 complies with the constraints given in (4) and assures that the resulting non-dominated fronts found using NSGA-II will not have excessive extreme values. The variable x_2 is represented using 16 bits and, since the range can be chosen arbitrarily, we set it to [–100,100]. The specified ranges allows for the calculation of the maximum and minimum values of both objective functions. It is necessary to emphasize that this assumption of known bounds does not in any way affect the conclusion obtained in this paper, since the paper is meant to illuminate potential problems, showing measures of how they might be avoided for real world problems.

Algorithmic Setup. The algorithm chosen for the investigations in this paper is the NSGA-II developed by Kalyanmoy Deb. NSGA-II was chosen since it is an overall good MOEA with respect to several different classes of problems [14,15] but have yet to be tested on badly scaled problems without normalized objective functions.

The selection operator is not expected to have a major influence on the results derived in this paper. The only requirements to the selection operator are that solutions belonging to a lower ranked front should be preferred over solutions belonging to higher ranked fronts, and in case the solutions belong to the same front the one with the largest crowding distance should be preferred. Since the selection method used in NSGA-II is tournament selection with size 2, which meets the specified requirements, that setting is used without modification.

The problem defined in (4) is very simple and it is thus not expected that using different types of crossover will influence the performance of the algorithm too much. Thus, the crossover operator is chosen to be uniform. The crossover probability p_c is chosen to be 0.9 which should ensure good convergence for this simple problem [16].

With the binary representation, the use of bitwise mutation is straightforward to implement and is expected to produce results that will be similar to those obtained if other and more complex mutation methods is used. The mutation probability p_m is set to 0.01.

Using the previously defined parameters it can be determined that the search space is comprised of 2^{32} possible solutions, whereas, when the known optimal Pareto front is taken into account, the number of Pareto non-dominated solutions consists of 2^{16} points. With a population size of 200 individuals it will be possible to cover 0.3% of the non-dominated front if the entire population belongs to that front. A coverage that loose will give ample possibility to fully investigate how the crowding mechanisms will perform while still providing a good estimate of the Pareto non-dominated front, which in this case is known to be continuous and smooth.

The algorithm is run for 200 generations, which should be sufficient for finding solutions on the true Pareto non-dominated front, and also to apply the crowding measures which should spread the solutions uniformly along the front.

30 independent trials, with differing random seeds, will be conducted for each experiment. This is done to ensure that no individual run will be able to overly influence the results of an experiment and the conclusions drawn from those results.

The parameters used to produce the results of this paper are summarized in table 1, and with the experimental setup now in place we can move forward to the experiments.

Table 1. Parameters used for NSGA-II.

Parameter description	Value	Designation
Number of objectives	2	f_1, f_2
Format of variables	Binary	-
Number of variables	2	x_1, x_2
No. of bits for x_1 variable	16	l_{x_1}
No. of bits for x_2 variable	16	l_{x_2}
Range for x_1 variable	[0.1,10]	$[x_{1_{min}}, x_{1_{max}}]$
Range for x_2 variable	[-100,100]	$[x_{2_{min}}, x_{2_{max}}]$
Selection operator	Tournament	-
Tournament size	2	s
Crossover type	Uniform	-
Crossover probability	0.9	p_c
Mutation type	Bitwise	-
Mutation probability	0.01	p_m
Population size	200	n_{pop}
Maximum generations	200	n_{max}

5 Results

In this section the results of the discussed methods are compared to the original NSGA-II algorithm. For all of the tests performed, the algorithm succeeded in finding the actual Pareto fronts with the entire population, all 200 solution points, located on said front. Also, the algorithm always succeeded in finding the outermost solution points for the Pareto front, and for this reason the calculation of the spread on the Pareto front will not include any terms penalizing the lack in finding the outermost solutions.

5.1 Original NSGA-II

The algorithm was first run using 3 distinct γ-values (0.2, 1, and 5) with the original crowding distance calculations. It was then possible to illustrate the effects of a bad scaling for each of the cases where f_1 and f_2 were overly emphasized. It was also possible to show what resulted for the optimal situation, with equal scaling of the objectives. The results for the 3 different γ-values can be seen in Fig. 3,4a and 4b, where the true fronts are also shown[2].

From figure 3 it can be seen that when the objective functions have equal scaling ($\gamma = 1$), then NSGA-II is able to find the true Pareto optimal front with a uniform distribution of points which preserves the tradeoffs for both objective functions. The spread of the distribution for a single run can be calculated using

$$spr = \sum_{j=1}^{n_{pop}} \left(d_j - \bar{d}\right)^2 \quad , \tag{6}$$

[2] All of the result graphs presented in this paper are based on a run with a random seed of 0.1234.

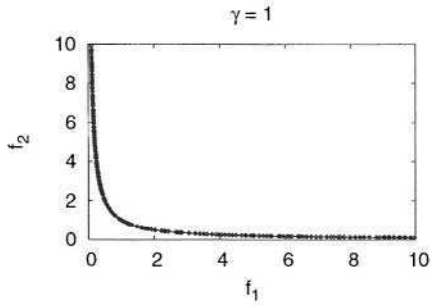
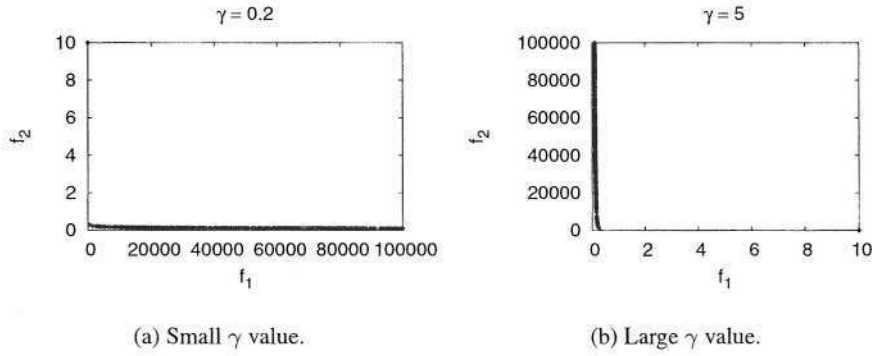


Fig. 3. Pareto non-dominated front for unity γ value plotted on the known optimal front.

where d_j is the crowding distance as calculated in (3) and $\bar{d}$ is the mean of the normalized Euclidean distances between the solution points on the graph. This spread metric is a modified version of the one used in [5], the main difference being the normalization of d_j and $\bar{d}$, which is not used in [5]. Due to the simplicity of the problem it has been deemed unnecessary to include the distance between the outermost solutions and the known extremes of the Pareto front, since those in all experiments coincided, thus giving an added value of zero. The normalization in (6) is done according to $f_{i,max} - f_{i,min}$ for both objectives, thus the distance between the minimum and maximum values for an objective after normalization is equal to 1. The spread is then averaged over 30 runs and the resulting mean spread for running NSGA-II with $\gamma = 1$ is $6.373 \cdot 10^{-3}$.



(a) Small γ value.

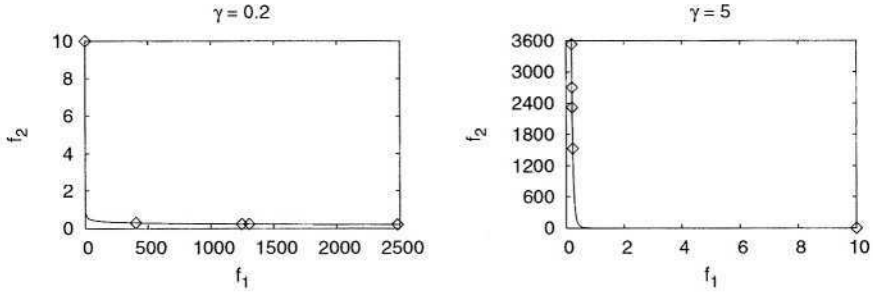
(b) Large γ value.

Fig. 4. Pareto non-dominated fronts for large and small γ values using original NSGA-II crowding and plotted on the known optimal front.

As seen in Fig. 4 it is clear that even though the Pareto optimal front is found, the distribution of points is not uniform with regard to both objectives. In other words, in Fig. 4a the tradeoffs for objective f_2 is almost non-existent, since all of the solution

points are located in the direction of objective function f_1. This can also be seen from the fact that the spread for this case is $924.8 \cdot 10^{-3}$, which is much higher than that for $\gamma = 1$. Similarly the opposite can be seen in Fig. 4b, where the solution points are distributed towards objective function f_2 and the spread is found to be $942.7 \cdot 10^{-3}$.

The problem for both of the cases ($\gamma = 0.2, \gamma = 5$) lies in the fact that the bounds are unknown which causes the crowding distance, calculated using (3), to overly emphasize the objective function with the highest values. Since the bounds are unknown a default value of 1 is used for $f_{i,max} - f_{i,min}$, which corresponds to the implementation used in the public version of NSGA-II [6]. The details of the emphasized objective functions can be seen more clearly in Fig. 5a and 5b, which gives a detailed view of the lower portion of the objective space for γ-values of 0.2 and 5.



(a) Small γ value with f_1 values less than 2500.

(b) Large γ value with f_2 values less than 3600.

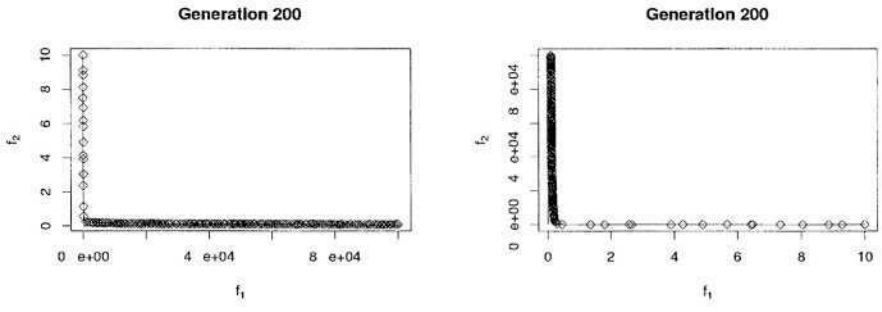
Fig. 5. Lower region of the Pareto non-dominated front for small and large γ values using original NSGA-II crowding.

In these figures it is clear that the crowding distance is dominated by the objective with the largest scale, which is equivalent to the situation described in section 2.

It is now clear that, when the objective functions with unknown bounds are badly scaled, the resulting non-dominated Pareto front does not have a distribution of points which allow for determining proper tradeoffs between the objective functions, since the tradeoffs for one objective totally overshadows the other. This can be concluded since no actual tradeoffs are shown on the Pareto front. As such, it is now time to take a look at the proposed approaches.

5.2 Global Scaling

The modified crowding measures, where the normalization values of $f_{i,min}$ and $f_{i,max}$ were set according to either global minimum and maximum values, as used by Bentley [9], or local minimum and maximum values, were then implemented and tested on the same problems.



(a) Small γ value. (b) Large γ value.

Fig. 6. Pareto non-dominated fronts for small and large γ values when using globally based normalization.

The results obtained using the global normalization scheme can be seen in Fig.6a and 6b. As seen on the figures, the global normalization is capable of maintaining a small number of Pareto non-dominated solutions for the objectives with the smaller objective values. This means that the global normalization did help in producing a better distributed set of solutions, nevertheless it is also evident that the problem was only partly solved, since the majority of solutions are still concentrated on the objective with the highest fitness values. This is further emphasized by the calculated spreads which are found to be $71.00 \cdot 10^{-3}$ and $70.29 \cdot 10^{-3}$ for γ-values 0.2 and 5 respectively.

Results for $\gamma = 1$ were also obtained but they did not differ significantly from the results obtained using the original crowding calculations of NSGA-II, with a spread of $6.833 \cdot 10^{-3}$.

5.3 Local Scaling

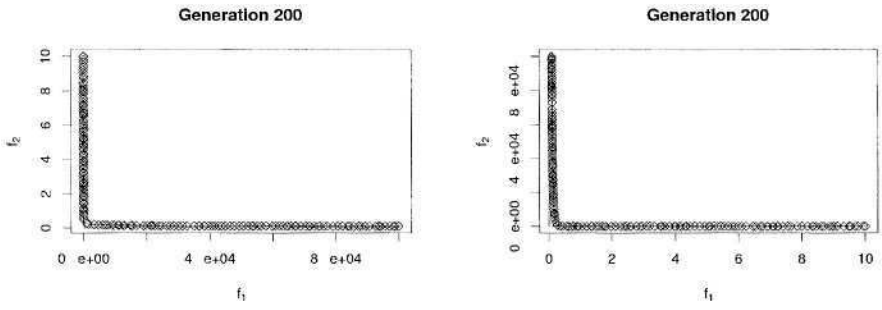
The algorithm was then run using the proposed method with normalization based locally on values for each front, which resulted in Fig. 7a and 7b.

It is very clear from the figures that the locally based normalization resulted in a distribution of solution points which is almost uniformly distributed for both objective functions. This is further emphasized by the fact that the spreads are found to be $8.116 \cdot 10^{-3}$ and $7.912 \cdot 10^{-3}$ for γ-values of 0.2 and 5 respectively.

Thus, by using a locally based normalization, when calculating the crowding distances, it was possible to ensure a nearly uniform distribution for a badly scaled problem with unknown bounds. Results were also obtained for $\gamma = 1$ but once again they did not display results that differed significantly from those obtained when using the original crowding calculations and the spread for this case was $6.439 \cdot 10^{-3}$.

5.4 Discussion

The results obtained are summarized in table 2. A salient issue remaining to be explained is how the globally based normalization of the crowding distances were able to only



(a) Small γ value. (b) Large γ value.

Fig. 7. Pareto non-dominated front for small and large γ values when using locally based normalization.

Table 2. Spreads obtained using different γ-values for original NSGA-II crowding distance calculations and for globally and locally based normalization of crowding distances.

Crowding method	$\gamma = 0.2$	$\gamma = 1$	$\gamma = 5$
Original	$924.8 \cdot 10^{-3}$	$6.373 \cdot 10^{-3}$	$942.7 \cdot 10^{-3}$
Global	$71.00 \cdot 10^{-3}$	$6.833 \cdot 10^{-3}$	$70.29 \cdot 10^{-3}$
Local	$8.116 \cdot 10^{-3}$	$6.439 \cdot 10^{-3}$	$7.912 \cdot 10^{-3}$

partially solve the problem of the non-uniform distribution of solutions on the Pareto front. In order to get a full understanding of that issue it is necessary take a look at how the MOEA itself is implemented.

The vital part of the explanation lies in the fact that the selection in NSGA-II is based on Pareto ranking. In many cases the solution space which can contain the Pareto non-dominated solutions might only be a subset of the entire objective space. Due to the ranking method, the subsequent fronts will always have higher values for at least one objective when compared to the solutions on the fronts that dominate the one currently considered. This means that the subsequent fronts might span a set that is bigger than that spanned by the fronts dominating the current ones. As a result, when the normalization parameter was calculated globally, it was found to be the maximum value of the set given by the union of all fronts. Thus, if the application of crossover and mutation had resulted in solutions that belonged to subsequent dominated fronts, which is highly likely, especially for the Pareto non-dominated front, then the normalization values was calculated based on values that most likely would be too large. The normalization parameter would thus also be too large and the distribution would be biased away from that objective accordingly.

For the locally based dynamic scaling, the scaling was calculated using only values already present in the front under consideration. Thus, higher values in subsequent fronts would have no effect on the current front and no biased scaling would occur. The crowding distances calculated for those fronts would then be independent of the relative

scaling of the objective functions and a uniform distribution can then be obtained for the badly scaled problems.

6 Conclusion

In this paper we investigated the effect of badly scaled objective functions on the distribution of solution points along the Pareto front with unknown objective function bounds. To correct the problem, two different dynamic scaling methods were implemented in NSGA-II and the results showed that one of the methods was able to successfully achieve a near uniform distribution of solution points along the Pareto front.

It is clear from the obtained results that dealing with a badly scaled problem with unknown bounds can result in problems. The results also show that, in order to obtain a uniform distribution of solution points for both objectives, dynamic scaling using a locally based normalization scheme will give the desired result. As a result, the proposed dynamic scaling proposed in this paper should be remembered when designing new MOEAs or if an existing MOEA, which do not use a dynamic normalization scheme, is used for a badly scaled optimization problem.

Acknowledgments. The authors would like to thank Kumara Sastry, Dr. Xavier Llorà and the reviewers for many helpful comments and suggestions.

References

1. Parmee, I.C., Watson, A.H.: Preliminary airframe design using co-evolutionary multiobjective genetic algorithms. Proceedings of the Genetic and Evolutionary Computation Conference 1999 **2**(1999)1657–1665
2. Das, D.B., Patvardhan, C.: New multi-objective stochastic search technique for economic load dispatch. IEEE Proceedings of Generation, Transmission and Distribution **145** (1998) 747–752
3. Dozier, G., McCullough, S., Homaifar, A., Tunstel, E., Moore, L.: Multiobjective evolutionary path planning via fuzzy tournament selection. Proceedings of 1998 IEEE International Conference on Evolutionary Computation (1998) 684–689
4. Coello Coello, C.A., Van Veldhuizen, D.A., Lamont, G.B.: Evolutionary Algorithms for Solving Multi-Objective Problems. Genetic Algorithms and Evolutionary Computation. Kluwer Academic Publishers, New York, NY (2002)
5. Deb, K.: Multi-Objective Optimization using Evolutionary Algorithms. 1st edn. John Wiley & Sons, Ltd., West Sussex, England (2001)
6. Deb, K., Pratap, A., Moitra, S.: A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. Parallel Problem Solving from Nature - PPSN VI (2000) 849–858 NSGA-II code available at KanGAL website: 'http://www.iitk.ac.in/kangal/'.
7. Obayashi, S.: Pareto genetic algorithm for aerodynamic design using the Navier-Stokes equations. In Quagliarella, D., Périaux, J., Poloni, C., Winter, G., eds.: Genetic Algorithms and Evolution Strategies in Engineering and Computer Science. John Wiley & Sons, Ltd., Trieste, Italy (1997) 245–266

8. Deb, K., Mohan, M., Mishra, S.: A fast multi-objective evolutionary algorithm for finding well-spread pareto-optimal solutions. Technical Report 2003002, Kanpur Genetic Algorithms Laboratory (KanGAL), Indian Institute of Technology Kanpur, Kanpur, PIN 208016, India (2003)
9. Bentley, P.J., Wakefield, J.P.: Finding acceptable solutions in the pareto-optimal range using multiobjective genetic algorithms. Proceedings of the Second Online World Conference on Soft Computing in Engineering Design and Manufacturing (WSC2) **5** (1998) 231–240
10. Knowles, J.D., Corne, D.W.: The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation. Proceedings of 1999 IEEE International Conference on Evolutionary Computation (1999) 98–105
11. Corne, D.W., Knowles, J.D., Oates, M.J.: The pareto envelope-based selection algorithm for multiobjective optimization. Parallel Problem Solving from Nature - PPSN VI (2000) 839–848
12. Deb, K.: Multi-objective genetic algorithms: Problem difficulties and construction of test problems. Evolutionary Computation **7** (1999) 205–230
13. Zitzler, E., Thiele, L.: Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. IEEE Transactions on Evolutionary Computation **3** (1999) 257–271
14. Deb, K., Jain, S.: Multi-speed gearbox design using multi-objective evolutionary algorithms. Technical Report 2002001, Kanpur Genetic Algorithms Laboratory (KanGAL), Indian Institute of Technology Kanpur, Kanpur, PIN 208016, India (2002)
15. Deb, K.: Unveiling innovative design principles by means of multiple conflicting objectives. Technical Report 2002007, Kanpur Genetic Algorithms Laboratory (KanGAL), Indian Institute of Technology Kanpur, Kanpur, PIN 208016, India (2002)
16. Goldberg, D.E.: The Design of Innovation: Lessons from and for Competent Genetic Algorithms. Genetic Algorithms and Evolutionary Computation. Kluwer Academic Publishers, Norwell, MA (2002)

Parameter-Less Hierarchical BOA

Martin Pelikan and Tz-Kai Lin

Dept. of Math. and Computer Science, 320 CCB
University of Missouri at St. Louis
8001 Natural Bridge Rd., St. Louis, MO 63121
pelikan@cs.umsl.edu
tlkq4@studentmail.umsl.edu

Abstract. The parameter-less hierarchical Bayesian optimization algorithm (hBOA) enables the use of hBOA without the need for tuning parameters for solving each problem instance. There are three crucial parameters in hBOA: (1) the selection pressure, (2) the window size for restricted tournaments, and (3) the population size. Although both the selection pressure and the window size influence hBOA performance, performance should remain low-order polynomial with standard choices of these two parameters. However, there is no standard population size that would work for all problems of interest and the population size must thus be eliminated in a different way. To eliminate the population size, the parameter-less hBOA adopts the population-sizing technique of the parameter-less genetic algorithm. Based on the existing theory, the parameter-less hBOA should be able to solve nearly decomposable and hierarchical problems in quadratic or subquadratic number of function evaluations without the need for setting any parameters whatsoever. A number of experiments are presented to verify scalability of the parameter-less hBOA.

1 Introduction

When the hierarchical Bayesian optimization algorithm (hBOA) was designed [1, 2], it was argued that hBOA can solve difficult nearly decomposable and hierarchical problems without the need for setting any parameters. As a result, to solve a new black-box optimization problem, it should be sufficient to plug the new problem into hBOA, press the start button, and wait until hBOA figures out where the optimum is. It was argued that all hBOA parameters except for the population size can be set to their default values without affecting good scalability of hBOA [3]. However, choosing an adequate population size was argued to be crucial [3] and in all experiments the user was assumed to set the population size optimally to obtain best performance while retaining reliable convergence. That is why the dream of having a fully parameter-less optimizer for the entire class of nearly decomposable and hierarchical problems remained one parameter away from reality.

The purpose of this paper is to propose a fully parameter-less hBOA by implementing the parameter-less population-sizing scheme of the parameter-less

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 24–35, 2004.

genetic algorithm (GA) [4] in hBOA. The parameter-less hBOA simulates a collection of populations of different sizes, starting with a small base population size. Each next population is twice as large as the previous one. To enable parallel simulation of a number of populations when the number of populations that must be simulated is not known in advance, each population is forced to proceed with at most the same speed as any smaller population, where speed is considered with respect to the number of function evaluations. This ensures that although there is no upper bound on the number of populations simulated in parallel and on their size, the overall computational overhead is still reasonable compared to the case with an optimal population size. In fact, theory exists that shows that the parameter-less population sizing scheme in the parameter-less GA does not increase the number of function evaluations until convergence by more than a logarithmic factor [5]. As a result, hBOA can be expected to perform within a logarithmic factor from the case with the optimal population size. This is verified with a number of experiments on nearly decomposable and hierarchical problems.

The paper starts by discussing probabilistic model-building genetic algorithms (PMBGAs) and the hierarchical BOA (hBOA). Section 3 discusses the parameter-less genetic algorithm, which serves as the primary source of inspiration for designing the parameter-less hBOA. Section 4 describes the parameter-less hBOA. Section 5 describes experiments performed and discusses empirical results. Finally, Section 6 summarizes and concludes the paper.

2 Hierarchical Bayesian Optimization Algorithm (hBOA)

Probabilistic model-building genetic algorithms (PMBGAs) [6,7] replace traditional variation operators of genetic and evolutionary algorithms by a two-step procedure. In the first step, a probabilistic model is built for promising solutions after selection. Next, the probabilistic model is sampled to generate new solutions. By replacing variation operators inspired by genetics with machine learning techniques that allow automatic discovery of problem regularities from populations of promising solutions, PMBGAs provide quick, accurate, and reliable solution to broad classes of difficult problems, many of which are intractable using other optimizers [1,2].

For an overview of PMBGAs, please see references [6] and [7]. PMBGAs are also known as estimation of distribution algorithms (EDAs) [8] and iterated density-estimation algorithms (IDEAs) [9]. The remainder of this section describes the hierarchical Bayesian optimization algorithm, which is one of the most advanced and powerful PMBGAs.

2.1 Hierarchical BOA (hBOA)

The hierarchical Bayesian optimization algorithm (hBOA) [1,2] evolves a population of candidate solutions to the given problem. The first population of candidate solutions is usually generated at random. The population is updated

```
Hierarchical BOA (hBOA)
  t := 0;
  generate initial population P(0);
  while (not done) {
    select population of promising solutions S(t);
    build Bayesian network B(t) with local struct. for S(t);
    sample B(t) to generate offspring O(t);
    incorporate O(t) into P(t) using RTR yielding P(t+1);
    t := t+1;
  };
```

Fig. 1. The pseudocode of the hierarchical BOA (hBOA).

for a number of iterations using two basic operators: (1) selection, and (2) variation. The selection operator selects better solutions at the expense of the worse ones from the current population, yielding a population of promising candidates. The variation operator starts by learning a probabilistic model of the selected solutions. hBOA uses Bayesian networks with local structures [10] to model promising solutions. The variation operator then proceeds by sampling the probabilistic model to generate new solutions, which are incorporated into the original population using the restricted tournament replacement (RTR) [11]. RTR ensures that useful diversity in the population is maintained for long periods of time. The run is terminated when a good enough solution has been found, when the population has not improved for a long time, or when the number of generations has exceeded a given upper bound. Figure 1 shows the pseudocode of hBOA. For a more detailed description of hBOA, see [12].

3 Parameter-Less Genetic Algorithm

The parameter-less genetic algorithm (GA) [4] eliminates the need for setting parameters—such as the population size, selection pressure, crossover rate, and mutation rate—in genetic algorithms. The crossover rate and selection pressure are set to ensure consistent growth of building blocks based on the schema theorem. The mutation rate can be eliminated in a similar manner. The population size is eliminated by simulating a collection of populations of different sizes. In the context of hBOA, eliminating the population size is the most important part of the parameter-less GA, because the selection pressure influences performance of hBOA by only a constant factor and there are no crossover or mutation rates in hBOA.

3.1 Eliminating Selection Pressure and Crossover Rate

The parameter-less GA assumes that selection and recombination are primary search operators. The choice of selection pressure and crossover rate should consider the following facts. Selection must be strong enough to ensure consistent

growth of superior building blocks of the optimum, but it must not be too strong because otherwise diversity in the population might be lost prematurely. On the other hand, the crossover rate must be large enough to ensure sufficient exploration of the search space but if blind crossover operators (e.g., one-point and uniform crossover) are used, each application of crossover can break an important building block and thus crossover must not be applied too frequently.

Additionally, there is a tradeoff between selection and crossover. Greater selection pressures allow greater crossover rates. Analogically, smaller selection rates allow only smaller crossover rates. The tradeoff can be formalized using a simplification of the schema theorem, which claims that the expected number of copies of each partial solution or schema after selection and crossover is given by $s(1-\epsilon)$, where s characterizes the selection strength as a factor by which the number of best solutions will grow, and ϵ represents disruption of the schema by crossover. By ignoring mutation and making a conservative assumption that crossover always disrupts the schema, it is easy to show that $\epsilon = p_c$ and that setting $s = 4$ and $p_c = 0.5$ ensures net growth of 2 for the schemata contained in the best solution.

The original parameter-less GA did not consider mutation. To incorporate mutation, the probability of disrupting a schema due to mutation would have to be incorporated into ϵ. For a bit-flip mutation and binary strings, where each bit of a solution is flipped with a fixed probability p_m, a bounding case could assume that the schema under consideration spans across the entire solution. In that case, the probability of disrupting the schema due to mutation can be computed as $1-(1-p_m)^n$, where n is the total number of bits in the solution.

For interacting variables where traditional variation operators fail, even the parameter-less GA is going to suffer from excessive disruption and ineffective mixing of building blocks, as the negative effects of blind variation cannot be eliminated by tweaking GA parameters, but only by modifying the operators themselves.

3.2 Eliminating Population Size

To eliminate the population size, the parameter-less GA simulates a collection of populations of different size [4]. It is important that for any population size N, there exists a population of size greater or equal than N in the collection. Otherwise, problems that require population size greater or equal than N could not be solved. Consequently, the collection must contain infinitely many populations, the size of which cannot be upper bounded.

The parameter-less GA arranges the collection of populations as a *sequence*. The size of the first population in the sequence is set to a small constant called the base population size. The size of the second population is twice the size of the first population. In general, each next population is twice the size of the previous population and the population size thus grows exponentially starting with the base population size.

Each population in the collection is allowed to run one generation for each k generations of the population twice as large, where $k \geq 2$ is an integer con-

stant. The original parameter-less GA considered $k = 4$. That means, that the smaller population was allowed to proceed at twice the speed of the next larger population, where speed is again measured with respect to fitness evaluations. It can be shown that for any $k \geq 2$, the infinite collection of populations can be simulated tractably without increasing the number of function evaluations until convergence by more than a logarithmic factor with respect to the optimal population size [5].

3.3 How Many Generations for Each Population?

Based on convergence theory for both large and small populations [13,14,15], the number of generations can be assumed to be upper-bounded by a constant that does not depend on the population size. Since small populations process generations faster than larger populations, it seems reasonable to terminate simulation of each population at some point and use computational resources more efficiently.

The parameter-less GA terminates a population in the collection if either of the following criteria is satisfied:

- The population converges and it consists of many copies of a single solution. In this case, it can be expected that no more improvement will take place anymore or that the search will become very inefficient. Clearly, this criterion is not going to have much effect if niching is used.
- A larger population has a greater average fitness. Since larger populations converge generally at most as fast as smaller populations, this situation indicates that the smaller population got stuck in a local optimum and it can thus be terminated.

It is important to note that the logarithmic overhead computed in [5] does not consider the termination criterion and the parameter-less GA should thus perform well even without terminating any populations in the collection.

4 Parameter-Less hBOA

The parameter-less hBOA incorporates the population-sizing technique of the parameter-less GA into hBOA. Since hBOA ensures growth and mixing of important building blocks via learning and sampling a probabilistic model of promising solutions, there is no reason to restrict crossover rate and all offspring can be created by sampling the probabilistic model of promising solutions. Additionally, any selection pressure that favors best candidate solutions can be used without changing scalability of hBOA by more than a constant factor.

The parameter-less hBOA simulates a collection of populations

$$P = \{P_0, P_1, \ldots\}.$$

The size of the first population P_0 is denoted by N_0 and it is called the base population size. The size of the population P_i is denoted by N_i and it can be obtained by multiplying the base population size by a factor of 2^i:

$$N_i = N_0 2^i.$$

For all $i \in \{1, 2, \ldots\}$, one generation of P_i is executed after executing $k \geq 2$ generations of P_{i-1}. Here we use $k = 2$ so that all populations proceed at the same speed with respect to the number of evaluations. Each population is initialized just before its first iteration is executed. The pseudocode that can be used to simulate the collection of populations described here is shown in Figure 2. This implementation is slightly different from the one based on a k-ary counter described in the first parameter-less GA study [4].

```
Parameter-less hBOA
  initialize P[0];
  generation[0]=0;
  max_initialized=0;
  i=0;
  while (not done) {
    simulate one generation of P[i];
    generation[i] = generation[i]+1;
    if (generation[i] mod k = 0) {
      i = i + 1;
      if (i>max_initialized) {
        initialize P[i];
        max_initialized=i;
      }
    }
    else
      i = 0;
  };
```

Fig. 2. The pseudocode of the parameter-less hBOA.

4.1 When to Terminate a Population?

The same termination criteria as in the parameter-less GA can be used in the parameter-less hBOA. However, since hBOA uses niching, no population in the collection can be expected to converge for a long time. Additionally, we terminate each population after it executes for a number of generations equal to the number of bits in the input string. According to our experience, enabling each population to run for more generations does not improve performance further, whereas decreasing the limit on the number of generations might endanger convergence on exponentially scaled and hierarchical problems. Similarly as in the parameter-less GA, in the parameter-less hBOA populations can also be run indefinitely

without increasing the worst-case overhead compared to the case with an optimal population size as predicted by theory.

5 Experiments

This section describes experimental methodology, test problems, and experimental results.

5.1 Experimental Methodology

The parameterless hBOA was applied to artificial hierarchical and nearly decomposable problems and 2D $\pm J$ spin glasses with nearest neighbor interactions and periodic boundary conditions. For artificial problems, problem size was varied to examine scalability of the parameter-less hBOA. The performance of the parameter-less hBOA was compared to that of hBOA with optimal population size. For spin glasses, systems of different size were tested and 100 random instances were examined for each problem size to ensure that the results would provide insight into hBOA performance on a wide range of spin glass instances. To improve hBOA performance on spin glasses, a deterministic local search that flips each bit in a candidate solution until the solution cannot be improved anymore is used to improve each candidate solution before it is evaluated. The local searcher favors the best change at each iteration [12]. In all experiments, the base population size $N_0 = 10$ is used.

For each problem instance and each problem size, the parameter-less hBOA is first run to find the optimum in 100 independent runs and the total number of evaluations in every run is recorded. The average number of function evaluations is then displayed. These results are compared to the results for hBOA with the minimum population size that ensures that 30 independent runs converge to the optimum. The minimum population size was determined using bisection until the width of the resulting interval is at most 10% of the lower bound.

The remainder of this section discusses test problems and experimental results.

5.2 Deceptive Function of Order 3

In the deceptive function of order 3 [16,17], the input string is first partitioned into independent groups of 3 bits each. This partitioning should be unknown to the algorithm, but it should not change during the run. A 3-bit deceptive function is applied to each group of 3 bits and the contributions of all deceptive functions are added together to form the fitness. The 3-bit deceptive function is defined as follows:

$$dec_3(u) = \begin{cases} 1 & \text{if } u = 3 \\ 0 & \text{if } u = 2 \\ 0.8 & \text{if } u = 1 \\ 0.9 & \text{if } u = 0 \end{cases}, \tag{1}$$

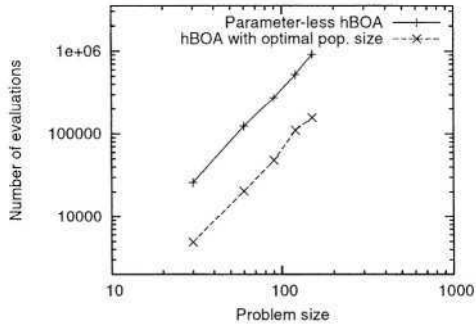


Fig. 3. Parameter-less hBOA and hBOA with the optimal population size on the deceptive function of order 3.

where u is the number of 1s in the input string of 3 bits. The 3-bit deceptive function is fully deceptive [17], which means that variation operators should not break interactions between bits in each group, because all statistics of lower order lead the algorithm away from the optimum. That is why most crossover operators as well as the model in UMDA will fail at solving this problem faster than in exponential number of evaluations, which is just as bad as with random search. Since deceptive functions bound a broad class of decomposable problems, performance of the parameter-less hBOA on this class of problems should indicate what performance can be expected on other decomposable problems.

Figure 3 shows the number of evaluations of the hierarchical BOA with optimal population size determined by the bisection method and the parameter-less hBOA on order-3 deceptive functions of $n = 30$ to $n = 150$ bits. The results indicate that the number of function evaluations for decomposable problems increases only by a near constant factor, and eliminating the parameters of hBOA thus does not affect scalability of hBOA on decomposable problems qualitatively.

5.3 Hierarchical Traps of Order 3

In an order-3 hierarchical trap with L levels, the total number of bits in a candidate solution is assumed to be an integer power of 3, $n = 3^L$. A candidate solution is evaluated on multiple levels and the overall value of the hierarchical trap is computed as the sum of contributions on all levels. On each level, a basis function similar to the deceptive function of order 3 is used:

$$basis(u, f_{lo}, f_{hi}) = \begin{cases} f_{hi} & \text{if } u = 3 \\ f_{lo} - u\frac{f_{lo}}{u-1} & \text{otherwise} \end{cases}, \quad (2)$$

On the lowest level, the candidate solution is partitioned into independent groups of 3 bits each, and each group is evaluated using the basis function with $f_{lo} = f_{hi} = 1$. The contributions of 000 and 111 are thus equally good, and 000 and

111 are superior to other combinations of 3 bits in any group. The contributions of all these groups are added together to form the overall contribution of the first level. Each group is then mapped (or interpreted) into a single symbol on the next level using the following interpretation function:

$$map(X) = \begin{cases} 0 & \text{if } X = 000 \\ 1 & \text{if } X = 111 \\ - & \text{otherwise} \end{cases}, \tag{3}$$

where X denotes the input 3 bits or symbols, and "–" is a null symbol. The second level thus contains $\frac{n}{3}$ symbols from $\{0,1,-\}$.

Symbols on the second level are also partitioned into independent groups of 3 bits each and each group contributes to the fitness on this level using the same basis function as on the first level, where $f_{hi} = f_{lo} = 1$. However, groups that contain the symbol "–" do not contribute to the fitness at all. The overall contribution of the second level is multiplied by 3 and added to the contribution of the first level. Each group is then mapped to the third level using the same interpretation function as was used to map the first level to the second one.

The same principle is used to evaluate and map each higher level except for the top level, which contains 3 symbols (the last 3 symbols are not mapped anymore). The contribution of kth level is always multiplied by 3^{k-1} so that the overall contribution of each level is of the same magnitude. The only difference when evaluating the top level of 3 symbols is that $f_{lo} = 0.9$, whereas $f_{hi} = 1$.

The global optimum of hierarchical traps is in the string of all 1s. However, blocks of 0s and 1s seem to be equally good on each level. Furthermore, any evolutionary algorithm is biased to solutions with many 0s, because the neighborhood of solutions with many 1s is inferior. Since hierarchical traps bound a broad class of hierarchical problems, performance of the parameter-less hBOA on this class problems should indicate what performance can be expected on other hierarchical problems.

Figure 4 shows the number of evaluations of the hierarchical BOA with optimal population size determined by the bisection method and the parameter-less hBOA on hierarchical traps of $n = 27, 81$, and 243 bits. The results indicate that the number of function evaluations for hierarchically decomposable problems increases only by a constant factor, and eliminating the parameters of hBOA does not affect scalability of hBOA on this class of problems qualitatively.

5.4 2D Ising ±J Spin Glasses

A 2D spin-glass system consists of a regular 2D grid containing n nodes. The edges in the grid connect nearest neighbors. Additionally, edges between the first and the last element in each dimension are added to introduce periodic boundary conditions.

With each edge, there is a real-valued constant associated with it called also a coupling constant. Each spin can obtain two values: +1 or –1. Given a set

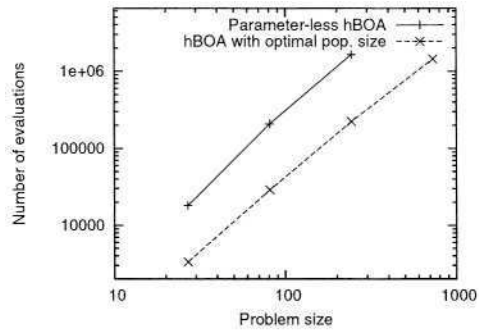


Fig. 4. Parameter-less hBOA and hBOA with the optimal population size on the hierarchical trap.

of coupling constants and spins, the energy of the spin glass system can be computed as

$$E(S) = \sum_{(i,j)\in E} s_i J_{i,j} s_j, \tag{4}$$

where $i \in \{0, 1, \ldots, n-1\}$, $j \in \{0, 1, \ldots, n-1\}$, E is the set of edges in the system, s_i and s_j denote the values of the spins i and j, respectively, and $J_{i,j}$ denotes the coupling constant between the spins i and j.

Here the task is to find the ground state for a 2D spin glass with specified coupling constants, where the ground state is the configuration of spins that minimizes the energy of the system given by Equation 4. Each spin glass configuration is represented by a string of bits where each bit corresponds to one spin: a 0 represents a spin −1, and a 1 represents a spin +1. We created 100 random spin glasses of sizes 6×6 (36 spins) to 14×14 (196 spins) by generating coupling constants to be +1 or −1 with equal probabilities. Spin glasses with coupling constants restricted to +1 and −1 are called $\pm J$ spin glasses.

Figure 5 shows the number of evaluations of the hierarchical BOA with optimal population size determined by the bisection method and the parameter-less hBOA on 2D $\pm J$ spin glasses. The good news is that the number of evaluations until convergence of the parameter-less hBOA still grows as a low-order polynomial with respect to the number of decision variables. The bad news is that, unlike for single-level and hierarchical traps, in this case the parameter-less population sizing does influence scalability of hBOA. More specifically, the order of the polynomial that approximates the number of evaluations increases by about 1.

Clearly, a linear factor by which the number of evaluations increases disagrees with the existing theory, which claims that the factor should be at most logarithmic [5]. Our hypothesis why this happens is that in this case the dynamics of hBOA with local search changes when increasing the population size. Using the hybrid method enables the two searchers—hBOA and the local searcher—

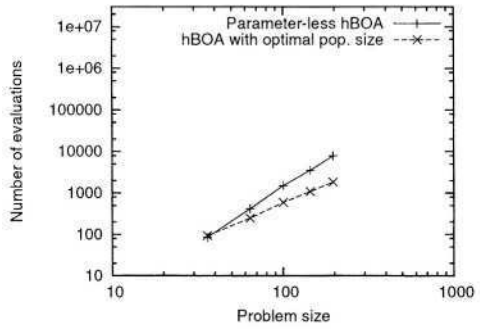


Fig. 5. Parameter-less hBOA and hBOA with the optimal population size on ± *J* spin glasses.

to cooperate and solve the problem faster. hBOA allows the local searcher to exit the enormous number of local optima, which make local search by itself intractable. On the other hand, the local searcher decreases the population sizes in hBOA significantly by making the algorithm to focus on regions with local optima. The performance of the hybrid then depends on how well the division of labor between the local and global searcher is done. Increasing the population size in hBOA with local search can influence the division of labor in the hybrid. Since the parameter-less hBOA simulates a number of populations of different sizes, we believe that the division labor is affected and more opportunities are given to hBOA at the expense of the local searcher. We are currently verifying the above hypothesis. The division of labor between local and global searcher can be also tuned by changing the maximum number of generations. However, to ensure that the algorithm will scale also exponentially scaled problems, where the number of generations grows linearly with the size of the problem, the maximum number of generations should not be limited by a function that grows slower than linearly.

6 Summary and Conclusions

This paper described, implemented, and tested the parameter-less hierarchical BOA. The parameter-less hBOA enables the practitioner to simply plug in the problem into hBOA without requiring the practitioner to first estimate an adequate population size or other problem-specific parameters. Despite the parameter-less scheme, low-order polynomial time complexity of hBOA on broad classes of optimization problems is retained. The parameter-less hBOA is thus a true black-box optimization algorithm, which can be applied to hierarchical and nearly decomposable problems without setting any parameters whatsoever. An interesting topic for future work is to develop techniques to improve hBOA performance via automatic tuning of other parameters, such as the maximum order of interactions in the probabilistic model or the window size in RTR.

Acknowledgments. M. Pelikan was supported by the Research Award at the University of Missouri at St. Louis and the Research Board at the University of Missouri at Columbia.

References

1. Pelikan, M., Goldberg, D.E.: Escaping hierarchical traps with competent genetic algorithms. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001) (2001) 511–518
2. Pelikan, M., Goldberg, D.E.: A hierarchy machine: Learning to optimize from nature and humans. Complexity **8** (2003)
3. Pelikan, M., Sastry, K., Goldberg, D.E.: Scalability of the Bayesian optimization algorithm. International Journal of Approximate Reasoning **31** (2002) 221–258
4. Harik, G., Lobo, F.: A parameter-less genetic algorithm. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99) **I** (1999) 258–265
5. Pelikan, M., Lobo, F.G.: Parameter-less genetic algorithm: A worst-case time and space complexity analysis. IlliGAL Report No. 99014, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL (1999)
6. Pelikan, M., Goldberg, D.E., Lobo, F.: A survey of optimization by building and using probabilistic models. Computational Optimization and Applications **21** (2002) 5–20
7. Larrañaga, P., Lozano, J.A., eds.: Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation. Kluwer, Boston, MA (2002)
8. Mühlenbein, H., paaß G.: From recombination of genes to the estimation of distributions I. Binary parameters. Parallel Problem Solving from Nature (1996) 178–187
9. Bosman, P.A.N., Thierens, D.: Continuous iterated density estimation evolutionary algorithms within the IDEA framework. Workshop Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000) (2000) 197–200
10. Chickering, D.M., Heckerman, D., Meek, C.: A Bayesian approach to learning Bayesian networks with local structure. Technical Report MSR-TR-97-07, Microsoft Research, Redmond, WA (1997)
11. Harik, G.R.: Finding multimodal solutions using restricted tournament selection. Proceedings of the International Conference on Genetic Algorithms (ICGA-95) (1995) 24–31
12. Pelikan, M.: Bayesian optimization algorithm: From single level to hierarchy. PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL (2002)
13. Mühlenbein, H., Schlierkamp-Voosen, D.: Predictive models for the breeder genetic algorithm: I. Continuous parameter optimization. Evolutionary Computation **1** (1993) 25–49
14. Thierens, D., Goldberg, D.: Convergence models of genetic algorithm selection schemes. Parallel Problem Solving from Nature (1994) 116–121
15. Ceroni, A., Pelikan, M., Goldberg, D.E.: Convergence-time models for the simple genetic algorithm with finite population. IlliGAL Report No. 2001028, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL (2001)
16. Ackley, D.H.: An empirical study of bit vector function optimization. Genetic Algorithms and Simulated Annealing (1987) 170–204
17. Deb, K., Goldberg, D.E.: Sufficient conditions for deceptive and easy binary functions. Annals of Mathematics and Artificial Intelligence **10** (1994) 385–408

Computational Complexity and Simulation of Rare Events of Ising Spin Glasses

Martin Pelikan[1], Jiri Ocenasek[2], Simon Trebst[2], Matthias Troyer[2], and Fabien Alet[2]

[1] Dept. of Math. and Computer Science, 320 CCB
University of Missouri at St. Louis
8001 Natural Bridge Rd., St. Louis, MO 63121
pelikan@cs.umsl.edu
[2] Computational Laboratory (CoLab)
Swiss Federal Institute of Technology (ETH)
CH-8092 Zürich, Switzerland
jirio@inf.ethz.ch
{trebst,troyer,falet}@comp-phys.org

Abstract. We discuss the computational complexity of random 2D Ising spin glasses, which represent an interesting class of constraint satisfaction problems for black box optimization. Two extremal cases are considered: (1) the $\pm J$ spin glass, and (2) the Gaussian spin glass. We also study a smooth transition between these two extremal cases. The computational complexity of all studied spin glass systems is found to be dominated by rare events of extremely hard spin glass samples. We show that complexity of all studied spin glass systems is closely related to Fréchet extremal value distribution. In a hybrid algorithm that combines the hierarchical Bayesian optimization algorithm (hBOA) with a deterministic bit-flip hill climber, the number of steps performed by both the global searcher (hBOA) and the local searcher follow Fréchet distributions. Nonetheless, unlike in methods based purely on local search, the parameters of these distributions confirm good scalability of hBOA with local search. We further argue that standard performance measures for optimization algorithms—such as the average number of evaluations until convergence—can be misleading. Finally, our results indicate that for highly multimodal constraint satisfaction problems, such as Ising spin glasses, recombination-based search can provide qualitatively better results than mutation-based search.

1 Introduction

The spin glass problem is an old-standing but still intensively studied problem in physics [1]. First, experimental realizations of spin glass systems do exist and their properties, in particular their dynamics, are still not well explained. Second, spin glasses pose a challenging, unsolved problem in theoretical physics since the nature of the spin glass state at low temperatures is not understood. It

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 36–47, 2004.

is widely believed that this is due to the intrinsic complexity of the rough energy landscape of spin glasses.

In statistical physics, one usual goal is to calculate a desired quantity (e.g. magnetization) over a distribution of configurations of a spin glass system for a given temperature. The probability of observing a specific spin configuration, C, of the spin glass is governed by the Boltzmann distribution, that is to say it is inversely proportional to the exponential of the ratio of its energy and temperature : $p(C) \sim \exp(-E(C)/T)$. Thus, as temperature decreases, the distribution of possible configurations of the spin glass concentrates near the configurations with minimum energy, which are also called ground states. The ground-state properties capture most of the low temperatures physics, and it is therefore very interesting to find and study them.

From another perspective, spin glasses represent an interesting class of problems for black-box optimization where the task is to find ground states of a given spin glass sample, because the energy landscape in most spin glasses exhibits features that make it a challenging optimization benchmark. One of these features is the large number of local optima, which often grows exponentially with the number of decision variables (spins) in the problem. Because of the large number of local optima, using local search operators, such as mutation, is almost always intractable.

In this paper we present, analyze, and discuss a series of experiments on 2D Ising spin glasses. Random spin glass instances for a fixed lattice geometry (square lattice) are generated by randomly sampling a fixed distribution of coupling constants. We distinguish two basic classes of random 2D Ising spin glass systems: (1) coupling constants are initialized randomly to either +1 or –1, and (2) coupling constants are generated from a zero-mean Gaussian distribution. A transition between these two cases is also considered. We apply the hierarchical Bayesian optimization algorithm (hBOA) with local search to all considered classes of spin glasses, and provide a thorough statistical analysis of hBOA performance on a large number of problem instances in each class. The results are discussed in the context of state-of-the-art Monte Carlo methods, such as the Wang-Landau algorithm [2] and the multicanonical method [3]. Finally, we identify important lessons from this work for genetic and evolutionary computation.

In the following we present a short review of the hierarchical Bayesian optimization algorithm and extremal value distributions used in the statistical analysis. In section 3 we define the 2D Ising spin glass systems analyzed in this work, and introduce several classes of random spin glass instances. Section 4 presents experimental methodology and results. Section 5 discusses experimental results. Finally, Section 6 summarizes and concludes the paper.

2 Numerical Methods and Statistical Analysis

This section briefly discusses the hierarchical Bayesian optimization algorithm (hBOA) [4,5] and extremal value distributions, which will be used to analyze experimental results.

2.1 Hierarchical Bayesian Optimization Algorithm (hBOA)

The hierarchical Bayesian optimization algorithm (hBOA) [4,5] is one of the most advanced genetic and evolutionary algorithms based primarily on selection and recombination. hBOA evolves a population of candidate solutions to a given problem. Using a *population* of solutions as opposed to a single solution has several advantages; for example, it enables simultaneous exploration of multiple regions in the search space, it can help to alleviate the effects of noise in evaluation, and it allows the use of statistical and learning techniques to identify regularities in the black-box optimization problem under consideration.

The first population of candidate solutions is usually generated according to uniform distribution over all candidate solutions. The population is updated for a number of iterations using two basic operators: (1) selection, and (2) variation. The selection operator selects better solutions at the expense of the worse ones from the current population, yielding a population of promising candidates. The variation operator starts by learning a probabilistic model of the selected solutions that encodes features of these promising solutions and the inherent regularities. hBOA uses Bayesian networks with local structures [6] to model promising solutions. The variation operator then proceeds by sampling the probabilistic model to generate new solutions. The new solutions are incorporated into the original population using the restricted tournament replacement (RTR) [7], which ensures that useful diversity in the population is maintained over long periods of time.

To improve candidate solutions locally, hBOA applies a deterministic bit-flip hill-climber to each newly generated candidate solution that improves the solution by single-bit flips until no further improvement is possible. Flips that produce better solutions are of higher priority. It was previously shown that local search can significantly reduce population sizes for various optimization problems, including the spin glass problem [8]. This paper extends the previously published results [8] on applying hBOA to spin glasses by providing an in-depth statistical analysis on a large number of random spin glass instances for different problem sizes, and studying two common classes of spin glasses as well as a transition between them.

2.2 Extremal Value Distributions

Several quantities related to the computational complexity studied in this work are found to follow extremal value distributions. The central limit theorem for extremal values states that the extremes of large samples are distributed according to one of three extremal value distributions, depending on whether their shapes

are fat-tailed (tails decay polynomially), exponential (tails decay exponentially), or thin-tailed (tails decay faster than exponentially) [9]. The integrated probability density function for any of these extremal value distributions can be written as

$$H_{\xi;\mu;\beta}(x) = \exp\left(-\left(1+\xi\frac{x-\mu}{\beta}\right)^{\frac{1}{\xi}}\right), \tag{1}$$

where μ is the location parameter, β is the scaling parameter, and ξ is the shape parameter that indicates how fast the tail decays. If $\xi < 0$, $H_{\xi;\mu;\beta}(x)$ represents the Fréchet distribution (polynomial decay), if $\xi = 0$ it represents the Gumbel distribution (exponential decay), and if $\xi > 0$ it represents the Weibull distribution (faster than exponential decay). Distributions encountered in this work are Fréchet distributions, where the shape parameter ξ determines the power law decay of the fat tails of the distribution

$$\frac{dH_{\xi;\mu;\beta}}{dx} \xrightarrow{x\to\infty} x^{-(1-1/\xi)} . \tag{2}$$

From this asymptotic behavior one can see that the m-th moment of a fat tailed Fréchet distribution (with $\xi < 0$) is well defined only if $|\xi| < 1/m$.

3 The Ising Spin Glass

A 2D spin glass system consists of a regular 2D grid containing N nodes which correspond to the spins. The edges in the grid connect nearest neighbors. Additionally, edges between the first and the last element in each dimension are added to introduce periodic boundary conditions.

With each edge there is a real-valued constant associated which gives the strength of spin-spin coupling. For the classical Ising model each spin can be in one of two states: +1 or –1. Each possible set of values for all spins is called a spin configuration. Given a set of (random) coupling constants, $J_{i,j}$, and a configuration of spins, C, the energy can be computed as

$$E(C) = \sum_{\langle i,j\rangle} s_i J_{i,j} s_j , \tag{3}$$

where $i, j \in \{0, 1, \ldots, N-1\}$ denote the spins (nodes) and $\langle i, j\rangle$ nearest neighbors on the underlying grid (allowed edges). The random spin-spin coupling constants $J_{i,j}$ for a particular spin glass instance are given on input.

In statistical physics, the usual task is to integrate a known function over all possible configurations of spins, where the configurations are distributed according to the Boltzmann distribution. Probability of encountering a configuration, C at temperature T is given by

$$p(C) = \frac{\exp\left(-E(C)/T\right)}{\sum_{\tilde{C}} \exp\left(-E(\tilde{C})/T\right)} . \tag{4}$$

From the physics point of view, it is interesting to know the ground states (configurations associated with the minimum possible energy). Finding extremal energies then corresponds to sampling the Boltzmann distribution with temperature approaching 0 and thus the problem of finding ground states is simpler *a priori* than integration over a wide range of temperatures. However, most of the conventional methods based on sampling the above Boltzmann distribution 4 fail to find the ground states configurations because they get often trapped in a local minimum.

The problem of finding ground states is a typical optimization problem, where the task is to find an optimal configuration of spins that minimizes energy. Although polynomial-time deterministic methods exist for both types of 2D spin glasses [10,11], most algorithms based on local search operators, including a (1+1) evolution strategy, conventional Monte Carlo simulations, and Monte Carlo simulations with Wang-Landau [2] or multicanonical sampling [3], scale exponentially and are thus impractical for solving this class of problems [12]. The origin for this slowdown is due to the suppressed relaxation times in the Monte Carlo simulations in the vicinity of the extremal energies because of the enormous number of local optima in the energy landscape. Recombination-based genetic algorithms succeed if recombination is performed in a way that interacting spins are located close to each other in the representation; k-point crossover with a rather small k can then be used so that the linkage between contiguous blocks of bits is preserved (unlike with uniform crossover, for instance). However, the behavior of such specialized representations and variation operators cannot be generalized to similar slowly equilibrating problems which exhibit different energy landscapes, such as protein folding or polymer dynamics.

In order to obtain a quantitative understanding of the disorder in a spin glass system introduced by the random spin-spin couplings, one generally analyzes a large set of random spin glass instances for a given distribution of the spin-spin couplings. For each spin glass instance the optimization algorithm is applied and results statistically analyzed to obtain a measure of computational complexity. Here we first consider two types of initial spin-spin coupling distributions, the $\pm J$ spin glass and the Gaussian spin glass.

3.1 The $\pm J$ Spin Glass

For the $\pm J$ Ising spin glass, each spin-spin coupling constant is set randomly to either +1 or –1 with equal probability (see lower right panel in Figure 1). Energy minimization in this case can be transformed into a constraint satisfaction problem, where the constraints relate spins connected by a coupling constant. If $J_{i,j} > 0$, then the constraint requires spins i and j to be different, whereas if $J_{i,j} < 0$, then the constraint requires spins i and j to be the same. Energy is minimized when the number of satisfied constraints is maximized.

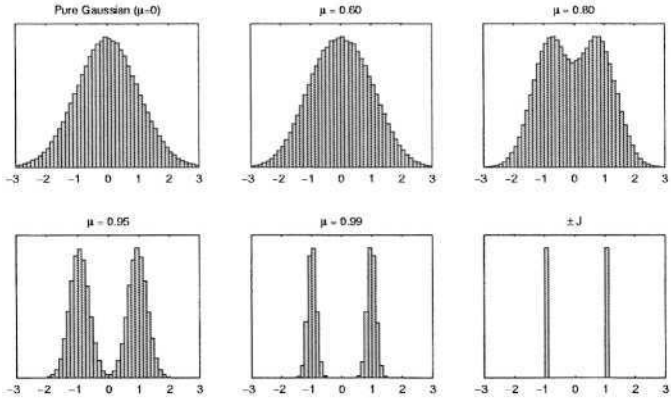


Fig. 1. Distribution of coupling constants for the transition from the Gaussian (upper left) to the $\pm J$ spin glass (lower right).

3.2 Gaussian Spin Glasses

In the Gaussian spin glass, coupling constants are generated according to a zero-mean Gaussian distribution with variance one (see upper left panel in Figure 1). For real-valued couplings, energy minimization can be casted as a constraint satisfaction problem with weighted constraints.

3.3 Transition between $\pm J$ and Gaussian Spin Glasses

To describe a smooth transition between the $\pm J$ and the Gaussian spin glass we vary the distribution of spin-spin coupling constants by defining a distribution as the sum of two Gaussian distributions, described by means, $\pm\tilde{\mu}$, and variance, $\tilde{\sigma}$, in such a way that the overall mean becomes $\mu = 0$ and the overall variance $\sigma = 1$. The explicit form of the two Gaussians is thus given by $\tilde{\sigma}^2 = 1 - \tilde{\mu}^2$. The $\pm J$ spin glass ($\tilde{\mu} = 1$) and the Gaussian spin glass ($\tilde{\mu} = 0$) then describe the extremal cases of this new family of distributions. The transition between the two extrema is then described by varying $\tilde{\mu}$ between 0 and 1 which is illustrated in Figure 1 for $\tilde{\mu} = 0, 0.60, 0.80, 0.95, 0.99, 1$.

4 Numerical Experiments

In the following we describe the numerical experiments in more detail and present results for the spin glasses described above.

4.1 Description of Experiments

For $\pm J$ and Gaussian 2D spin glasses, systems with equal number of spins in each dimension were used of size from $n = 8 \times 8$ to $n = 20 \times 20$. For each

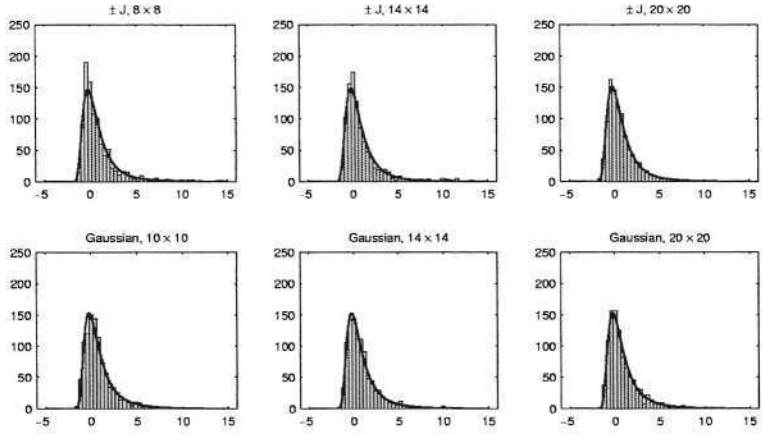


Fig. 2. Distribution of E_G for $\pm J$ spin glass systems of varying size. E_G and the density function are normalized using μ and β.

system size, 1000 random samples were generated. hBOA with the deterministic local searcher was then applied to find the ground state for each sample. For the transition from $\pm J$ to Gaussian spin glasses, we focused on a single system size, $n = 10 \times 10$.

For each spin glass sample, the population size in hBOA is set to the minimum population size required to find the optimum in 10 independent runs. The minimum population size is determined using bisection. The width of the final interval in bisection is at most 10% of its higher limit. Binary tournament selection without replacement is used. The windows size in RTR is set to the number of spins of the system under consideration, but it is always at most equal to 5% of the population size. The 5% cap on the window size is important to ensure fast convergence with even small populations. The cap explains the difference between the results presented here and the previous results, because populations are usually very small for hBOA with local search on Ising spin glasses [8].

Performance of hBOA was measured by (1) E_G, the total number of spin glass system configurations examined by hBOA (the number of restarts of the local searcher), and (2) E_L, the total number of steps of the local hill climber. Due to the lack of space, we only analyze E_G. E_L was greater than E_G by a factor of approximately $O(\sqrt{n})$. Clearly, we can expect that $E_G < E_L$. Nonetheless, it is computationally much less expensive to perform a local step in the hill climber than to evaluate a new spin glass configuration sampled by hBOA.

4.2 Results for $\pm J$ and Gaussian Couplings

The first important observation is that the distributions of E_G and E_L for all problem sizes and distributions of coupling constants follow Fréchet extremal value distributions. Applying a maximum likelihood estimator we can determine

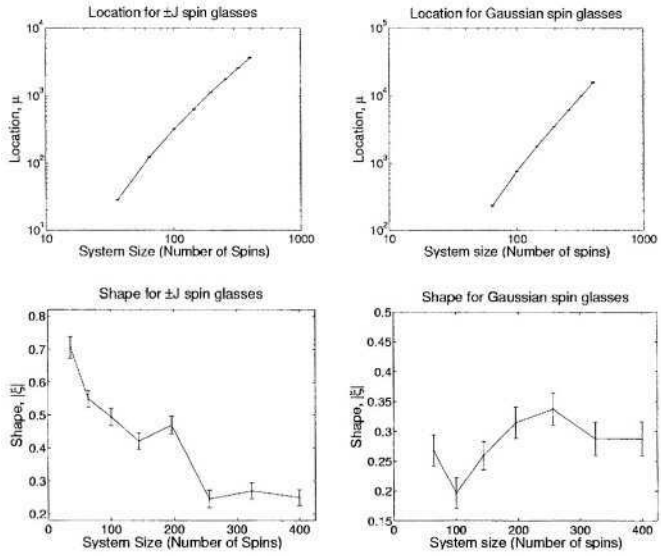


Fig. 3. Location μ and shape ξ for $\pm J$ and Gaussian spin glasses using maximum likelihood estimation. Standard error of the estimations displayed with error bars.

the parameters μ, β, and ξ of these distributions defined in Equation (1). Figure 2 shows the histograms and the corresponding probability density function for E_G for $\pm J$ spin glasses of various sizes.

The location parameter μ indicating the most likely value of E_G can be used to determine the scalability of hBOA. In Figures 3a and 3b the location parameter for both $\pm J$ and Gaussian spin glasses are shown versus the system size. Double logarithmic plots confirm that the location has an upper polynomial bound. For the $\pm J$ spin glass, the order of that polynomial approaches 1.5 as system size n grows, whereas for Gaussian couplings, the order of the polynomial seems to approach 2.2.

Figure 3c and 3d show the shape ξ for both $\pm J$ and Gaussian spin glasses with respect to the system size. Since it is always smaller than 1, we conclude that the mean is well-defined for all cases. For the variance (2nd moment) we find the shape parameter to be smaller than 1/2 only for systems larger than $n = 10 \times 10$. Thus, for system smaller than $n = 10 \times 10$ the variance is not well-defined and the mean has an infinite error.

4.3 Results for the Transition between $\pm J$ and Gaussian Couplings

For the transition between $\pm J$ and Gaussian couplings, E_G and E_L also follow Fréchet distributions. Figure 4 shows the distribution of E_G in the transition, including $\pm J$ and Gaussian cases. Figure 5 shows location and shape parameters for the transition.

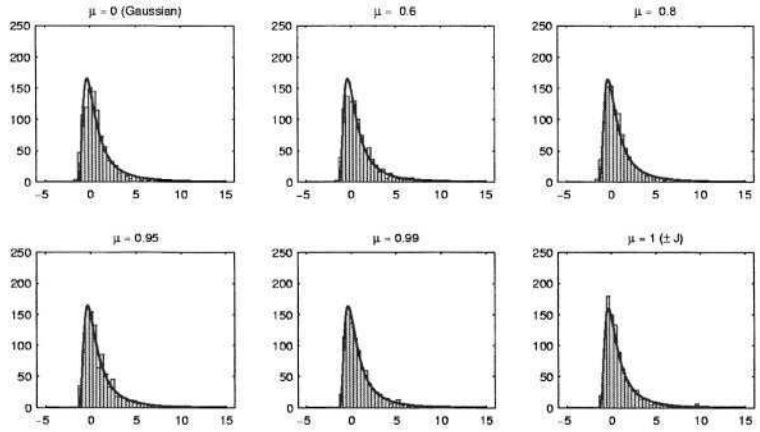


Fig. 4. Distribution of E_G for the transition from $\pm J$ to Gaussian spin glasses for $n = 10 \times 10$.

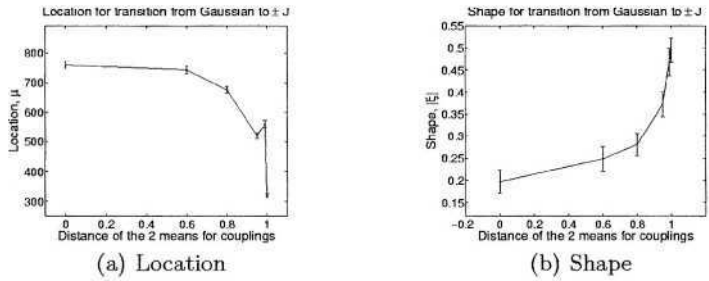


Fig. 5. Location μ and shape ξ for the transition between $\pm J$ and Gaussian spin glasses. Standard error of the estimations displayed with error bars. X-axis denotes the distance $\tilde{\mu}$ of the means used to generate couplings.

We can see that both location and shape parameters for the transition between $\pm J$ and Gaussian couplings lie between the corresponding parameters for the two extreme cases. That means that considering the two extreme cases provides insight not only in the cases themselves, but it can be used to guide estimation of parameters for a large class of other distributions of couplings.

5 Discussion

In the following we discuss the experimental results, first in the context of hBOA scalability theory and then in comparison with flat-histogram Monte Carlo results [12]. We close by presenting some general conclusions for genetic and evolutionary computation.

5.1 Experimental Results and hBOA Theory

An interesting question is whether the results obtained can be explained using hBOA convergence theory designed for a rather idealized situation, where the problem can be decomposed into subproblems of bounded order over multiple levels of difficulty. For random 2D Ising spin glasses, it can be shown that for a complete single-level decomposition it would be necessary to consider subproblems of order proportional to $\sqrt{n}$ as hypothesized by Mühlenbein [13], which would lead to exponentially sized populations [14]. Despite this, the number of function evaluations grows as a low-order polynomial of the number n of spins as predicted by hBOA scalability theory for decomposable problems of bounded difficulty [14]. Spin glasses with $\pm J$ couplings correspond to uniform scaling, where the theory predicts $O(n^{1.55})$ evaluations; indeed the location parameter μ indeed seems to approach a polynomial of order approx. 1.5. Spin glasses with Gaussian couplings exhibit a non-uniform scaling, where exponential scaling can be taken as a bounding case. For exponential scaling, the number of evaluations would be predicted to grow as $O(n^2)$; here the location parameter seems to grow slightly faster with a polynomial of order approx. 2.2. However, the order of this polynomial decreases with problem size.

5.2 Comparison to Flat-Histogram Monte Carlo

Monte Carlo (MC) methods are usually used to integrate a function $f(x)$ with some probability density distribution over the input parameter x. The common approach is to sample a series of values of x according to the specified probability distribution, and averaging the values of $f(x)$.

While conventional MC has been successfully used in numerous applications, it sometimes produces inferior results for low temperatures because the random walk through the space of all possible configurations (values of x) of the system has difficulties in overcoming energy barriers. One of the ways to alleviate this difficulty is to modify the simulated statistical-mechanical ensemble and use Wang-Landau sampling [2] to sample each energy level equally likely, thereby producing a flat histogram. The Wang-Landau algorithm thus represents a class of methods also known as flat-histogram MC. This approach not only alleviates the problem of energy barriers, but it also enables computation of the number of configurations at different energy levels, which can in turn be used to quickly compute thermal averages for any given temperature without having to rerun the simulation.

For flat-histogram MC, the distribution of round-trip times in energy measured by the total number of applications of local operators was recently shown to follow Fréchet distributions [12]. However, the absolute value of the shape parameter for flat-histogram MC was shown to approach 1. As a result, the mean of this distribution is not defined. Further, the location parameter found for flat-histogram MC grows exponentially [12], although for this class of spin glasses it is possible to analytically compute the entire energy spectrum in polynomial time, $O(n^{3.5})$ [10].

5.3 Important Lessons for Genetic and Evolutionary Computation

The results presented in this paper indicate that it can be misleading to estimate the mean convergence time by an average over several independent samples (runs), because in some cases the mean, variance, and other moments of the respective distribution may become ill-defined. In this work, the location parameter serves as a well-defined quantity to express computational complexity of various optimization and simulation techniques, including hBOA and flat-histogram MC. It can be expected that similar distribution will be observed for other evolutionary algorithms, as they reflect intrinsic properties of the spin glass [12].

Random 2D Ising spin glasses represent interesting classes of constraint satisfaction problems with a large number of local optima. The results presented in this work indicate that for such classes of problems, recombination-based search can provide optimal solutions in low-order polynomial time, whereas mutation-based methods scale exponentially. However, local search is still beneficial for local improvement of solutions in recombination-based evolutionary algorithms, because incorporating local search decreases population sizing requirements. A similar observation was found for MAXSAT [8].

6 Conclusions

Random classes of Ising spin glass systems represent an interesting class of constraint satisfaction problems for black-box optimization. Similar to flat-histogram MC, computational complexity of hBOA—expressed in the number of solutions explored by both hBOA and the local hill climber until the optimum—is found to show large sample-to-sample variations. The obtained distribution of optimization steps follow a fat-tailed Fréchet extremal value distribution. However, for hBOA the shape parameter defining the decay of the tail is small enough for the first two moments of the observed distributions to exist for all but smallest system sizes. The location parameter as well as the mean of this distribution scale like a polynomial of low order. The experiments show that similar behavior can be observed for $\pm J$ and Gaussian spin glasses, as well as for the transition between these two cases. For $\pm J$ spin glasses, performance of hBOA agrees with scalability theory for hBOA on uniformly scaled problems, whereas for Gaussian spin glasses, performance of hBOA agrees with scalability theory for hBOA on exponentially scaled problems.

There are some general conclusions for genetic and evolutionary computation. First, measuring time complexity by the average number of function evaluations until the optimum is found can sometimes be misleading when rare events dominate the sample-to-sample variations because the mean and other moments can be ill-defined. Second, it was shown for this specific problem that recombination-based search can efficiently deal with exponentially many local optima and still find the global optimum in low-order polynomial time.

Acknowledgments. Pelikan was supported by the Research Award at the University of Missouri at St. Louis and the Research Board at the University of Missouri. Trebst and Alet acknowledge support from the Swiss National Science Foundation. Most of the calculations were performed on the Asgard cluster at ETH Zürich. The hBOA software, used by Pelikan, was developed by Martin Pelikan and David E. Goldberg at the University of Illinois at Urbana-Champaign.

Ground states for $\pm J$ spin glass instances for $n = 6 \times 6$ to $n = 16 \times 16$ were verified with the results computed by S. Sabhapandit and S. N. Coppersmith of the University of Wisconsin, whereas the results on the remaining instances were verified using the spin glass ground state server at the University of Cologne.

References

1. Mézard, M., Parisi, G., Virasoro, M.A.: Spin glass theory and beyond. World Scientific, Singapore (1987)
2. Wang, F., Landau, D.P.: Efficient, multiple-range random walk algorithm to calculate the density of states. Physical Review Letters **86** (2001) 2050–2053
3. Berg, B.A., Neuhaus, T.: Multicanonical ensemble - a new approach to simulate first order phase-transition. Physical Review Letters **68** (1992)
4. Pelikan, M., Goldberg, D.E.: Escaping hierarchical traps with competent genetic algorithms. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001) (2001) 511–518
5. Pelikan, M., Goldberg, D.E.: A hierarchy machine: Learning to optimize from nature and humans. Complexity **8** (2003)
6. Chickering, D.M., Heckerman, D., Meek, C.: A Bayesian approach to learning Bayesian networks with local structure. Technical Report MSR-TR-97-07, Microsoft Research, Redmond, WA (1997)
7. Harik, G.R.: Finding multimodal solutions using restricted tournament selection. Proc. of the International Conference on Genetic Algorithms (ICGA-95) (1995) 24–31
8. Pelikan, M., Goldberg, D.E.: Hierarchical BOA solves Ising spin glasses and MAXSAT. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2003) **II** (2003) 1275–1286
9. Fisher, R.A., Tippett, L.H.C.: Limiting forms of the frequency distribution of the largest and smallest member of a sample. In: Proceedings of Cambridge Philosophical Society. Volume 24. (1928) 180–190
10. Galluccio, A., Loebl, M.: A theory of Pfaffian orientations. I. Perfect matchings and permanents. Electronic Journal of Combinatorics **6** (1999) Research Paper 6.
11. Galluccio, A., Loebl, M.: A theory of Pfaffian orientations. II. T-joins, k-cuts, and duality of enumeration. Electronic Journal of Combinatorics **6** (1999) Research Paper 7.
12. Dayal, P., Trebst, S., Wessel, S., Würtz, D., Troyer, M., Sabhapandit, S., Coppersmith, S.N.: Performance limitations of flat histogram methods. Physical Review Letters (in press)
13. Mühlenbein, H., Mahnig, T., Rodriguez, A.O.: Schemata, distributions and graphical models in evolutionary optimization. Journal of Heuristics **5** (1999) 215–247
14. Pelikan, M., Sastry, K., Goldberg, D.E.: Scalability of the Bayesian optimization algorithm. International Journal of Approximate Reasoning **31** (2002) 221–258

Fitness Inheritance in the Bayesian Optimization Algorithm

Martin Pelikan[1] and Kumara Sastry[2]

[1] Dept. of Math. and Computer Science, 320 CCB
University of Missouri at St. Louis
8001 Natural Bridge Rd., St. Louis, MO 63121
pelikan@cs.umsl.edu
[2] Illinois Genetic Algorithms Laboratory, 107 TB
University of Illinois at Urbana-Champaign
104 S. Mathews Ave. Urbana, IL 61801
kumara@illigal.ge.uiuc.edu

Abstract. This paper describes how fitness inheritance can be used to estimate fitness for a proportion of newly sampled candidate solutions in the Bayesian optimization algorithm (BOA). The goal of estimating fitness for some candidate solutions is to reduce the number of fitness evaluations for problems where fitness evaluation is expensive. Bayesian networks used in BOA to model promising solutions and generate the new ones are extended to allow not only for modeling and sampling candidate solutions, but also for estimating their fitness. The results indicate that fitness inheritance is a promising concept in BOA, because population-sizing requirements for building appropriate models of promising solutions lead to good fitness estimates even if only a small proportion of candidate solutions is evaluated using the actual fitness function. This can lead to a reduction of the number of actual fitness evaluations by a factor of 30 or more.

1 Introduction

To ensure reliable convergence to a global optimum, genetic and evolutionary algorithms (GEAs) must often maintain a large population of candidate solutions for a number of iterations. However, in many real-world problems, fitness evaluation is computationally expensive and evaluating even moderately sized populations of candidate solutions is intractable. For example, fitness evaluation may include a large finite element analysis, it may consist of a complex traffic simulation, or it may require interaction with a human expert.

This leads to an interesting question: Would it be possible to make GEAs evolve not only the population of candidate solutions, but also a model of fitness, which could be used to evaluate a certain proportion of newly generated candidate solutions (fitness inheritance)? Fortunately, the answer to the above question is positive, and a few studies have been made to support this argument. Methods were proposed for fitness inheritance in the simple genetic algorithm

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 48–59, 2004.

(GA) [1] and the univariate marginal distribution algorithm (UMDA) [2]. In both cases, the results were promising and suggested that fitness inheritance can significantly reduce the number of fitness evaluations.

The purpose of this paper is to propose a method that uses models of promising solutions developed by the Bayesian optimization algorithm (BOA) [3,4] to model the fitness landscape and estimate fitness of newly generated candidate solutions. Two types of models are considered: (1) traditional Bayesian networks with full conditional probability tables (CPTs) used in BOA and (2) Bayesian networks with local structures used in BOA with decision graphs (dBOA) [5] and the hierarchical BOA (hBOA) [6,7]. Since the model in BOA captures significant nonlinearities in the fitness landscape, using this model as the basis for developing a model of the fitness landscape seems to be a promising approach. Of course, other methods, such as neural networks or various regression models, could be used instead. The proposed method is examined on BOA with decision trees on three example problems: onemax, concatenated traps of order 4, and concatenated traps of order 5. The results indicate that fitness inheritance is beneficial in BOA even if only less than 1% of candidate solutions are evaluated using the actual fitness function. It turns out that due to the population sizing requirements for creating a correct model of promising solutions, the more fitness inheritance, the better.

The paper starts by discussing BOA and previous fitness inheritance studies. Section 4 presents the proposed method for fitness inheritance in BOA. Section 5 presents and discusses experimental results. Section 6 summarizes and concludes the paper.

2 Bayesian Optimization Algorithm

Probabilistic model-building genetic algorithms (PMBGAs) [8] replace traditional variation operators of genetic and evolutionary algorithms [9,10] by building a probabilistic model of promising solutions and sampling the model to generate new candidate solutions. The Bayesian optimization algorithm (BOA) [3] uses Bayesian networks to model candidate solutions.

BOA evolves a population of candidate solutions to the given problem. The first population of candidate solutions is usually generated randomly according to a uniform distribution over all solutions. The population is updated for a number of iterations using two basic operators: (1) selection, and (2) variation. The selection operator selects better solutions at the expense of the worse ones from the current population, yielding a population of promising candidates. The variation operator starts by learning a probabilistic model of the selected solutions that encodes features of these promising solutions and the inherent regularities. Bayesian networks are used to model promising solutions because Bayesian networks are among the most powerful tools for capturing and representing decomposition [11], which is an inherent feature of most complex real-world systems [12]. The variation operator then proceeds by sampling the probabilistic model to generate new solutions, which are incorporated into the

original population. Here, a simple replacement scheme is used where new solutions fully replace the original population. A more detailed description of BOA can be found in [4].

The remainder of this section discusses Bayesian networks, which are going to serve as the basis for developing the model of fitness in BOA.

2.1 Bayesian Networks

Bayesian networks (BNs) [13,14,15] are among the most popular graphical models, where statistics, modularity, and graph theory are combined in a practical tool for estimating probability distributions and inference. A Bayesian network is defined by two components: (1) a structure, and (2) parameters. The structure is encoded by a directed acyclic graph with the nodes corresponding to the variables in the modeled data set (in this case, to the positions in solution strings) and the edges corresponding to conditional dependencies. The parameters are represented by a set of conditional probability tables (CPTs) specifying a conditional probability for each variable given any instance of the variables that the variable depends on.

A Bayesian network encodes a joint probability distribution given by

$$p(X) = \prod_{i=1}^{n} p(X_i|\Pi_i), \tag{1}$$

where $X = (X_1, \ldots, X_n)$ is a vector of all the variables in the problem; Π_i is the set of parents of X_i (the set of nodes from which there exists an edge to X_i); and $p(X_i|\Pi_i)$ is the conditional probability of X_i given its parents Π_i.

A directed edge relates the variables so that in the encoded distribution, the variable corresponding to the terminal node is conditioned on the variable corresponding to the initial node. More incoming edges into a node result in a conditional probability of the variable with a condition containing all its parents. In addition to encoding dependencies, each Bayesian network encodes a set of independence assumptions. Independence assumptions state that each variable is independent of any of its antecedents in the ancestral ordering, given the values of the variable's parents.

To learn Bayesian networks, a greedy algorithm is usually used for its efficiency and robustness. The greedy algorithm starts with an empty Bayesian network. Each iteration then adds an edge into the network that improves quality of the network the most. Network quality can be measured by any popular scoring metric for Bayesian networks, such as the Bayesian Dirichlet metric with likelihood equivalence (BDe) [16,17] or the Bayesian information criterion (BIC) [18]. The learning is terminated when no more improvement is possible.

2.2 Conditional Probability Tables (CPTs)

Conditional probability tables (CPTs) store conditional probabilities $p(X_i|\Pi_i)$ for each variable X_i. The number of conditional probabilities for a variable that

is conditioned on k parents grows exponentially with k. For binary variables, for instance, the number of conditional probabilities is 2^k, because there are 2^k instances of k parents and it is sufficient to store the probability of the variable being 1 for each such instance. Figure 1 shows an example CPT for $p(X_1|X_2, X_3, X_4)$.

Nonetheless, the dependencies sometimes also contain regularities. Furthermore, the exponential growth of full CPTs often obstructs the creation of models that are both accurate and efficient. That is why Bayesian networks are often extended with local structures that allow more efficient representation of local conditional probability distributions than full CPTs [19,20].

2.3 Decision Trees and Graphs for Conditional Probabilities

Decision trees are among the most flexible and efficient local structures, where conditional probabilities of each variable are stored in one decision tree. Each internal (non-leaf) node in the decision tree for $p(X_i|\Pi_i)$ has a variable from Π_i associated with it and the edges connecting the node to its children stand for different values of the variable. For binary variables, there are two edges coming out of each internal node; one edge corresponds to 0, whereas the other edge corresponds to 1. For more than two values, either one edge can be used for each value, or the values may be classified into several categories and each category would create an edge.

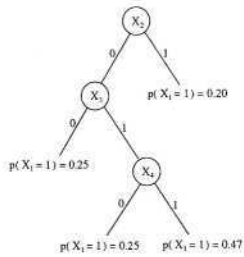
Each path in the decision tree for $p(X_i|\Pi_i)$ that starts in the root of the tree and ends in a leaf encodes a set of constraints on the values of variables in Π_i. Each leaf stores the value of a conditional probability of $X_i = 1$ given the condition specified by the path from the root of the tree to the leaf. A decision tree can encode the full conditional probability table for a variable with k parents if it splits to 2^k leaves, each corresponding to a unique condition. However, a decision tree enables more efficient and flexible representation of local conditional distributions. See Figure 1b for an example decision tree for the conditional probability table presented earlier.

A decision graph allows more edges to terminate in a single node. In other words, internal nodes in the decision tree are allowed to share children and, as a result, each node can have more than one parent. That makes this representation even more flexible. However, our experience indicates that, in BOA, decision graphs usually do not provide better performance than decision trees. See Figure 1c for an example decision graph.

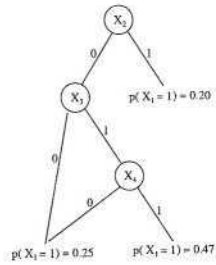
To learn Bayesian networks with decision trees, a decision tree for each variable X_i is initialized to an empty tree with a univariate probability of $X_i = 1$. In each iteration, each leaf of each decision tree is split to determine how quality of the current network improves by executing the split, and the best split is performed. The learning is finished when no splits improve the current network anymore. Quality of each model can be estimated using any popular scoring metric. Here we use a combination of the BDe [16,17] and BIG [18] metrics, where the BDe score is penalized with the number of bits required to encode parameters [4]. For decision graphs, a merge operation is introduced to allow for merging two leaves of any (but always the same) decision graph.

Π_1			$p(X_1 = 1 \mid \Pi_1)$
X_2	X_3	X_4	
0	0	0	0.25
0	0	1	0.25
0	1	0	0.25
0	1	1	0.47
1	0	0	0.20
1	0	1	0.20
1	1	0	0.20
1	1	1	0.20

(a) Conditional probability table.



(b) Decision tree.



(c) Decision graph.

Fig. 1. A conditional probability table for $p(X_1|X_2, X_3, X_4)$ using traditional representation (a) as well as local structures (b and c).

3 Previous Fitness Inheritance Studies

Despite the importance of fitness inheritance in robust population-based search, surprisingly few studies of fitness inheritance can be found. This section reviews the most important studies.

3.1 Fitness Inheritance in the Simple GA

Smith, Dike, and Stegmann [1] proposed two approaches to fitness inheritance in the simple GA [10]. The first approach is to compute the fitness of an offspring as the average fitness of its parents. The second approach is to consider a weighted average based on how similar the offspring is to each parent. The results indicated that GAs with fitness inheritance outperformed those without inheritance. However, the above study of fitness inheritance did not consider the effects of fitness inheritance on crucial GA parameters such as the population size and the number of generations. As a result, the speed-up achieved by using fitness inheritance could not be estimated properly.

Zhang, Julstrom, and Chen [21] used the aforementioned fitness inheritance model in the simple GA for design of vector quantization codebooks.

3.2 Fitness Inheritance in PMBGAs

Sastry, Goldberg, and Pelikan [2] considered the univariate marginal distribution algorithm (UMDA), which is one of the simplest PMBGAs. Using fitness inheritance in UMDA introduces new challenges, because UMDA does not use two-parent recombination and therefore it is difficult to find direct correspondence between parents and their offspring. Instead, Sastry et al. extend the probabilistic model to allow for estimating fitness of newly sampled candidate solutions.

UMDA models the population of promising solutions after selection using the probability vector, which stores the probability of a 1 at each position. These

probabilities are then used to sample new candidate solutions. To incorporate fitness inheritance, the probability vector $p = (p_1, p_2, \ldots, p_n)$ is extended to include additional two statistics $\bar{f}(X_i = 0)$ and $\bar{f}(X_i = 1)$ for each string position i. The term $\bar{f}(X_i = 0)$ denotes the average fitness of all solutions where the ith bit is 0; analogously, the term $\bar{f}(X_i = 1)$ denotes the average fitness of solutions with the ith bit equal to 1. The fitness of each new solution can then estimated as

$$f_{est}(X_1, X_2, \ldots, X_n) = \bar{f} + \sum_{i=1}^{n} \left(\bar{f}(X_i) - \bar{f} \right), \tag{2}$$

where $\bar{f}$ is the average fitness of all solutions used to estimate the fitness.

Sastry et al. [2] developed theory for UMDA on onemax that estimates the number of actual fitness evaluations when a given proportion of candidate solutions inherits fitness, whereas the remaining candidate solutions are evaluated using the actual fitness. The basic idea is to start by adapting the population sizing and time to convergence models to UMDA with fitness inheritance, and relate these quantities to their counterparts in standard UMDA. If optimal population size is used in both cases, Sastry et al. showed that only about 20% evaluations can be saved. However, if the same population size is used in both cases, the number of evaluations decreases by a factor of more than three.

4 Modeling Fitness in BOA

This section describes how the fitness model is built and updated using Bayesian networks, and how new candidate solutions can be evaluated using the model. Both Bayesian networks with full CPTs as well as the ones with local structures are discussed. The section also discusses where the statistics can be acquired from to build an accurate fitness model.

4.1 Modeling Fitness Using Bayesian Networks

In UMDA, probabilities of a 1 at each position that form the probability vector are each coupled with an average fitness of a 0 and a 1 at that position. Analogically, Bayesian networks can be extended to incorporate an average fitness of a 0 and a 1 for each statistic stored by the model.

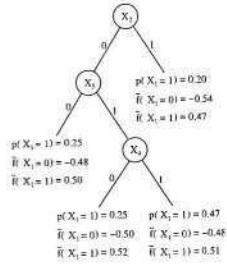
In BOA, for every variable X_i and each possible value x_i of X_i, an average fitness of solutions with $X_i = x_i$ must be stored for each instance π_i of X_i's parents Π_i. In the binary case, each row in the conditional probability table is thus extended by two additional entries. Figure 2a shows an example conditional probability table extended with fitness information based on the conditional probability table presented in Figure 1a. The fitness can then be estimated as

$$f_{est}(X_1, X_2, \ldots, X_n) = \bar{f} + \sum_{i=1}^{n} \left(\bar{f}(X_i|\Pi_i) - \bar{f}(\Pi_i) \right), \tag{3}$$

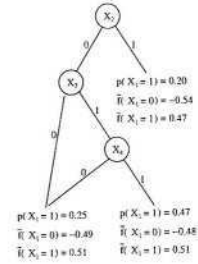
where $\bar{f}(X_i|\Pi_i)$ denotes the average fitness of solutions with X_i and Π_i, and $\bar{f}(\Pi_i)$ is the average fitness of all solutions with Π_i. Clearly,

Π_1			$p(X_1 = 1 \mid \Pi_1)$	$\bar{f}(X_1 = 0 \mid \Pi_1)$	$\bar{f}(X_1 = 1 \mid \Pi_1)$
X_2	X_3	X_4			
0	0	0	0.25	−0.47	0.51
0	0	1	0.25	−0.52	0.49
0	1	0	0.25	−0.50	0.53
0	1	1	0.47	−0.37	0.47
1	0	0	0.20	−0.63	0.58
1	0	1	0.20	−0.45	0.46
1	1	0	0.20	−0.49	0.73
1	1	1	0.20	−0.51	0.62

(a) Conditional probability table.



(b) Decision tree.



(c) Decision graph.

Fig. 2. Fitness inheritance in a conditional probability table for $p(X_1|X_2, X_3, X_4)$ (a) and its representation using local structures (b and c).

$$\bar{f}(\Pi_i) = \sum_{X_i} p(X_i|\Pi_i)\bar{f}(X_i|\Pi_i). \tag{4}$$

4.2 Modeling Fitness Using Bayesian Networks with Decision Graphs

A similar method as for full CPTs can be used to incorporate fitness information into Bayesian networks with decision trees or graphs. The average fitness of each instance of each variable must be stored in every leaf of a decision tree or graph. Figure 2 shows an example decision tree and graph extended with fitness information based on the decision tree and graph presented earlier in Figure 1. The fitness averages in each leaf are restricted to solutions that satisfy the condition specified by the path from the root of the tree to the leaf.

Since decision graphs enable better encoding of statistical dependencies in the selected population of selected solutions, where the statistical dependencies originate in nonlinearities in the fitness function [22], using decision graphs for fitness inheritance should also improve fitness inheritance.

4.3 Where to Inherit Fitness from?

We still have not faced the following question: Where to obtain information to compute statistics used for fitness inheritance? More specifically, for each instance x_i of X_i and each instance π_i of X_i's parents Π_i, we must compute the average fitness of all solutions with $X_i = x_i$ and $\Pi_i = \pi_i$. Here we use two sources for computing the fitness-inheritance statistics:

1. Selected parents that were evaluated using the actual fitness function, and
2. the offspring that were evaluated using the actual fitness function.

The reason for restricting computation of fitness-inheritance statistics to selected parents and offspring is that the probabilistic model used as the basis for

selecting relevant statistics represents nonlinearities in the population of parents and the population of offspring. Since it is best to maximize learning data available, it seems natural to use these two populations to compute the fitness-inheritance statistics. The reason for restricting input for computing these statistics to solutions that were evaluated using the actual fitness function is that the fitness of other solutions was estimated only and it involves errors that could mislead fitness inheritance and propagate through generations. Both using only those solutions that were evaluated using the actual fitness function and incorporating the offspring in estimating inheritance statistics differs from previous fitness inheritance studies [1,2].

5 Experiments

This section describes experiments and provides experimental results. Test problems are described first. Next, experimental results are presented and discussed.

5.1 Onemax

Onemax is a simple linear function that computes the sum of bits in the input binary string:

$$f_{onemax}(X_1, X_2, \ldots, X_n) = \sum_{i=1}^{n} X_i, \tag{5}$$

where $(X_1, X_2, \ldots, X_n)$ denotes the input binary string of n bits. In onemax, the fitness contribution of each bit is independent of its context. That is why a simple model used in UMDA that considers each variable independently of other variables suffices and yields convergence to the optimum in about $O(n \log n)$ evaluations. However, any other models of bounded complexity should work well, and practically any crossover operator used in standard GAs should also suffice.

In the model of fitness developed by BOA, the average fitness of a 1 in any leaf should be approximately 0.5, whereas the average fitness of a 0 in any leaf should be approximately –0.5. As a result, solutions will get penalized for 0s, while they would be rewarded for 1s. The average fitness will vary throughout the run. This paper considers onemax of $n = 50$ bits.

5.2 Concatenated 4-Bit Trap

In concatenated 4-bit traps [23,24], the input string is first partitioned into independent groups of 4 bits each. This partitioning should be unknown to the algorithm, but it should not change during the run. A 4-bit trap function is applied to each group of 4 bits and the contributions of all traps are added together to form the fitness. Each 4-bit trap is defined as follows:

$$trap_4(u) = \begin{cases} 4 & \text{if } u = 4 \\ 3 - u & \text{otherwise} \end{cases}, \tag{6}$$

where u is the number of 1s in the input string of 4 bits. An important feature of traps is that in each of the 4-bit traps, all 4 bits must be treated together, because all statistics of lower order lead the algorithm away from the optimum [24]. That is why most crossover operators as well as the model in UMDA will fail at solving this problem faster than in exponential number of evaluations, which is just as bad as blind search.

Unlike in onemax, $\bar{f}(X_i = 0)$ and $\bar{f}(X_i = 1)$ depend on the state of the search because the distribution of contexts of each bit changes over time and bits in a trap are not independent. The context of each leaf also determines whether $\bar{f}(X_i = 0) < \bar{f}(X_i = 1)$ or $\bar{f}(X_i = 0) > \bar{f}(X_i = 1)$ in the leaf. This paper considers a trap consisting of 10 copies of the 4-bit trap, where the total number of bits is $n = 40$.

5.3 Concatenated 5-Bit Trap

Concatenated traps of order 5 can be defined analogically to traps of order 4, but instead of dealing with groups of 4 bits, groups of 5 bits are considered. The contribution of each group of 5 bits is computed as

$$trap_5(u) = \begin{cases} 5 & \text{if } u = 5 \\ 4 - u & \text{otherwise} \end{cases}, \tag{7}$$

where u is the number of 1s in the input string of 5 bits. Traps of order 5 also necessitate that all bits in each group are treated together, because statistics of lower order are misleading.

Average fitness values $\bar{f}(X_i)$ depend on context similarly as for traps of order 4, and they thus follow similar dynamics. This paper considers a trap consisting of 10 copies of the 5-bit trap, where the total number of bits is $n = 50$.

5.4 Experimental Results

On each test problem, the following fitness inheritance proportions were considered: 0 to 0.9 with step 0.1, 0.91 to 0.99 with step 0.01, and 0.991 to 0.999 with step 0.001. For each test problem and fitness inheritance proportion, 30 independent experiments were performed. Each experiment consisted of 10 independent runs with the minimum population size to ensure convergence to a solution within 10% of the optimum (i.e., with at least 90% correct bits) in all 10 runs. For each experiment, bisection method was used to determine the minimum population size, and the number of evaluations (excl. the evaluations done using the model of fitness) was recorded. The average of 10 runs in all experiments was then computed and displayed as a function of the proportion of candidate solutions for which fitness was estimated using the fitness model.

The results on onemax, traps of order 4, and traps of order 5, are shown in figures 3 and 4. In all experiments, the number of actual fitness evaluations decreases with the inheritance proportion and it reaches the optimum when the proportion of candidate solutions for fitness inheritance is more than 99%. That means that, considering only the actual fitness evaluations, evaluating less than

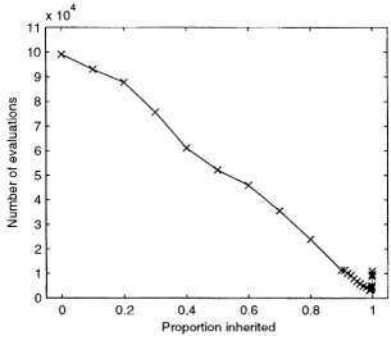
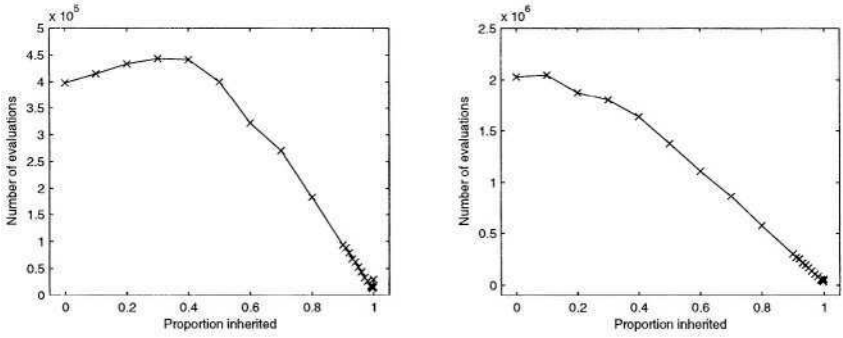


Fig. 3. Results on a 50-bit onemax.



(a) A 40-bit trap of order 4. (b) A 50-bit trap of order 5.

Fig. 4. Results on concatenated traps of order 4 and 5.

1% of candidate solutions with the actual fitness seems to be beneficial. As a result, the number of evaluations of the actual fitness can be decreased by a factor of more than 31 for onemax, 32 for the trap of order 4, and 53 for the trap of order 5. Although the actual savings depend on the problem considered, it can be expected that fitness inheritance enables significant reduction of fitness evaluations on many problems because deceptive problems of bounded difficulty test BOA on the boundary of its design envelope to determine if BOA can solve a class of nearly decomposable problems [25].

Considering only the actual fitness evaluations ignores time complexity of selection, model construction, generation of new candidate solutions, and fitness estimation. Combining these factors with the complexity estimate for the actual fitness evaluation can be used to compute the optimal proportion of candidate solutions to evaluate using fitness inheritance. Nonetheless, the results presented in this paper clearly indicate that using fitness inheritance in BOA can reduce the number of solutions that must be evaluated using the actual fitness function by a factor of 30 or more. Consequently, if fitness evaluation is a bottleneck, there is a lot of space for improvement using fitness inheritance in BOA.

6 Summary and Conclusions

Fitness inheritance enables genetic and evolutionary algorithms to evaluate only a certain proportion of candidate solutions using the actual fitness function, while the fitness of remaining solutions is computed using a model of the fitness landscape updated on the fly. Using fitness models that can be updated and used efficiently can significantly speed up solution to problems where fitness evaluation is computationally expensive.

This paper showed that while fitness inheritance yields only moderate speed-ups of about 20% in simple GAs and UMDA, in BOA the benefits of using fitness inheritance become more significant. Due to rather large population-sizing requirements for creating an adequate probabilistic model of promising solutions in BOA, the number of actual function evaluations decreases even if less than 1% of candidate solutions are evaluated using the actual fitness function, while the fitness of the remaining solutions is estimated using only its model. That is an important result, because BOA and other advanced PMBGAs often require large populations, and evaluating large populations can become intractable for problems with computationally expensive fitness evaluation.

An important topic for future work is to incorporate fitness inheritance in presence of niching, which can lead to accumulation of candidate solutions whose fitness is overestimated. Resolving this problem would enable the use of fitness inheritance in the hierarchical BOA (hBOA) [6,7], which combines BOA with local structures and niching. Another important topic is to develop theory that extends theoretical work on fitness inheritance in UMDA to BOA and other competent GAs. Finally, it is important to apply the proposed fitness inheritance model to solve challenging real-world problems with expensive fitness evaluation.

Acknowledgments. The authors would like to thank David E. Goldberg for discussions and comments. A part of this work was supported by the Research Award at the University of Missouri at St. Louis and the Research Board at the University of Missouri. Most experiments were done on Asgard cluster at the Institute of Theoretical Physics at the Swiss Federal Institute of Technology (ETH) Zürich. The hBOA software, used by Pelikan, was developed by Martin Pelikan and David E. Goldberg at the University of Illinois at Urbana-Champaign.

References

1. Smith, R.E., Dike, B.A., Stegmann, S.A.: Fitness inheritance in genetic algorithms. Proceedings of the ACM Symposium on Applied Computing (1995) 345–350
2. Sastry, K., Goldberg, D.E., Pelikan, M.: Don't evaluate, inherit. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001) (2001) 551–558 Also IlliGAL Report No. 2001013.
3. Pelikan, M., Goldberg, D.E., Cantú-Paz, E.: BOA: The Bayesian optimization algorithm. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99) **I** (1999) 525–532 Also IlliGAL Report No. 99003.
4. Pelikan, M.: Bayesian optimization algorithm: From single level to hierarchy. PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL (2002)

5. Pelikan, M., Goldberg, D.E., Sastry, K.: Bayesian optimization algorithm, decision graphs, and Occam's razor. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001) (2001) 519–526
6. Pelikan, M., Goldberg, D.E.: Escaping hierarchical traps with competent genetic algorithms. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001) (2001) 511–518 Also IlliGAL Report No. 2000020.
7. Pelikan, M., Goldberg, D.E.: A hierarchy machine: Learning to optimize from nature and humans. Complexity **8** (2003)
8. Pelikan, M., Goldberg, D.E., Lobo, F.: A survey of optimization by building and using probabilistic models. Computational Optimization and Applications **21** (2002) 5–20
9. Holland, J.H.: Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor, MI (1975)
10. Goldberg, D.E.: Genetic algorithms in search, optimization, and machine learning. Addison-Wesley, Reading, MA (1989)
11. Jordan, M.I., ed.: Learning in Graphical Models. MIT Press (1998)
12. Simon, H.A.: The Sciences of the Artificial. The MIT Press, Cambridge, MA (1968)
13. Howard, R.A., Matheson, J.E.: Influence diagrams. In Howard, R.A., Matheson, J.E., eds.: Readings on the principles and applications of decision analysis. Volume II. Strategic Decisions Group, Menlo Park, CA (1981) 721–762
14. Pearl, J.: Probabilistic reasoning in intelligent systems: Networks of plausible inference. Morgan Kaufmann, San Mateo, CA (1988)
15. Buntine, W.L.: Theory refinement of Bayesian networks. Proceedings of the Uncertainty in Artificial Intelligence (UAI-91) (1991) 52–60
16. Cooper, G.F., Herskovits, E.H.: A Bayesian method for the induction of probabilistic networks from data. Machine Learning **9** (1992) 309–347
17. Heckerman, D., Geiger, D., Chickering, D.M.: Learning Bayesian networks: The combination of knowledge and statistical data. Technical Report MSR-TR-94-09, Microsoft Research, Redmond, WA (1994)
18. Schwarz, G.: Estimating the dimension of a model. The Annals of Statistics **6** (1978) 461–464
19. Chickering, D.M., Heckerman, D., Meek, C.: A Bayesian approach to learning Bayesian networks with local structure. Technical Report MSR-TR-97-07, Microsoft Research, Redmond, WA (1997)
20. Friedman, N., Goldszmidt, M.: Learning Bayesian networks with local structure. In Jordan, M.I., ed.: Graphical models. MIT Press, Cambridge, MA (1999) 421–459
21. Zheng, X., Julstrom, B., Cheng, W.: Design of vector quantization codebooks using a genetic algorithm. In: Proceedings of the International Conference on Evolutionary Computation (ICEC-97), Picataway, NJ, IEEE Press (1997) 525–529
22. Pelikan, M., Sastry, K., Goldberg, D.E.: Scalability of the Bayesian optimization algorithm. International Journal of Approximate Reasoning **31** (2002) 221–258
23. Ackley, D.H.: An empirical study of bit vector function optimization. Genetic Algorithms and Simulated Annealing (1987) 170–204
24. Deb, K., Goldberg, D.E.: Sufficient conditions for deceptive and easy binary functions. Annals of Mathematics and Artificial Intelligence **10** (1994) 385–408
25. Goldberg, D.E.: The design of innovation: Lessons from and for competent genetic algorithms. Volume 7 of Genetic Algorithms and Evolutionary Computation. Kluwer Academic Publishers (2002)

Limit Cycle Prediction in Multivariable Nonlinear Systems Using Genetic Algorithms

Farzan Rashidi[1] and Mehran Rashidi[2]

[1] Control Research Department, Engineering Research Institute, Tehran, Iran
P.O.Box: 13445-754, Tehran
f.rashidi@ece.ut.ac.ir
[2] Hormozgan Regional Electric Co., Bandar-Abbas, Iran
mrashidi@mehr.sharif.edu

Abstract. This paper presents an intelligent method based on multiuobjective genetic algorithm (MOGA) for prediction of limit cycle in multivariable nonlinear systems. First we address how such the systems may be investigated using the Single Sinusoidal Input Describing Function (SIDF) philosophy. The extension of the SIDF to multi loop nonlinear systems is presented. For the class of separable nonlinear element of any general form, the harmonic balance equations are derived. A numerical search based on multiobjective genetic algorithm is addressed for the direct solution of the harmonic balance system matrix equation. The MOGA is employed to solve the multiobjective formulation and obtain the quantitative values for amplitude, frequency and phase difference of possible limit cycle operation. The search space of MOGA is the space of the possible limit cycle parameters, such as amplitudes, frequency and phase difference between the interacting loops. Finally computer simulation is performed to show how the analysis given in the paper is used to predict the existence of the limit cycle of the multivariable nonlinear systems.

1 Introduction

The frequency response method is a powerful tool for the analysis and design of linear control systems. It is based on describing a linear system by a complex-valued function instead of differential equation. However, frequency domain analysis can not be directly applied to nonlinear systems because frequency response functions can not be defined for nonlinear systems. The Describing Function (DF) method is an extended version of the frequency response method, which can be used to approximately analyze and predict nonlinear behavior[3,2]. The main use of describing function method is for the prediction of limit cycles (oscillations) in nonlinear systems [1]. A limit cycle is the phenomenon that can be observed in systems composed of nonlinear elements. The phenomenon is of fundamental importance in nonlinear systems and, as far as the design of a nonlinear system is concerned, it should be considered along with the stability analysis [6]. The applicability of DF to limit cycle analysis is due to the fact

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 60–68, 2004.

that the form of the signals in the limit-cycling system is usually approximately sinusoidal. In fact, we assume that the linear part of the system has low-pass properties, which can attenuate the harmonics of the nonlinear system.
In this paper the describing function method is extended to multi-loop systems and the MOGA formulation is designed to numerically search for limit cycles. The computer simulations are performed to show how the proposed method is used to predict the existence of the limit cycle of the nonlinear multivariable systems.
The configuration of this paper is as follow: Section 2 offers a brief summary of the sinusoidal input describing function theory, because it will be used in the sequel. In section 3 the definition of limit cycle as a characteristic of nonlinear systems is presented. The extension of the philosophy of the harmonic balance equation to multivariable nonlinear systems will discussed in section 4. Several approaches have been formulated to analyze and predict the limit cycle oscillation in nonlinear systems, in section 5 a brief review of multiobjective genetic algorithm that we will use for predicting of amplitude, frequency and phase of limit cycle of nonlinear MIMO systems, is presented. Simulation results and some concluding remarks is discussed in next sections.

2 Sinusoidal Input Describing Function

The Sinusoidal Input Describing Function (SIDF) approach generally can be used to study periodic phenomena. It is applied for two primary purposes: limit cycle analysis and characterizing the input/output behavior of a nonlinear plant in the frequency domain. In short a SIDF describes the amplitude and phase of the first harmonic of the periodic output signal of the nonlinear system with respect to the sinusoidal input signal [4]. Due to the characteristics of a nonlinear system, the SIDF will be dependent on both the amplitude and the frequency of the input signal. The SIDF can easily be measured with a Dynamic Signal Analyzer, which generates a sinusoidal source signal with a presanctified amplitude and frequency. The corresponding amplitude and phase of the output signal are computed online, resulting in the SIDF for the specified input amplitude and frequency. In order to develop the basic version of the describing function method, the system has to satisfy the following conditions [5]:

1. There is only a single nonlinear component and the system can be rearranged into the form shown in Figure 1.
2. The nonlinear component is time-invariant.
3. Corresponding to a sinusoidal input, only the fundamental component of the output is considered.
4. The nonlinearity is odd.

If the input of the nonlinear system is a sine wave $e(t) = A\sin(\omega t)$, then the output is periodic and can be expressed as:

$$u(t) = \frac{a_0}{2} + \sum_{i=1}^{\infty}[a_n \cos(n\omega t) + b_n \sin(n\omega t)] \tag{1}$$

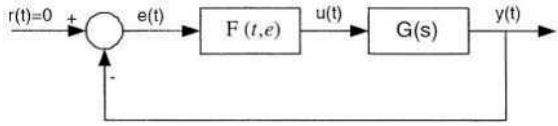


Fig. 1. Nonlinear System for Describing Function Analysis

where the coefficients a_i's and b_i's are generally functions of A and ω determined by:

$$a_0 = \frac{1}{\pi}\int_{-\pi}^{\pi} u(\theta)d\theta$$
$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} u(\theta)\cos(n\theta)d\theta$$
$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} u(\theta)\sin(n\theta)d\theta \tag{2}$$

where $\theta = \omega t$. Because of our assumptions, $a_0 = 0$, $n = 1$, and

$$u(t) = a_1\cos(\omega t) + b_1\sin(\omega t) = M(A,\omega)\sin(\omega t + \varphi(A,\omega)) \tag{3}$$

where

$$M(A,\omega) = \sqrt{a_1^2 + b_1^2} \quad \text{and} \quad \varphi(A,\omega) = \arctan(\frac{a_1}{b_1}) \tag{4}$$

From the above equations it can be seen that the fundamental component of the output corresponding to a sinusoidal input is a sinusoid of the same frequency and can be written as:

$$M(A,\omega)e^{j(\omega t+\varphi(A,\omega))} = (b_1 + ja_1)e^{j(\omega t)} \tag{5}$$

The describing function of the nonlinear element is defined as the complex ratio of the fundamental component of the nonlinear element by the input sinusoid, i.e.

$$N(A,\omega) = \frac{M(A,\omega)e^{j(\omega t+\varphi(A,\omega))}}{A\sin(\omega t)} = \frac{1}{A}(b_1 + ja_1) \tag{6}$$

By replacing the nonlinear element, *F(e)* ,in figure 1, with its describing function, $N(A,\omega)$, the nonlinear element can be treated as if it where a linear element with a frequency response function. As can be seen from equation (6), Generally, the describing function depends on the frequency and amplitude of the input signal.

3 Limit Cycle Prediction for Nonlinear Systems

A limit cycle is a periodic signal, $x_{LC}(t+T) = x_{LC}(t)$ for all t and some T (the period) such that perturbed solutions either approach x_{LC} (a stable limit

cycle) or diverge from it (an unstable one). An approach to limit cycle analysis that has gained widespread acceptance is the frequency-domain / SIDF method. In Figure 1 if we replace *F(e)* with $N(A,\omega)$ and assume that a self-sustained oscillation of amplitude A and frequency ω exists in the system then for $r = 0$, $y \neq 0$, we have:

$$N(A,\omega)G(j\omega) + 1 = 0 \quad \text{or} \quad N(A,\omega)G(j\omega) = -1 \tag{7}$$

This equation, called the *harmonic balance equation* [7]. If any limit cycles exist in our system, and the four assumptions (mentioned in section 2) are satisfied, then the amplitude and frequency of the limit cycles can be predicted by solving the *harmonic balance equation.* If there are no solutions to the harmonic balance equation then the system will have no limit cycles (under the above assumptions). It is generally very difficult to solve this equation by analytical methods.

3.1 Limit Cycle Prediction for Nonlinear Multivariable Systems

Provided that certain limitation are placed on the form of the linear system elements, the extension of the philosophy of harmonic linearization to multi-variable systems is conceptually straightforward and has been suggested by several authors [1,2]. The equation governing limit cycle operation in the autonomous multi-variable nonlinear feedback system of figure 1 can be expressed as:

$$det\Big(\tilde{N}(A,\omega)G(j\omega) + I\Big) = 0 \tag{8}$$

Where $\tilde{N}(A,\omega)$ is a matrix of single sinusoidal input describing functions corresponding to the nonlinear elements of $N(A,\omega)$. Thus for no limit-cycle to exist no eigenvalue of $\tilde{N}(A,\omega)G(j\omega)$ can equal $(-1, j0)$. Consider the 2×2 system shown in Figure 2, the set of equation governing this model is given by equation (9).

$$\begin{aligned}(1 + n_{11}g_{11})A_1 + (n_{12}g_{12})A_2e^{j\varphi} &= 0\\ (n_{21}g_{21})A_1 + (1 + n_{22}g_{22})A_2e^{j\varphi} &= 0\end{aligned} \tag{9}$$

The solution of this equation is sought for specific values of A_1, A_2, ω and φ, where A_1 and A_2 are amplitudes of limit cycles in loop 1 and 2 respectively, ω is frequency of oscillation for both loops and φ is phase-shift between the loops. In this case there are two equations with four unknown variables. Equation (9) can be solved by searching the space of A_1, A_2, ω and φ. As we mentioned above, solving the harmonic balance equation is not trivial; for higher order systems the analytical solution is very complex.

In this paper the Multiobjective Genetic Algorithm (MOGA) is used to search over the existence of any possible limit cycle operation in nonlinear systems and subsequently over controller structures as well as over the controller parameters.

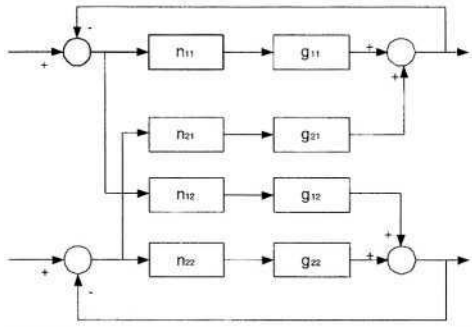


Fig. 2. A two-inputs two-outputs nonlinear system

4 Multiobjective Evolutionary Algorithms

The Genetic Algorithms (GAs) are the stochastic global search method that mimic the metaphor of natural biological evolution. These algorithms maintain and manipulate a population of solutions and implement the principle of survival of the fittest in their search to produce better and better approximations to a solution. This provides an implicit as well as explicit parallelism that allows for the exploitation of several promising areas of the solution space at the same time. The implicit parallelism is due to the schema theory developed by Holland, while the explicit parallelism arises from the manipulation of a population of points [8]. The implementation of GA involves some preparatory stages. Having decoded the chromosome representation into the decision variable domain, it is possible to assess the performance, or fitness, of individual members of a population. This is done through an objective function that characterizes an individual's performance in the problem domain. During the reproduction phase, each individual is assigned a fitness value derived from its raw performance measure given by objective function. Once the individuals have been assigned a fitness value, they can be chosen from population, with a probability according to their relative fitness, and recombined to produce the next generation. Genetic operators manipulate the genes. The recombination operator is used to exchange genetic information between pairs of individuals. The crossover operation is applied with a probability p_x when the pairs are chosen for breeding. Mutation causes the individual genetic representation to be changed according to some probabilistic rule. Mutation is generally considered to be a background operator that ensures that the probability of searching a particular subspace of the problem space is never zero. This has the effect of tending to inhibit the possibility of converging to a local optimum.

Multiobjective Evolutionary Algorithms (MOEA) are based on multi-objective Genetic Algorithms (MOGA). The MOEA begins with a population of possible solutions, called strings. Each string is fed into a model as the candidate solution, in this case these strings are the parameters of the configured controller model.

This model is usually a computer program representation of the solution to the problem. Multi-objective simply means that there is more than one objective involved [9]. For each string, each objective represents a separate cost. The manner in which a string is deemed superior or inferior to other strings is carried out by a selection mechanism. The selected strings undergo the evolutionary process where the traits of the selected strings (which may or may not be good) are selected and combined to form new strings for the next generation. In theory, with each generation, the strings in the population should return better and better cost functions by obtaining strings nearer to the optimal solutions. In practice, often there are limits to the values of cost functions that can be achieved. This depends on the objective functions and the constraints imposed on the model parameters.

5 Simulation Results

The following simulation results illustrate the capabilities of proposed MOGA for limit cycle prediction. In these simulations we choose two 2×2 nonlinear systems. In both systems the numerical solution of equation (9) is formulated as two objective problems. One is absolute value of the first equation and the second objective is simply the absolute of the second equation in (9), as shown in equation (10). The MOGA searches in the space of A_1, A_2, ω and φ to find a set of such parameters that minimize both objectives. Due to the inherent approximation in using SIDF and also the nature of multi-objective formulation, it may not be possible to reach the exact minimum ,which is zero, as required by the equation set (10). Therefore MOGA may converge to a set of Parteo Optimal solution, so in the two nonlinear systems four of the best obtained results were selected as the final results.

$$\begin{aligned} Objectiv_1 &= |(1 + n_{11}g_{11})A_1 + (n_{12}g_{12}A_2)e^{j\varphi}| \\ Objectiv_2 &= |(n_{21}g_{21})A_1 + (1 + n_{22}g_{22})A_2e^{j\varphi}| \end{aligned} \tag{10}$$

5.1 Simulation Results for the First System

The block diagram of the first system has shown in figure 3. As can be seen, this diagram has two linear and nonlinear parts. The elements of the nonlinear part are similar and consist of four ideal relays but the elements of the linear part consists of four linear transfer functions. Parameters setting of MOGA for this system has given in Table 1. The obtained results for prediction of limit cycle

Table 1. MOGA parameters for the first nonlinear system

No. of generation	Population size	Mutation	Crossover
74	36	0.01	0.94

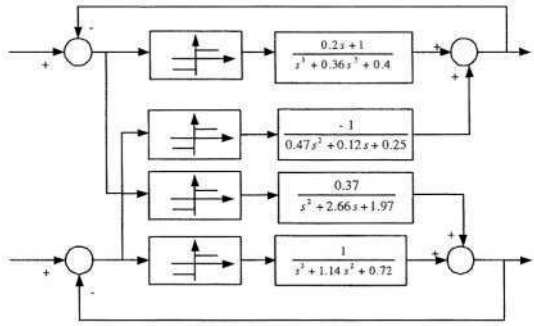


Fig. 3. The block diagram of the first system

of this system has presented in Table 2. With inspection of table 2, we can see that the obtained results of proposed approach is too close to analytical method, which describes the salification of our method.

5.2 Simulation Results for the Second System

The block diagram of the second nonlinear system has shown in figure 4. As can be seen, its configuration is similar to the first system, but the nonlinear matrix ,in additional of ideal relay, consists of different nonlinear behavior such as saturation and dead zone. The parameters setting of MOGA for this system

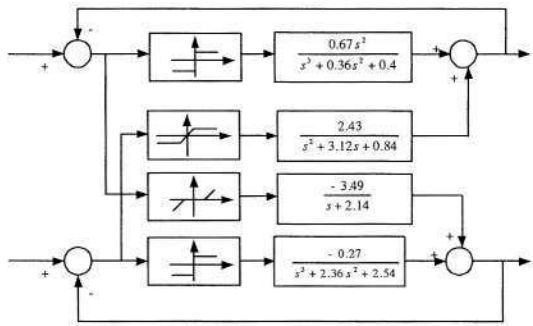


Fig. 4. The block diagram of the second system

are those parameters given in table 1. The obtained results for prediction of limit cycle of this system has presented in Table 3. With inspection of table 3, we can see that the obtained results of proposed approach is too close to analytical method. All programs of the MOGA method have been written in MATLAB/SIMULINK software and c++ language. These programs have been

Table 2. Obtained results of limit cycle prediction with MOGA for the first system

Best obtained results	ω	A_1	A_2	φ	$Objective_1$	$Objective_2$
1	0.37654	1.45627	0.86423	2.35647	0.01162	0.02317
2	0.37622	1.45638	0.86475	2.35609	0.01768	0.02406
3	0.37697	1.45704	0.86429	2.35698	0.01967	0.02385
4	0.37604	1.45677	0.86448	2.35637	0.01145	0.01249
Analytical method	0.37596	1.45668	0.86437	2.35643	–	–

Table 3. Obtained results of limit cycle prediction with MOGA for the second system

Best obtained results	ω	A_1	A_2	φ	$Objective_1$	$Objective_2$
1	0.64725	0.87654	1.28343	3.48752	0.01422	0.01963
2	0.64711	0.87637	1.28387	3.48727	0.01271	0.01537
3	0.64787	0.87672	1.28376	3.48794	0.03624	0.02415
4	0.64762	0.87616	1.28307	3.48765	0.03245	0.02714
Analytical method	0.64716	0.87652	1.28367	3.48713	–	–

executed on a Pentium IV personal computer. On this computer, the response time of the proposed method for all testing cases, was less than 83 second, which makes feasible the application of the proposed approach for limit cycle prediction in reasonable and acceptable computation time.

6 Conclusion

In this paper an intelligent method of predicting limit cycle amplitude, frequency, and phase for nonlinear multivariable systems was presented. this method was based on multiobjective genetic algorithm, which was capable of predicting specified modes of theoretical limit cycle operation. An advantage of this method ,as we showed, was that MOGA can be directed to search for all possible solutions including sub-harmonic components that are ignored in the derivation of the SIDF. Furthermore the proposed method was capable of quantifying the magnitude, frequency and the phase of the limit cycles as well as the loop interaction effects in the frequency domain which proves useful in any subsequent controller design. The effectiveness of the proposed method was demonstrated trough examples. Obtained results showed that MOGA can be used for predicting of limit cycle in MIMO systems with high nonlinearity in a reasonable and acceptable computation time.

References

1. S.D Katebi and M.R Katebi: Control Design for Multivariable Multi-valued nonlinear systems. Journal of System Analysis Modeling and Simulation (SAMS), vol. 15,13–37, 1994.

2. A. Gelb and W. E. van der Velde, Multiple Input Describing Functions and Nonlinear Systems Design. McGraw-Hill, New York, 1968.
3. D. P. Atherton, Nonlinear Control Engineering, Describing Function Analysis and Design. Van Nostrand Reinhold, London, 1975.
4. J.-J. E. Slotine and W. Li, Applied Nonlinear Control. Prentice-Hall, New Jersey, 1991.
5. K. C. Patra, Y. P. Singh: Graphical method of predication of limit cycle for multivariable nonlinear Systems. IEE Proc. Control Theory and Appl., Vol.143 No. 5, 1996.
6. E. Kim, H. Lee, M. Park: Limit-cycle prediction of a fuzzy control system based on describing function method, IEEE Transactions on Fuzzy Systems, Vol. 8,Issue 1, 11–22, 2000.
7. Heyns, L. J., Kruger, J. J., Describing function-based analysis of a nonlinear hydraulic transmission line, IEEE Transactions on Control Systems Technology, Vol. 2, Issue 1, 31–35, 1994.
8. P. J. Flemming and R. C. purshouse: Genetic Algorithms in Control Engineering, Research report NO. 789. Dept. of Automatic and System Eng, University of Sheffield, UK, 2001.
9. Rashidi, F., Rashidi, M., Design of Multi-Objective Evolutionary Technique Based Intelligent Controller for Multivariable Nonlinear Systems, to apear in Lecture notes in Artificial Intelligence, 2004.

Evolving Reusable Neural Modules

Joseph Reisinger, Kenneth O. Stanley, and Risto Miikkulainen

Department of Computer Sciences
The University of Texas at Austin
1 University Station C0500
Austin, TX 78712-1188
{joeraii,kstanley,risto}@cs.utexas.edu
http://www.cs.utexas.edu/~{joeraii,kstanley,risto}

Abstract. Topology and Weight Evolving Artificial Neural Networks (TWEANNs) have been shown to be powerful in nonlinear optimization tasks such as double pole-balancing. However, if the input, output, or network structures are high dimensional, the search space may be too large to search efficiently. If the symmetries inherent in many large domains were correctly identified and used to break the problem down into simpler sub-problems (to be solved in parallel), evolution could proceed more efficiently, spending less time solving the same sub-problems more than once. In this paper, a coevolutionary modular neuroevolution method, Modular NeuroEvolution of Augmenting Topologies (Modular NEAT), is developed that automatically performs this decomposition during evolution. By reusing neural substructures in a scalable board game domain, modular solution topologies arise, making evolutionary search more efficient.

1 Introduction

Topology and Weight Evolving Artificial Neural Network (TWEANN) methods become increasingly intractable as the network complexity of the individual solutions (and thus the complexity of the space of solutions) being searched increases. This complexity is known as *dimensionality,* and is influenced primarily by two factors: the number of inputs and outputs a solution network has, and the topological complexity (e.g. amount of hidden neurons) of that network. Many board games are difficult to learn because they have high dimensional input and output spaces (i.e. the board size is large). Even powerful Neuroevolution (NE) methods such as NeuroEvolution of Augmenting Topologies (NEAT) are challenged in these domains because the search space is too large [1]. In this paper, an NE method is proposed to increase the efficiency of high-dimensional search.

The dimensionality problem exists in biological evolution, however nature is able to overcome it and produce exceedingly complex phenotypes through modularization of the genotype-phenotype map [2]. Also, developmental modularity allows for reuse of complex phenotypic structures without correspondingly complex mutations in the genotype. Modularization can be thought of as a decomposition operation that reconstitutes the search space in a way that is more amenable to natural evolution [3].

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 69–81, 2004.

The key to tackling the dimensionality problem in artificial evolution, then, is to find ways to perform a similar modular decomposition of the genotypic search space. One way to accomplish this decomposition in NE is to evolve reusable neural substructures (modules) and combine them together.

With the added level of abstraction that modules provide, more complex solutions are searched with fewer operations than standard NE. The trade-off is that the search becomes coarser. For example, mutations represent combinations of modular building blocks instead of weights, making it difficult or even impossible to find some solutions.

In this paper, a coevolutionary NE method Modular NeuroEvolution of Augmenting Topologies (Modular NEAT) is proposed that allows reusing fundamental neural substructures, which encourages modularization. Modular NEAT is tested on an artificial board game domain with scalable input/output complexity, and is shown to be 1) several times more efficient than the standard NEAT method and also 2) generate modular solution networks. This approach allows for the solutions to a class of difficult problems to be approximated more easily than with non-modular NE techniques.

2 Modularity in Natural Evolutionary Systems

Modularity can be intuitively defined as "the integration of functionally related structures and the dissociation of unrelated structures" [4]. In other words the structures are some combination of reusable, interchangeable, and functionally separate. A technical discussion of modularity is given in [5] and also [6]. Independent functions are coded independently using modularity, and as such each function can be optimized separately with little interference with other already optimized functions. Thus modularity not only structures the genotype-phenotype map for mutational stability, but also increases evolvability.

Natural modularity makes a significant class of difficult problems tractable by rearranging the structure of the search space [3]. This has previously been accomplished in NE through gene-duplication [7,8,9], and gene reuse[10, 11]. In a gene-duplication scheme, two copies of the gene are made and the resulting two functions can specialize independently. In contrast, modularity can also arise from gene *reuse,* which has not yet been fully explored with regards to NE. In a gene reuse scheme, genes that perform one function are reused wherever that function is needed, and are functionally separate from other genes. Gene reuse can be thought of as a process for generalizing gene function, whereas gene duplication can be thought of as a process for specializing it.

In this paper, automatic modular decomposition is implemented in a cooperative coevolution system with a population of reusable genes (i.e. modules) and a population of blueprints specifying combinations of modules. In such a system,

- Evolution and modularization take place at the same time as modules are discovered.
- The space of a complex fitness function is restructured [12].

– Genes are reused explicitly without the need for a more complex genotype-phenotype map (such as a system with development).

Thus an NE system that takes advantage of modularization can be more efficient than one that does not. The next section presents Modular NEAT, an implementation for modular decomposition with coevolution.

3 Modular NEAT Method

A method that allows for concurrent evolution and modularization of network topologies is developed and tested in this paper. The method is based on NEAT [1], which provides a simple but efficient way to automatically discover the correct level of complexification.

3.1 Standard NEAT Implementation

The standard NEAT implementation has been shown to be a highly effective NE method in several domains [13]. It addresses three problems commonly found in TWEANN systems: 1) how to crossover topologically disparate chromosomes, 2) how to protect new topological innovation, and 3) how to keep topologies as simple as possible throughout evolution [1]. This is accomplished through historical markings, speciation, and incremental complexification.

First, each genome in NEAT includes a list of *connection genes,* each of which refers to two *node genes* being connected. In order to perform crossover, the system must be able to tell which genes match up between *any* two individuals in the population. For this reason, NEAT keeps track of the historical origin of every gene. Two genes that have the same historical origin represent the same structure (although possibly with different weights), since they were both derived from the same ancestral gene from some point in the past. Tracking the historical origins requires very little computation. Whenever a new gene appears (through structural mutation), an *innovation number* is incremented and assigned to that gene. The innovation numbers thus represent a chronology of every gene in the system, and allows crossover of diverse networks without extensive topological analysis. With historical markings the problem of having to match different topologies [14] is avoided.

Second, NEAT networks are speciated so that individuals compete primarily within their own niche. This way, topological innovations are given time to optimize their structure before they have to compete with the entire population. Also, networks share the fitness of their species [15], to prevent one species from taking over the entire population.

Third, unlike other TWEANN methods [7,16], NEAT networks are built from a minimal configuration and complexified incrementally to ensure that solutions of minimal complexity are searched first. This procedure has two advantages: first, it minimizes topology bloat, and second, it improves the efficiency of evolution by complexifying the search space only as needed. On the other hand,

minimal starting is problematic in domains with high input/output dimensionality because "minimal" starting actually represents a huge space of weights. The Modular NEAT method allows for input and output connections to be added during evolution, allowing for even simpler initial topologies.

For more details about NEAT, see Stanley and Miikkulainen [1].

3.2 Modular NEAT Overview

To allow for concurrent evolution and modularization, Modular NEAT reuses arbitrarily complex neural substructures, from a single connection to large subnetworks, to form complete neural networks. This method is similar in vein to the gene-duplication scheme of NE proposed by Calabretta et. al. [9,17], however in Modular NEAT, genes are reused in different spatial locations in the network instead of duplicated. This encourages generalization of function, instead of specialization.

Modular NEAT encodes each reusable substructure (module) as a functionally independent NEAT network from a set of input neurons to a set of output neurons. Since the building blocks are complete neural networks, they can be evolved using standard NEAT.

In addition to the population of NEAT modules, Modular NEAT makes use of blueprints that specify combinations of those modules. Both populations are evolved together symbiotically, as in the SANE method [18]. Good modules must evolve to work well with other modules, and across different spatial locations in the network.

The main difference is that in SANE each neuron can only be used in one place in the resulting network. If a neuron were reused, it would perform the exact same function as before, effectively just reinforcing that particular function. In Modular NEAT, modules can be bound to *different subsets* of input and output neurons without overlap, or with little overlap, so reuse of modules actually has a constructive effect. For example, if the optimal solution network has bilateral symmetry (i.e. the left and right halves are the same network), one module half the size of the target network can be bound on both halves to solve the problem.

3.3 Module Population

Modules are composed of a set of links between input, output and hidden neurons that are specific to the module. These *abstract* neurons are later assigned to real neurons in the solution network, through a process called binding. Each link has a globally unique identifier (for historical marking), a weight, and a start and end neuron. The module population is divided into species based on how similar the individual network topologies are (figure 1) [1]. Topological similarity is determined according to the equation

$$d = k_1 w + k_2 t,$$

where w is the sum of the absolute value of the weight differences for all connections, and t is the number of differing topological innovations (i.e. the number of

links not shared by both individuals). The constants k_1 and k_2 can be changed in order to balance the net effects of the two difference measures.

The module population is speciated for two reasons in addition to those proposed by NEAT: first, it helps maintain specializations in module function, and second, it provides a simple way to determine what other modules could serve as replacements when one is needed in a blueprint (i.e. when another member of the species dies and the blueprints need to be updated).

Module crossover is implemented similarly to crossover in NEAT. The matching historical markings are lined up and crossed over randomly (genes that do not line up are inherited from the most fit parent). Crossover using historical markings ensures that only complete mutations are inherited (i.e. no dangling neurons or unconnected edges will be present in the children). The majority of module crossovers take place with modules of the same species, however it is possible for higher fitness modules of some species to be paired with other modules outside the species.

There are three types of module mutations, two of which increase the dimensionality of the network:

- Perturb an individual weight (Gaussian weight mutation).
- Split a link (adding an additional input, output, or hidden neuron).
- Add a new link between two existing neurons.

Modules begin as a single connection between an abstract input and output neuron, so before new links can be added, the module must mutate to include new input or output neurons. Note that Modular NEAT diverges from NEAT in that new input and output neurons can be added to a module, up to the maximum specified by the solution network topology. This allows modules to "grow" new inputs or outputs as needed.

3.4 Blueprint Population

Blueprints are composed of a fixed-size list of modules paired with mappings from the abstract input/output neurons to real input/output neurons from the final neural network. These mappings are referred to as bindings. For example, module A may specify a link between neuron 0 and 1, however its binding in a certain blueprint may be 0:5, 1:6, 2:7, etc. Thus all references to abstract neuron 0 in the module under this binding actually refer to neuron 5 (in the actual network), etc. These module/binding pairings also have historical markings, in order to track the spread of mutations for data collection (figure 1).

Blueprints are recombined using standard one-point crossover. Cuts in the genetic material can only be made between modules, so blueprint crossover does not contribute to module mutation. The child blueprints inherit the order of the modules and their bindings from the parents.

There are three types of blueprint mutation: toggling a gene (module+binding) expression on or off, changing an entire module and changing part of the bindings on a module. When mutating one module to another, as much of

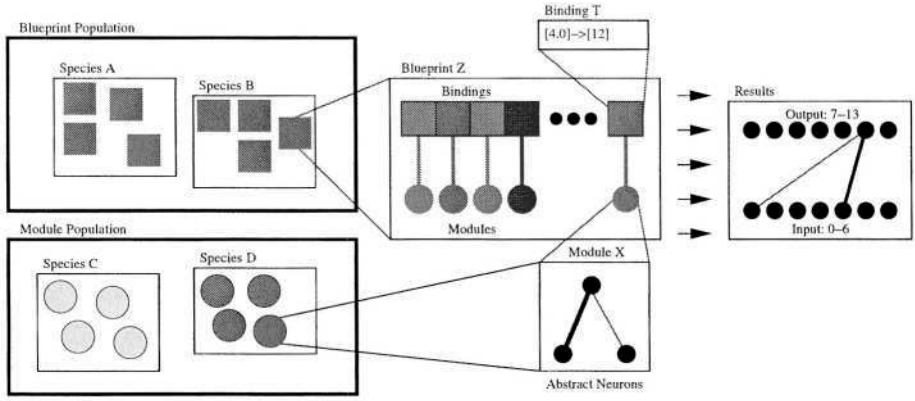


Fig. 1. Graphical indication of how a module is bound into the solution network. An example blueprint from the blueprint population is shown with its binding and module pairs list. The result of applying a binding to a single module is demonstrated in the resulting network.

the original binding is left intact as possible. If the new module has fewer inputs or outputs, the current binding is cut to fit, likewise if the new module is too large, then new neurons are added randomly to the current binding.

3.5 Evolving Modules and Blueprints

The modular NEAT algorithm consists of the module and blueprint populations undergoing cooperative coevolution. This is implemented in two sequential phases, a module phase and a blueprint phase. Both phases include separate selection, crossover, and mutation steps, after which fitness calculation takes place and the weakest members of the two populations are replaced. This process consists of the following steps:

- Initialize Modules – create an initial population of modules with a static size. These modules begin with minimum complexity.
- Initialize Blueprints – create an initial population of blueprints from the population of modules. Each blueprint has a static size n, i.e. n modules are randomly bound to it. For each blueprint, calculate an initial fitness.
- Until a certain terminating condition:
  - Select modules and perform crossover (binary tournament selection, NEAT multi-point crossover [1]).
  - Select blueprints and perform crossover (tournament selection and one-point crossover).
  - Mutate modules by randomly perturbing weights, adding new neurons, and adding new connections. Modules that change input/output size must be rebound at the blueprint level.
  - Mutate blueprints by changing bindings, and change bound modules.

- Calculate fitness for modules and blueprints.
- Replace modules.
- Replace blueprints.

– Repeat.

During the fitness evaluation step, the fitness in the population is calculated, and the scores are propagated to each of the modules. To help keep modules diverse, the fitness scores a module receives from each blueprint are summed and divided by the number of times that module is used (expressed) in the blueprint population, making sure that modules that are expressed less often have a fair chance to compete against well-established modules.

Also, the blueprint population is allowed several generations to optimize after a change is made in the module population. Kvasnička et. al. presented a similar method in which the genotype and genotype-phenotype map are optimized on different time scales [19]. Applied to Modular NEAT, Kvasnička's procedure allows for more data to be gathered on module fitness resulting in better module evaluations, and also allows the blueprints to better incorporate old modules, before new ones are created. Experimentally, a coevolutionary ratio of 5 blueprint generations to a single module generation is effective.

In the next section, a scalable board game domain is proposed that highlights the types of domains under which Modular NEAT is effective.

4 Experimental Setup

Modular NEAT was tested in an aritificial board game domain where the size of the board can be varied, i.e. the input/output dimensionality. The game itself is relatively simple, so as to focus on modular reuse.

The board is a square with $n \times n$ positions initially with $n/2$ black stones placed about randomly on the left half of the board, and $n/2$ white stones on the right half. The neural network then proceeds to play n stones the board. Stones played on the left half appear white, and stones played on the right half appear black. Play occurs iteratively, with each subsequent board state fed as a three-state (-1, 0, and 1) vector to the neural network, and location of the highest output value taken as the network's next move. To score a game, the neural network is given a point for each of the stones that it places next to one of the initial stones (i.e. "checks" it). Scoring occurs after a fixed number of moves, 5 stones for the 5×5 case, and 10 stones for the 7×7 case.

The domain requires the neural network to learn to differentiate between empty space and the types of stones on the board, and also whether or not it has already played next to a certain stone (otherwise it may end up wasting stones trying to check the same piece more than once). By reversing the color for the two halves of the board (and thus negating the input), modules that are usable on one side of the board are rendered useless on the other half. Having two functionally different regions encourages a modularization that minimizes interactions between the two halves.

A 10 × 10 board requires a neural network with 100 inputs and 100 outputs, but is still simple enough to be solved without a correspondingly complex topology. Thus this domain tests Modular NEAT's ability to deal with large input/output spaces, complimenting NEAT's ability to deal with topologically complex solutions.

For the experiments described in this paper, the following genetic algorithm parameters were used for the module population: 100 modules, 10% outside species mating, 1% add link, .8% add hidden node, 3% add abstract input/output, and 50% Gaussian weight mutation. For the blueprint population the parameters: 500 blueprints, 10% outside species mating, 1% gene expression mutation, 5% change module mutation, and 10% perturb binding mutation were used. Also, each network was evaluated over 50 random board configurations, and each trial was run for 3000 generations, to ensure that the highest fitness possible is reached. After 3000 generations, little useful evolution takes place.

5 Results

Several experiments were conducted with Modular NEAT. First, it was compared to NEAT in terms of solution quality and evolutionary efficiency, second, the degree to which modular phenotypes emerged was measured, and third, how useful the evolved modularizations were was tested.

5.1 Comparison to NEAT

Modular NEAT was compared to the standard NEAT implementation of [1]. With 500-member populations over 3000 generations on a 5 × 5 instance of the game, standard NEAT was able to evolve solutions that played correctly on average 96.66% of the time over 30 trials, and modular NEAT 98.9% (difference is statistically significant, $p < 10^{-23}$). Also, the efficiency of the two algorithms in evolving a 96% accurate solution was measured. It took NEAT on average 1753.87 generations, while Modular NEAT took only 420.73 generations (statistically significant, $p < 10^{-23}$), an effective four-fold increase in efficiency (table 1).

Table 1. Final solution quality in 3000 generations, and number of generations required on average to reach the goal fitness for both NEAT and Modular NEAT. Modular NEAT not only evolves solutions of higher quality, but also evolves them several times faster. All differences are statistically significant ($p < 0.05$), averaged over 30 trials.

	Standard NEAT		Modular NEAT	
	Quality	Efficiency	Quality	Efficiency
5x5	96.66	1753.87	98.9	420.73
7x7	87.5	1981.33	92.5	327.2

In the 7 × 7 case, the solution quality was on average 87.5% for NEAT and 92.5% for Modular NEAT. In evolving to a fitness goal of 86.6%, NEAT took on average 1981.33 generations, while Modular NEAT took only 327.2 generations, an effective six-fold increase in efficiency (table 1). All differences in means were statistically significant ($p < 0.05$).

These results indicate that Modular NEAT is able to obtain better solution networks than standard NEAT, and find good networks faster. Let us examine how modules contributed to this result next.

5.2 Modularization

Modular NEAT is able to identify and separate the two functions by assigning different module species to either one half or the other. The degree to which modular phenotypes evolved can be measured by counting how often a module takes an input from one half and maps it to an output of the second half, or vice versa. This kind of crosslink does not improve fitness, and should be suppressed if possible.

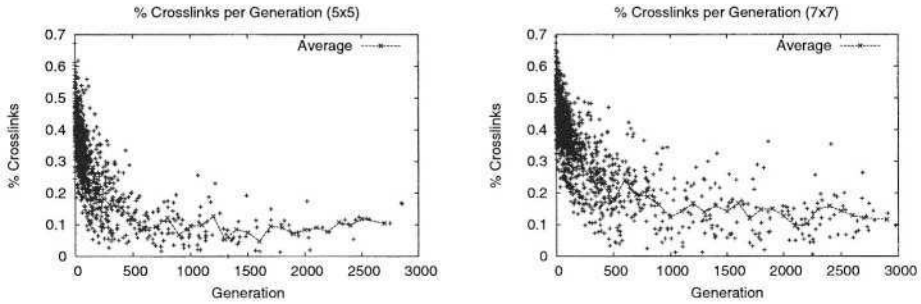


Fig. 2. Crosslink percentage per generation in the 5×5 and 7×7 board game domains. Unlike NEAT, Modular NEAT reduces crosslinks over time, focusing evolution on useful connections.

The expected percentage of crosslinks out of the total links, if no modularization occurs, is 50%. In NEAT, since no modularization occurs, the percentage of crosslinks in the champion genomes indeed stays around 50% during evolution. In Modular NEAT, however, this percentage decreases over time. In the 5 × 5 case, the percentage of crosslinks decreases to below 10% on average, and in the 7 × 7 case it decreases to slightly above 10% (figure 2).

This result indicates that the substructures can be arranged in a way that separates them by function, supporting the hypothesis that reusing substructures leads to a modular phenotype. By reusing certain modules on only one half or the other, evolution of both sides can occur simultaneously without interference.

5.3 Module Use

Because modules are reused in many different combinations across many different spatial locations, typical module evaluation is often not consistent, and hence the modules themselves remain relatively generic in function. Two tests are conducted to measure the usefulness of the evolved modules.

Topological Decomposition. How well does Modular NEAT decompose the problem space? One test is to freeze the module bindings and module topologies and randomize the module weights. Re-evolving only the weights from this base gives a good idea as to how well the module topologies and bindings decompose the problem, when compared to a randomly evolved modularization. A random modularization can be created by running the algorithm for a fixed period of time, evaluating blueprints normally and then assigning all modules the same fitness. With this method, the average module complexity can be kept about the same in both the evolved population and the random population.

The evolved population solved the problem on average 1.37 times faster than the random population with 160 generations of setup time, 1.57 times faster with 320 generations of setup time, and 1.65 times faster with 480 generations of setup time (table 2). In all three cases the differences in means were statistically significant ($p < 0.05$). Thus the decomposition encoded in the evolved module topologies and the module bindings is useful even if all the module weights are randomized.

Table 2. Averages of efficiencies (out of 60 trials) for the test and random populations. Setup Generation refers to the number of generations used in setting up the modularization before the module weights are randomized. Speedup is the effective gain in efficiency from the evolved module topologies and bindings combined. The evolved population clearly outperforms the random one.

Setup Generations	160	320	480
Evolved	291.933	333.9	237.898
Random	400.111	524.792	391.8
Speedup	1.371	1.572	1.647

Module Population Decomposition. The results in the previous section indicate that bindings and module topologies together enhance evolution. However, it does not answer how useful the module population itself is, without the set of bindings. The set of modules would have to be generic enough to guide the blueprint evolution quickly to the correct decomposition, and still be specialized enough to represent a better solution to the problem in this configuration, in order to be useful.

One test is to perform evolution for a fixed number of generations, then destroy the blueprint population, and re-evolve from a new random blueprint

population that uses the same set of modules. The re-evolved population should be able to reach the same fitness level than the initial population, but faster.

Two experiments were conducted, one with evolution taking place for 300 generations, and one for 500 generations (table 3). On the 300 generation version, the best solution took on average 255.977 generations to reach, and the re-evolved solution took 236.167 generations, representing an effective speedup of 1.08 times. These means differ by about twenty generations, however the difference is not statistically significant ($p = 0.152$). With 500 generations, the means differed by approximately seventy generations, representing a speedup of 1.19 times ($p < 0.003$).

Table 3. Modules and blueprints are evolved for a fixed number of generations and then the bindings are randomized and evolution proceeds until the highest fitness from the previous run is reached again. The number of generations on average to reach the highest fitness both initially, and after randomization is recorded.

Setup Generations	300	500
Before	255.967	451.433
After	236.167	380.3
Speedup	1.084	1.187

These results indicate that evolved modules are more useful than random ones. However they are not useful enough to warrant eradicating the set of bindings: separating modules from their expression does not significantly speed up evolution.

Taken together, the conclusions in these two experiments suggest that binding information could be specified on the modules themselves, instead of on separate chromosomes. Such an encoding is a promising direction for future research, as will be discussed in the next section.

6 Discussion and Future Work

Modular NEAT is powerful because it can restructure the search space and yield automatic decompositions of the problem without solving parallel sub-tasks more than once. That is, a module that performs a certain function can be reused wherever that function is needed. Although searching through combinations of highly complex modules may prevent the algorithm from finding the optimal solution, the chances are high that it finds a *good* solution. In domains currently intractable due to their dimensionality, finding such a decomposition may be an essential first step in actually finding a solution.

As the experiments in section 5.3 indicate, separating generic modules from their bindings yields a module population only slightly more useful than the initial random one. Even with highly generic modules, modular decompositions

are only useful when the modules and their expression information are coupled. This result makes intuitive sense. If the information is separated, generic modules will be selected more often since modules do not have genetic control over their expressions. What is needed is a system that combines both module topology and expression information into the same chromosome.

In the current version of Modular NEAT, explicit module bindings increase the size of the search space proportional to the number of reuses. However, as long as the overhead of module binding can be kept low and the number of reuses kept high, this overhead is negligible. Compared to systems with developmental elements (i.e. genotype-phenotype maps that are not one-to-one) [7,20,21], explicit module bindings are not efficient. Natural and artificial developmental systems rely on relative targeting [2], which could be used in Modular NEAT to reduce the binding overhead for each module (e.g. modules could be tessellated regularly across the binding space).

Modular NEAT is most suited to domains where there are spatial regularities, such as Go and Othello. Like the test domain, they have rotational symmetry. Another large, self-similar domain is low-level vision processing such as filtering. Although it may be orientation specific, the input is still scalable. Since modules can add new input connections, the controllers can start out using a small subset of the inputs and add more if needed.

7 Conclusions

Modular NEAT has been shown to 1) make evolution significantly more efficient than standard NEAT in a scalable board game domain, and 2) implicitly generate modular solutions. Since there is no explicit selection for modularity built into Modular NEAT, phenotypic modularity must arise from the reuse of fundamental substructures. In nature, this is similar to the modularity that arises from reusing genes. By simultaneous modularization and evolution through cooperative coevolution, Modular NEAT is a principled approach to a significant class of difficult problems, i.e. those with high input/output dimensionality and self-similarity.

References

1. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary Computation **10** (2002) 99–127
2. Stanley, K.O., Miikkulainen, R.: A taxonomy for artificial embryogeny. Artificial Life **9** (2003) 93–130
3. Hart, W.E., Belew, R.K.: The role of development in genetic algorithms. Technical Report CS94-394, University of California, San Diego, CA (1994)
4. Bongard, J.C.: Evolving modular genetic regulatory networks. In: Proc. of CEC 2002, IEEE Press (2002) 1872–1877
5. Wagner, G., Altenberg, L.: Complex adaptations and the evolution of evolvability. Evolution **50** (1996) 967–976

6. Futuyma, D.J.: Evolutionary biology. Sinauer, (Sunderland, MA)
7. Gruau, F., Whitley, D., Pyeatt, L.: A comparison between cellular encoding and direct encoding for genetic neural networks. In Koza, J.R., Goldberg, D.E., Fogel, D.B., Riolo, R.L., eds.: Genetic Programming 1996: Proceedings of the First Annual Conference, Stanford University, CA, USA, MIT Press (1996) 81–89
8. Nolfi, S.: Using emergent modularity to develop control systems for mobile robots. Adaptive Behavior **5** (1997) 343–363
9. Calabretta, R., Nolfi, S., Parisi, D., Wagner, G.P.: A case study of the evolution of modularity: Towards a bridge between evolutionary biology, artificial life, neuro- and cognitive science. In Adami, C., Belew, R., Kitano, H., Taylor, C., eds.: Proceedings of Artificial Life VI. MIT Press (1998)
10. Watson, R.A., Pollack, J.B.: Symbiotic composition and evolvability. In Kelemen, J., Sosik, P., eds.: Proc. of Advances in Artificial Life, 6th European Conference, (ECAL 2001), Springer (2001) 480–490
11. de Jong, E.D.: Representation development from pareto-coevolution. In: Proc. of GECCO 2003, Morgan-Kaufman (2003)
12. Bentley, P.J., Kumar, S.: Three ways to grow designs: A comparison of embryogenies for an evolutionary design problem. In: Proc. of GECCO'99. (1999) 35–43
13. Stanley, K.O., Miikkulainen, R.: Competitive coevolution through evolutionary complexification. Journal of Artificial Intelligence Research **21** (2004)
14. Radcliffe, N.J.: Genetic set recombination and its application to neural network topology optimization. Neural computing and applications **1** (1993) 67–90
15. Goldberg, D.E., Richardson, J.: (Genetic algorithms with sharing for multimodal function optimization) 148–154
16. Yao, X.: Evolving artificial neural networks. Proc. IEEE **87** (1999) 1423–1447
17. Calabretta, R., Nolfi, S., Parisi, D., Wagner, G.P.: Duplication of modules facilitates the evolution of functional specialization. Artificial Life **6** (2000) 69–84
18. Moriarty, D.E.: Symbiotic Evolution of Neural Networks in Sequential Decision Tasks. PhD thesis, The University of Texas at Austin (1997)
19. Kvasnička, V., Pospíchal, J.: Emergence of modularity in genotype-phenotype mappings. Artificial Life **8** (2002)
20. Gruau, F.: Automatic definition of modular neural networks. Adaptive Behavior **3** (1994) 151–184
21. Belew, R.K., Kammeyer, T.E.: Evolving aesthetic sorting networks using developmental grammars. In Forrest, S., ed.: Proc. of the 5th International Conference on Genetic Algorithms, San Francisco, CA, Morgan Kaufman (1993)

How Are We Doing? Predicting Evolutionary Algorithm Performance

Mark A. Renslow[1], Brenda Hinkemeyer[2], and Bryant A. Julstrom[3]

Department of Computer Science
St. Cloud State University, St. Cloud, MN 56301 USA
[1]markminn@mac.com
[2]brenda@nikosha.net
[3]julstrom@eeyore.stcloudstate.edu

Abstract. Given an evolutionary algorithm for a problem and an instance of the problem, the results of several trials of the EA on the instance constitute a sample from the distribution of all possible results of the EA on the instance. From this sample, we can estimate, non-parametrically or parametrically, the probability that another run of the EA, independent of the initial ones, will identify a better solution to the instance than any seen in the initial trials. We derive such probability estimates and test the derivations using a genetic algorithm for the traveling salesman problem. We find that while the analysis holds promise, it should probably not depend on the assumption that the distribution of an EA's results is normal.

1 Introduction

Evolutionary algorithms are applied to instances of computationally difficult optimization problems for which optimum solutions are not known. Because EAs are probabilistic, we cannot in general know how good the solution returned by any one run might be, and we often carry out multiple trials of an EA on an instance.

The results of such trials constitute a sample from the distribution of all possible results of the EA on the target instance. We can use this sample to determine whether subsequent runs might or might not be worth doing. Specifically, we can estimate, from an initial set of trials, the probability of identifying a better solution on the next run, or in the next fifty runs. Such information would be useful, for instance, when deciding whether to carry out more trials of a computationally expensive EA.

We consider, then, the following problem. Given an evolutionary algorithm for a problem, an instance of the problem, and the results of an initial set of trials of the EA on the instance, what is the probability that one additional trial, independent of those already completed, will return a solution better than any observed in the initial trials?

A non-parametric analysis identifies a probability that depends only on the number of initial trials. A parametric analysis assumes that the values the EA re-

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 82–89, 2004.

turns conform to a particular distribution—here, normal—and identifies a probability that depends on that assumption and on the mean and standard deviation of the initial results. The two analyses are tested with a simple genetic algorithm on five instances of the well-known traveling salesman problem. We find that neither analysis is as accurate as we might like, and the second is flawed by the assumption of normality, but the method itself holds promise.

The following sections of the paper define the problem precisely, present non-parametric and parametric derivations of the probability that another trial identifies a better solution, and describe the tests of both analyses using a genetic algorithm for the traveling salesman problem.

2 The Problem

Let Q be an optimization problem, like the traveling salesman problem or the 0-1 knapsack problem, and let E be an evolutionary algorithm for Q. The fitness of a genotype in E is the objective function value of the solution the genotype represents. When E is applied to an instance of Q, it reports, on its termination, the single best solution represented in its population.

Without loss of generality, assume that Q seeks to minimize its objective function, as in the TSP, and let Q_o be a particular instance of Q. Assume that Q_o is difficult enough for E that E is unlikely to return an optimum solution to Q_o. Since E is probabilistic, repeated independent trials of it on Q_o will return solutions with a variety of fitnesses.

Let the random variable X be the fitness of the solution E returns at the conclusion of one trial on Q_o. X has some (unknown) distribution. Given n probes into this distribution—that is, given the results of n independent trials of E on Q_o—what can we say about the distribution of X ? In particular, how likely is it that another trial will improve on the best result observed in the initial trials?

Note that the problem just posed is a special case of a more general one: Given a sample from an unknown distribution, what can we say about that distribution and probabilities based on it?

3 A Non-parametric Analysis

Consider a sequence of n probes into the unknown distribution of X; that is, a sequence of n independent runs of the evolutionary algorithm E on the problem instance Q_o. The trials will return values $X_1, X_2, \ldots, X_n$, the fitnesses of the n best solutions to Q_o that E discovers.

Assume that the probability that two trials return identical results is negligible. Then the probability that the second trial will return a solution with smaller evaluation than the first is $P[X_2 < X_1] = 1/2$; the probability that the third trial will return a better solution than the first two is $P[X_3 < X_1, X_2] = 1/3$;

and so on. In general, the probability that the last of k trials will return the solution with the smallest evaluation is

$$P[X_k < X_1, X_2, \ldots, X_{k-1}] = \frac{1}{k}. \tag{1}$$

When n trials have been completed and the values $x_1, x_2, \ldots, x_n$ observed, it is no longer strictly accurate to say that the probability that the next trial will improve on its predecessors is $1/(n+1)$, since this probability now depends on the recorded values and on the distribution of X. However, that distribution is unknown, so we will reckon the probability that the next trial returns a better result than its predecessors as $p = 1/(n+1)$. More generally, if we carry out m additional trials, at least one of them will improve on the first n trials if it is not the case that none do. The probability that one additional trial fails to return a better result is $1 - p = n/(n+1)$, so the probability that none of m additional trials returns a better result is $(1-p)^m = (n/(n+1))^m$, and the probability that at least one of them does improve on the best of the first n trials is then

$$1 - (1-p)^m = 1 - \left(1 - \frac{1}{n+1}\right)^m. \tag{2}$$

For example, if $n = 20$, then the probability that the twenty-first trial returns a solution with smaller evaluation than any of the first twenty is 1/21 = 0.0476. If we carry out 25 additional trials, the probability that at least one of them returns a better solution than any in the first batch is

$$1 - \left(1 - \frac{1}{21}\right)^{25} = 0.705. \tag{3}$$

4 A Parametric Analysis

Again, let $X_1, X_2, \ldots, X_n$ be the fitnesses of the solutions returned by n independent trials of the evolutionary algorithm E on the problem instance Q_o. If we can assume a particular distribution for X, more precise predictions of the evolutionary algorithm's behavior become possible.

For example, assume that X has the normal distribution $N(\mu, \sigma^2)$ with mean μ and variance σ^2. The mean $\overline{X}$ of the n returned values is an unbiased estimator of μ, and their variance $S^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \overline{X})^2$ is an unbiased estimator of σ^2; we approximate $N(\mu, \sigma^2)$ with $N(\overline{x}, s^2)$, where $\overline{x}$ and s^2 are the mean and variance of the trials' results.

The probability that another trial will return a solution with a smaller evaluation than any of the first n trials depends on the smallest evaluation among them, on $\overline{x}$, and on $s = \sqrt{s^2}$, as Figure 1 illustrates. In particular, if $x_{\min} = \min\{x_1, x_2, \ldots, x_n\}$, then

$$p = P[X < x_{\min}] = P\left[Z < \frac{x_{\min} - \overline{x}}{s}\right] = 0.5 - \Phi\left(\frac{\overline{x} - x_{\min}}{s}\right), \tag{4}$$

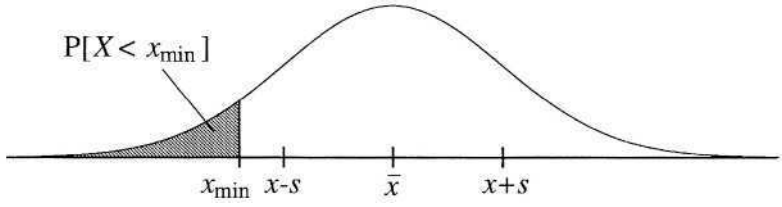


Fig. 1. Given the results of n trials with mean $\overline{x}$, standard deviation s, and smallest value $x_{\min}$, the probability $P[X < x_{\min}]$ that a new trial will return a solution with smaller evaluation than $x_{\min}$, under the assumption that X is normally distributed

where Z is the standard normal random variable with distribution $N(0,1)$ and $\Phi(z)$ is the area under Z's density function above z.

Again, the probability that at least one of m additional trials returns a solution with smaller evaluation than any among the first n is $1 - (1 - p)^m$. In addition, it is possible to estimate the probability that an additional trial will return a solution with evaluation smaller than an arbitrary value x_o:

$$P[X < x_o] = P\left[Z < \frac{x_o - \overline{x}}{s}\right]. \tag{5}$$

For example, if twenty trials of the evolutionary algorithm E on the problem instance Q_o have returned solutions whose evaluations have mean $\overline{x} = 455.3$, standard deviation $s = 18.6$, and minimum $x_{\min} = 426.8$, then the probability p that another trial will return a value less than $x_{\min}$ is

$$\begin{aligned} p = P[X < 426.8] &= P\left[Z < \frac{426.8 - 455.3}{18.6}\right] = P[Z < -1.53] \\ &= 0.5 - \Phi(1.53) = 0.5 - 0.437 = 0.063, \end{aligned} \tag{6}$$

and the probability that at least one of 25 additional trials will improve on $x_{\min}$ is $1 - (1 - 0.063)^{25} = 0.803$.

Moreover, the probability that another trial will identify a solution with evaluation less than, say, 425.0 is

$$\begin{aligned} P[X < 425.0] &= P\left[Z < \frac{425.0 - 455.34}{18.6}\right] = P[Z < -1.63] \\ &= 0.5 - \Phi(1.63) = 0.5 - 0.448 = 0.052. \end{aligned} \tag{7}$$

The distribution of X, the value returned by one trial of E on Q_o, depends not only on E's coding, fitness function, and operators but also on all its features and parameter values; change any of these, and the distribution changes. Also, X's distribution is bounded below by the evaluation of an optimum solution and so is likely to be skewed rather than normal. However, an analysis like that above can be carried out with any other distribution, and a χ^2 statistic can be used to test the hypothesis that X conforms to the chosen distribution.

5 Tests

The analyses of the previous two sections were tested with a straightforward (and not particularly effective) generational genetic algorithm. The GA addressed the well-known traveling salesman problem (TSP) in which, given a collection of cities and the distances between them, we seek a tour that visits each city exactly once and has minimum total distance. The GA represents candidate tours as permutations of the cities. The order of the cities in a permutation indicates the order in which the permutation's tour visits them, and the fitness of a permutation is that tour's length, which we seek to minimize.

The GA initializes its population with random permutations, and it chooses permutations to be parents in k-tournaments. The crossover operator is edge recombination [1], and the mutation operator is subtour inversion [2, pp.219–220]. Each new permutation is generated by exactly one operator, never both. The GA is 1-elitist, preserving the best permutation from the current generation into the next, and it runs through a fixed number of generations.

In these experiments, the GA's population contained 100 permutations. The size of its selection tournaments was $k = 2$, and a tournament's winner always became a parent. The probability that it would apply crossover to generate the next new permutation was 50%, and the probability of mutation was therefore also 50%. The GA ran through 1000 generations.

The GA was run on five TSP instances taken from TSPLIB[1] [3]. These instances contain between 52 and 280 cities. On each instance, the GA was run twenty independent times, and the mean tour length, the standard deviation of the lengths, and the shortest tour length were recorded. Based on these statistics, non-parametric and parametric (normal) probabilities that another run would identify a tour shorter than the shortest were derived according to the discussions in the previous two sections. These probabilities were tested by running the GA 2 000 more times on each instance and noting the relative frequency of the event that a trial identified a tour shorter than the initial minimum.

Table 1 summarizes the results of these trials. For each TSP instance, it lists the mean, standard deviation, and minimum of the tour lengths returned by the GA's initial twenty trials; the two estimated probabilities that another trial would identify a shorter tour; and the number of additional trials, the number of those that found a shorter tour, and the relative frequency of the shorter tours.

The non-parametric probability estimates are based only on the sizes of the initial samples. Since these sizes are all the same (20), so are the non-parametric estimates (0.0476). The parametric probability estimates assume that, as a random variable, the outcome of a run of the GA on a particular TSP instance has a normal distribution. We estimate the distribution's mean and variance with the sample mean and sample variance, so the parametric estimates depend on the mean, standard deviation, and minimum of the initial trials' results. These estimates vary from 0.0281 to 0.0582.

[1] elib.zib.de/pub/Packages/mp-testdata/tsp/tsplib/index.html

Table 1. Results of the trials of the genetic algorithm on five TSP instances. For each instance, the table lists the mean, standard deviation, and minimum of the tour lengths the GA returned in twenty initial trials; the non-parametric and parametric (normal) estimated probabilities that one more trial would identify a shorter tour; and the number of additional trials, the number that found a shorter tour, and the relative frequency of shorter tours

	Initial sample			Est. probs.		Add'l	Num	Rel.
Instance	Mean	StdDev	Min	Non-par	Normal	trials	< min	freq.
berlin52	9834.8	497.80	8883	0.0476	0.0281	2000	18	0.0090
st70	1153.7	64.91	1052	0.0476	0.0582	2000	130	0.0650
eil76	890.1	38.81	828	0.0476	0.0548	2000	188	0.0940
ch130	17396.2	627.43	16216	0.0476	0.0301	2000	109	0.0545
a280	15148.1	314.99	14573	0.0476	0.0336	2000	201	0.1005

In the 2 000 additional runs of the GA on each instance, the relative frequencies of the event that a trial returns a tour shorter than the shortest among the initial runs varies from 0.0090 to 0.1005. On two instances (st70 and ch130), the non-parametric probability estimates are close to the relative frequencies, from which they differ by about 27% and 13%, respectively. On the other instances, the estimates differ from the relative frequencies by at least 47%. Similarly, on only one instance (st70) is the parametric estimated probability close to the relative frequency, differing from it by about 10%. On eil76 the difference is about 42%, and on the remaining three instances the difference is greater.

Though the non-parametric estimates are slightly more accurate than the parametric estimates, as we might expect with small sample sizes, these results make it difficult to conclude that either the non-parametric or the parametric probability analyses provide useful estimates of the probability of observing a better result on a future trial.

One explanation for the failure of the parametric analysis to accurately estimate the desired probability may lie with the size of the initial sample. Larger samples would provide more accurate estimates of the distributions' means and standard deviations and allow more accurate probability estimates.

To test this hypothesis, we used the mean and standard deviation of all 2020 results on each instance to re-estimate the probabilities of new results smaller than the initial minima, and compared the new estimates to the relative frequencies. Table 2 shows the results of these calculations. For each instance, the table lists the mean and standard deviation of all 2020 results, the original minimum result, the parametric probability estimate based on all the results, and the relative frequency of results less than the original minimum.

In Table 2, we see that the revised estimated probabilities are, with one exception, very close to the relative frequencies. This suggests that in general, the samples on which such estimates are based should indeed be larger, though it does not indicate how much larger might be adequate.

Table 2. The parametric estimated probabilities, based on all 2020 trials of the GA on each instance, compared to the relative frequencies

	Entire sample		Initial	Est. par.	Rel.
Instance	Mean	StdDev	min	prob.	freq.
berlin52	9891.5	435.36	8883	0.0104	0.0090
st70	1140.2	58.58	1052	0.0655	0.0650
eil76	878.2	38.13	828	0.0934	0.0940
ch130	17362.8	734.86	15112	0.0011	0.0545
a280	15126.3	435.19	14573	0.1020	0.1005

It is also possible that the distributions of the GA's results on the instances are not normal. Certainly if the GA achieves results that are near optimal (not particularly the case here), we would expect the distribution of those results to be skewed, and if a distribution is not normal, probability estimates based on the assumption that it is are not likely to be accurate.

To examine the possibility that the underlying distributions were not normal, we performed chi-square tests of goodness-of-fit on the five sets of 2 020 results. Each tested the null hypothesis that the values conformed to a normal distribution whose mean and variance were the mean and variance of all the values. On two instances (st70 and eil76), we reject this hypothesis at the 1% significance level. On two others (berlin52 and ch130), we reject the hypothesis at the 10% level. Only for the one remaining instance (a280) can we not conclude that the 2020 values do not conform to the specified normal distribution.

6 Conclusion

Estimates of the probability that an evolutionary algorithm will identify an improved result would clearly be useful. We have investigated two techniques for generating such estimates. One assumes nothing about the distribution of an EA's results on a particular problem instance. The other assumes that those results conform to a normal distribution. Though tests of these techniques used a simple genetic algorithm for the traveling salesman problem, nothing in them depends on a particular problem, coding, set of operators, or EA design.

In tests using the genetic algorithm and five instances of the TSP, we found that neither technique produced accurate estimates of the probability that another run of the GA would produce a better result than the best in a small initial sample. The non-parametric estimates were slightly more accurate than those based on the assumption of normality.

The parametric (normal) estimates may have failed for either of two reasons. The initial samples of the GA's performance may have been too small to accurately estimate the underlying distribution's mean and variance, and the underlying distribution may not have been normal. Still, we suggest that methods like these, based on larger initial sample sizes and assuming other underlying distributions, should be able to accurately estimate the probabilities we seek.

References

1. Whitley, D., Starkweather, T., Fuquay, D.: Scheduling problems and traveling salesmen: The genetic edge recombination operator. In Schaffer, J.D., ed.: Proceedings of the Third International Conference on Genetic Algorithms, San Mateo, CA, Morgan Kaufmann Publishers (1989) 133–140
2. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolutionary Programs. Third edn. Springer, Berlin (1996)
3. Reinelt, G.: TSPLIB - A traveling salesman problem library. ORSA Journal on Computing **3** (1991) 376–384

Introduction of a New Selection Parameter in Genetic Algorithm for Constrained Reliability Design Problems

Laure Rigal, Bruno Castanier, and Philippe Castagliola

IRCCYN/École des Mines de Nantes
La Chantrerie – 4, rue Alfred Kastler – BP 20722
F-44307 Nantes CEDEX 3 – France
1.rigal,bruno.castanier,philippe.castagliola@emn.fr

Abstract. In this article we propose to introduce a new selection parameter in Genetic Algorithms (GAs) for a class of constrained reliability design problems. Our work demonstrates two major points. The first one is that the populations are quickly included in the space of the feasible solutions for a sufficiently large selection of parameter value. The second one is that the value of the selection parameter controls the exploration strategy of the feasible space. These two properties illustrate that an adapted choice of the selection parameter value allows to improve the performance of GA. Furthermore, our numerical examples tend to show that, with an adapted choice of the selection parameter, these GAs are in practice more efficient than previously proposed GAs for this class of problems.

1 Introduction

Over the last few decades, due to the number of occurring accidents, safety has taken an increasingly significant role. Intense research has been carried out to obtain optimal safety system at minimal cost. In this paper we are interesting in a class of Reliability Design Problems (RDP) described in [7], [10]. These problems deal with the system structure and component choice optimization in order to obtain the best compromise between system reliability and cost. These problems are most frequently NP-hard. If conventional optimization tools, for example, integer programming, Lagrangian method, dynamic programming [9] are often ineffective, the use of GAs, in the RDP, provide some promising results. But in practice GAs are extremely complex to use because of the parameter tuning and the existence of many different ways to apply the mutation crossing-over and selection process [13]. The analysis of the given solutions usually requires an important experience in GAs practice. At the end, the choice of a particular GA is usually based on heuristic approaches, empirical results and GAs expert judgements. As far as we know, few theoretical works are related to the study of the performance of GA. Most of them [1], [2], [11], [12], [14] propose a modelisation based on Markov theory in order to describe its stationary properties. None of these studies prove that the population of GA converge towards

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 90–101, 2004.

a population only constituated by solutions which are global optimum even if the number of generations tends to infinity. By extending some classical results on Simulated Annealing [4], [5] to GA, Cerf [6] proves this convergence of a new GA with three parameters allowing the control of a specified cooling-schedule. However, the application of this new GA might vary a lot from one problem to another and seems to be very complicated to implement and to use. As far as we know this GA has never been implemented. Nevertheless, practical-oriented-studies seem to confirm that the time-dependent assumption for the parameters in Evolutionary Algorithms (EAs) is well fitted [18]. The choice and the design of optimal time-dependent parameters function are at least as difficult to find the best static as parameters. But in practical studies, they have proven their efficiency at least, with EA. The idea of letting the algorithm adjust its own parameters for free is indeed appealing [18]. The realistic formulation of RDP leads to generally constrained or highly constrained problems. However the application of GA on highly constrained optimization problem could fail to give the right solution, i.e. the optimal or nearly optimal solution [15]. Different approaches are used in order to improve the GA performance like repair operators and specific penalty functions. Because the repairs operators method is not well-fitted to our problems [10], we decide to adopt the penalty approach which seems to give promising results. There are three different ways to apply penalty methods in the selection process [17] : the *static method,* the *dynamic method* [16] which modifies the penalty coefficient along the evolution according to a user-defined-schedule and the *adaptive method* which gleans information from the population in order to update the value of penalty functions. All these methods focus on the tuning of the penalty function to access more rapidly to the feasible space. These approaches are particularly interesting to find a first feasible solution when the proportion of feasible solutions is very small (approximately less than one pour-cent). When the proportion of feasible solutions increases, upper than 10%, the control of the selection force, allows to direct the population very quickly in the feasible space. In fact, if the size of the feasible space is large, the exploration strategy of the feasible space has an important impact on the efficiency of the GA. In this article we propose to demonstrate that the selection parameter give the possibility to choose different exploration strategies of the feasible space. Consequently an adapted choice of this selection parameter allows to improve the efficiency of the GA. Our main goal is to propose both an efficient and practical optimization tool for constrained problems particularly encountered in RDP. In the first section, a modelisation of the GA proposed in [9] for a practical RDP based on Markov theory allows to characterize all the different steps of a GA and, particularly, the crossing-over. It is interesting to underline that, as far as we know, this is the first time that this crossing-over process has been modelled. This crossing-over is very difficult to model because of the addition of a random number of children. In the second section, we explain why and how we modify GA. In the third section, we apply the theory developed in [3] [6] on optimization problems with particular space structures in order to demonstrate the efficiency

of the proposed GAs. Finally in the last section, we present some asymptotical results of modified GA performance, illustrated by a numerical comparison.

2 Mathematical Model of the Classical GA

A general optimization problem is the maximization of an objective function $f : E \to [1; +\infty[$ subject to several constraints. The solution space E can be partitioned into the feasible solution space E_R and the unfeasible solution space $\bar{E}_R$. Let X_n be the population at the n-th iteration. A realization of the random population X_n is a sequence of m solutions $x = (x_1, x_2, \ldots, x_m) \in E^m$ where $x_i \in E$, $i = 1, \ldots, m$ is a vector of L binary variables (0 or 1) and denoted $x_i = (x_i^1, x_i^2, \ldots, x_i^L)$. Let $P(X_{n+1} \mid X_n)$ be the one-step probability transition matrix. A one-step transition from X_n to X_{n+1} is divided in three distinct phases: the mutation process, the crossing-over process and the selection process. Hence, the evaluation of the probability matrix $P(X_{n+1} \mid X_n)$ requires the characterization of each process.

2.1 Modeling of the Mutation Process

A mutation is a switch of a bit from 0 to 1 (or 1 to 0) with a given probability p_m (with $0 < p_m < 1$). Let U_n be the random population obtained after the mutation process of the population X_n, and u be the associated realization vector where $u = (u_1, u_2, \ldots, u_m) \in E^m$. The mathematical expression of the probability to obtain u_i after carrying out the mutation process on x_i is determined in [14]. Let d be the Hamming distance. The expression of the mutation process transition matrix α from X_n to U_n is, $\forall (x, u) \in E^m \times E^m$:

$$\alpha(x,u) = \prod_{i=1}^{m} \underbrace{p_m^{d(x_1,u_1)}}_{\text{mutation}} \times \underbrace{(1-p_m)^{L-d(x_1,u_1)}}_{\text{no mutation}}$$

2.2 Modeling of the Crossing-over Process

The considered crossing-over process recombines the pair of consecutive chromosomes with probability p_c in order to generate new chromosomes namely children by applying the classical one-point crossing-over rule described in page 6 of [13]. Let $CO_K(u_1, u_2)$ be the two children generated by the one point crossing-over process applied on two chromosomes (u_1, u_2), given the cut site K. One can remark that not all the couples generate two children. If a couple (u_{2i-1}, u_{2i}) does not generate, we then introduce two "virtual" chromosomes called "empty chromosome" denoted $v_\emptyset$. Note that, if the solution space dimension m is odd, the last chromosome u_m remains alone and, according to this new notation, generates an empty chromosome $v_\emptyset$. Let $E_\emptyset = E \cup \{v_\emptyset\}$ be the extended solution space. Let β_{E^2} be the probability to obtain (v_1, v_2) after carrying out one point crossing-over process on (u_1, u_2) :

$$\begin{aligned}\beta_{E^2}: \quad E^2 \times (E_\emptyset)^2 &\to \mathbb{R}^+ \\ ((u_1,u_2),(v_1,v_2)) &\mapsto \underbrace{\frac{p_c}{L-1}\sum_{k=1}^{L-1}\delta_2(CO_K(u_1,u_2);(v_1,v_2))}_{\text{expression1}} \\ &\quad + \underbrace{(1-p_c).\delta_2((v_\emptyset,v_\emptyset);(v_1,v_2))}_{\text{expression2}}\end{aligned} \tag{1}$$

where δ_j is a comparison function of 2 vectors of the same length j defined by, $\forall j \leq m$,

$$\begin{aligned}\delta_j: \quad E_\emptyset^j \times E_\emptyset^j &\to \mathbb{R}^+ \\ ((u_1,\ldots,u_j);(v_1,\ldots,v_j)) &\mapsto \begin{cases} 1 \text{ if } u_i = v_i, \ \forall i \leq j \\ 0 \text{ else} \end{cases}\end{aligned}$$

The two expressions in the right side of equation (1) correspond to :

- expression 1: the probability that the couple of parents (u_1, u_2) generates the children (v_1, v_2), in respect with the cutting site K.
- expression 2: the probability that no reproduction occurs with the couple (u_1, u_2).

In order to completely characterize the crossing-over process, we only have to consider all the opportunities for the one-point crossing-over. Let V_n be the random population obtained after the crossing-over process of the population, and v be the associated realization vector. The crossing-over process is characterized by the transition matrix β from U_n to V_n defined by, $\forall (u,v) \in E^m \times (E_\emptyset)^m$:

$$\beta(u,v) = \prod_{1\leq i\leq p} \beta_{E^2}\big((u_{2i-1},u_{2i}),(v_{2i-1},v_{2i})\big) \qquad \text{If } m = 2p$$

$$\beta(u,v) = \delta_1(v_m,v_\emptyset) \prod_{1\leq i\leq p} \beta_{E^2}\big((u_{2i-1},u_{2i}),(v_{2i-1},v_{2i})\big) \qquad \text{If } m = 2p+1$$

2.3 Modeling of the Selection Process

The selection process consists in creating a new population X_{n+1} by applying the proportional selection described in [14] on the population composed of U_n and V_n. Because of the introduction of $v_\emptyset$, we have to extend the objective function f to $f_\emptyset$ defined by:

$$\begin{aligned} f_\emptyset : E_\emptyset &\to \mathbb{R}^+ \\ s &\mapsto \begin{cases} f(s) & \text{if } s \in E \\ 0 & \text{if } s = v_\emptyset \end{cases}\end{aligned}$$

Let $z = (z_1, z_2, \ldots, z_{2m}) \in E^m \times E_\emptyset^m$ be a vector of the global population $U_n \times V_n$ and F be the probability to select the individual z_k from the population $z = (z_1, z_2, \ldots, z_{2m})$:

$$\begin{aligned} F : \{1,\ldots,2m\} \times E^m \times E_\emptyset^m &\to \mathbb{R}^+ \\ (k, z_1 \ldots, z_{2m}) &\mapsto \frac{f_\emptyset(z_k)}{\sum_{j=1}^{2m} f_\emptyset(z_j)} \end{aligned} \tag{2}$$

It can be demonstrated that the selection process is described by the transition matrix γ :

$$\forall (y,z) \in E^m \times \left(E^m \times E_\emptyset^m\right),\ \gamma(z,y) = \prod_{r=1}^{m} \Big[\sum_{k:z_k=y_r} F(k,z) \Big] \tag{3}$$

2.4 Characterisation of the Classical GA

Finally, with the evaluation of the the probabilities α, β γ and, considering all possible paths between the population X_n and X_{n+1}, we can define the one step transition probability of the classical GA:

$$P(X_{n+1} = y \mid X_n = x) = \sum_{(u,v)\in E^m \times E_\emptyset^m} \alpha(x,u)\beta(u,v)\gamma\big((u,v),y\big)$$

With the proposed modeling, we can easily demonstrate that this classical GA does not converge as previously demonstrated in [6], [12].

3 Introduction of a Selective Force Parameter

The purpose of this section is to propose a modified GA that quickly directs its research towards a part of the space which may contain the optimal solution. The modified GA that we propose is based on the results on simulated annealing [5] and on GAs [6]. In order to control the search, we have decided to introduce a new parameter ℓ which can be seen as the force to select the solutions in the population $U_n \times V_n$. Thus in the same way that we try to optimize the function f, we are going to obtain the maximization of the following function :

$$\begin{aligned} f_{\emptyset,\ell} : E_\emptyset &\to \mathbb{R}^+ \\ s &\mapsto \begin{cases} \exp(\ln(f(s))\ell) & \text{if } s \in E \\ 0 & \text{if } s = v_\emptyset \end{cases} \end{aligned}$$

The entire selection process remains the same one excepted for the mathematical expression of the objective function. Thus we deduce the mathematical expression of F_ℓ and γ_ℓ which are respectively the proportional selection probability and the matrix transition associated to ℓ by remplacing respectively in the expression 2 $f_\emptyset$ by $f_{\emptyset,\ell}$ and by remplacing in the expression 3 $F(k,z)$ by $F_\ell(k,z)$.

The mutation and reproduction transition matrices do not change. We denote by X_n^ℓ the population associated to the modified GA with the selection parameter ℓ. The transition probability of the chain $(X_n^\ell)_{n\geq 0}$ is given by: $\forall \ell \geq 1$,

$$P(X_{n+1}^\ell = x \mid X_n^\ell = y) = \sum_{(u,v)\in E^m \times E_\emptyset^m} \alpha(x,u)\beta(u,v)\gamma_\ell\big((u,v),y\big)$$

Let us define the set $D(y)$ of all the couples (u,v) which join any point x to y by:

$$\begin{aligned} D(y) &= \{(u,v) \in E^m \times E_\emptyset^m,\ \alpha(x,u)\beta(u,v)\gamma_\ell\big((u,v),y\big) > 0\} \\ &= \{(u,v) \in E^m \times E_\emptyset^m,\ \beta(u,v) > 0 \text{ and } [(u,v)] \supseteq [y]\} \end{aligned}$$

Where $[y]$ represents the chromosomes of the vector y. With this notation, we can have:

$$P(X_{n+1}^{\ell} = x \mid X_n^{\ell} = y) = \sum_{(u,v)\in D(y)} \alpha(x,u)\beta(u,v)\gamma_{\ell}\big((u,v),y\big) \tag{4}$$

The introduction of the parameter ℓ does not change the properties of the Markov chain and consequently $(X_n^{\ell})_{n\geq 0}$ is an homogenous, irreducible, aperiodic Markov chain that admits an unique invariant probability measure μ_{ℓ}.

If we increase the value of the selection parameter, the associated modified GA is more selective. In order to study the asymptotical behavior of the Modified GA, we propose to focus on the two extreme cases $\ell = 1$ and $\ell = +\infty$.

If $\ell = 1$, the modified GA is simply the Classical GA. Because of its proportional selection process, the major drawback of this algorithm, for our specific RDP, is that it conserves in the current population many solutions which do not satisfy the constraints. This is particularly important when the feasible space solution is very small compared to the whole solution space. Consequently this GA consumes a lot of time in treating unfeasible solutions.

When ℓ tends to infinity, the respective transition probabilities of the mutation and crossing over process do not change. On the other hand, the selection process tends towards a transition probability $\gamma_{+\infty}$ which can be proven to be:

$$\gamma_{+\infty}(z,y) = \lim_{\ell\to+\infty} P\big(X_{n+1}^{\ell} = y \mid (U_n^{\ell}, V_n^{\ell}) = z\big) = \prod_{r=1}^{m} \frac{1_{\hat{z}}(y_r)z(y_r)}{|\hat{z}|} \tag{5}$$

Where, $\forall z = (z_1, \ldots, z_{2m}) \in E^m \times E_{\emptyset}^m$, we have:

- $\hat{z} = \{z_k : 1 \leq k \leq 2m,\ z_k \in E,\ f(z_k) = \hat{f}(z)\}$ with $\hat{f}(z) = \max_{\{1\leq k\leq 2m,\ z_k\in E\}} f(z_k)$
- $\forall i \in E,\ z(i) = |\{1 \leq k \leq 2m,\ z_k = i\}|$
- $1_{\hat{z}}(y_k)$ is zero if y_k belongs to $\hat{z}$ and zero if not.

This GA has a well adapted property for our specific reliability design problem. The proof of equation (5) is not given here but this one can be interpreted as follow : if there exists in the current population $(U_n \times V_n)$ at least one feasible solution then all the population X_{n+1} after the selection process is necessarily in the feasible space solution. Nevertheless, its exploration strategy of the feasible space could be not adapted for some problems because of the elimination after the mutation and crossing-over process, of all the feasible solutions which are not an optimal solution of the population. Nevertheless, in the next section we are going to prove that it is possible to determine a group of GA appropriated for eliminating unfeasible solutions where each of these GAs has a different strategy of the feasible space exploration.

4 Performance of the Modified GAs for a Class of RDP

We propose in the first subsection 4.1 to characterize the asymptotical behavior of the modified GA. This approach allows us to deduce how the selection parameter direct the strategy of the whole exploration space. Consequently, we clearly show, that a sufficiently large value of the selection parameter ℓ allows to eliminate very quickly the unfeasible solutions. And at the same time, the value of the selection parameter could determine a feasible space exploration strategy which favours the diversity or, on the contrary, which favours the intensification towards the best solution of the feasible space. Finally, in the subsection 4.2, by underlining the correspondance between the length of the population and the value of the selection force parameter ℓ, we demonstrate that the modified GA will be able to concentrate very quickly its search in the restricted area of the feasible solutions.

4.1 Asymptotical Behavior

In this subsection, a study of the behavior of modified GAs probability transition when ℓ is large is presented. By transposing the reasoning proposed in [6] p. 55–57 to our modelisation, we can demonstrate that:

$$P(X_{n+1}^{\ell} = y \mid X_n^{\ell} = x) \underset{\ell \to +\infty}{\propto} \exp(-V^*(y)\ \ell) \tag{6}$$

Where

$$V^*(y) = \min_{(u,v)\in D(y)} \sum_{k=1}^{m} \left[\ln(\hat{f}(u,v)) - \ln(f(y_k)) \right] \tag{7}$$

This quantity is not null only for populations verifying $V^*(y) = 0$. Consequently, when ℓ becomes very large, all the population y which has a strictly positive value of $V^*(y)$ tends to be eliminated. And we can also demonstrate:

Lemma 1.

$$\forall y \in E^m,\ V^*(y) = 0 \ \Leftrightarrow\ y \in S$$

Finally, with the proposition (1.15) p. 19 in [5] we can easily demonstrate that $\lim_{\ell \to \infty} \mu_\ell(y) > 0 \Leftrightarrow\ y \in S$ and so, the modified GA may concentrates its exploration towards the population which are in S.

However, we prefer a GA which concentrates quickly its search towards the populations composed of the best feasible solutions. The equation 6 illustrates, that when the value of ℓ increases, the search is directed towards the populations y with smallest value of $V^*(y)$. Let G be the set of the populations which contain at least one unfeasible and one feasible solution. We assume that a static penalty method is used. Because of the large value of the static penalty, it can be shown that:

$$\forall y_1 \in G, \forall y_e \notin G,\ V^*(y_1) >> V^*(y_2)$$

Consequently, for any sufficiently large value of ℓ, GA eliminates the population contained in G. In the next subsection, with any sufficiently large value of m, we prove that this property allows to eliminate the unfeasible solutions. Thus, for any sufficiently large values of ℓ and m, GA concentrates its search towards the feasible solutions. Furthermore, in RDP, for any feasible solution, the corresponding objective function f is equal to the reliability of the system plus one. The values of f are in [1,2]. Thus, for any population composed only of feasible solutions, the value of $V^*(y)$ is small. Consequently, because of the equation 6, if the value of ℓ is not too large it seems to be possible to keep in the population of GA a variety of feasible solutions. However, if we increase the value of ℓ, the exploration of the feasible solutions space becomes more selective. Finally, in tuning different large values of ℓ, we obtain GAs that eliminate unfeasible solutions. And the exploration strategy for such GAs in the feasible space favors more or less the population diversity.

4.2 New GAs That Concentrate Quickly Their Search towards the Space of the Feasible Solutions

The aim of this subsection is to propose a GA which eliminate the unfeasible solutions with a probability close to one after only one iteration (Theorem 1). For that, we observe that the set of the population which contains some unfeasible solutions can be partioned into two sets. The first set contains at least one unfeasible and one feasible solution.

As we have already said in the last subsection, for a sufficiently large value of ℓ, the population which belongs to the first group can be eliminated. In order to eliminate the population which belongs to the second group composed only by the unfeasible solutions, we suggest to use the fact that the feasible solution space $\bar{E_R}$ is very small compared to the whole space solution E. The uniform distribution of the initial population X_0, permit to obtain:

$$P(X_0 \in \bar{E_R}^m) = \left(\frac{|\bar{E_R}|}{|E|}\right)^m > 0$$

which will tend to zero when the value of m tends to infinity. Hence we have $\forall \varepsilon > 0$, $\exists m \in \mathbb{N}$, $P(X_0 \in \bar{E_R}^m) < \frac{\varepsilon}{2}$. This last property is exactly the same for the population U_0 because of the conservation of the uniform distribution after the mutation process. So we have:

$$P(U_0 = u) = \sum_{v \in E_\emptyset^m} P\big((U_0, V_0) = (u, v)\big) = \frac{1}{|E|^m} \underset{m \to +\infty}{\to} 0 \tag{8}$$

By a similar demonstration in [6] p. 55, we can demonstrate that if the population (u, v) contains at least one feasible solution and the population y is composed only with unfeasible solutions, we have:

$$\lim_{\ell \to \infty} P\big(X_1^\ell = y \mid (U_0^\ell, V_0^\ell) = (u, v)\big) = 0$$

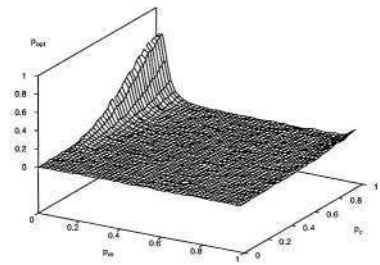
By using equations (4) and (8), $\forall y \in \bar{E_R}^m$, $\forall \ell \geq 1$ we can easily demonstrate:

$$P(X_1^\ell = y) \leq \sum_{\substack{(u,v)\in E^m \setminus \bar{E_R}^m \times E_\emptyset^m \\ [y]\subseteq[u,v]}} \frac{P\left(X_1^\ell = y \mid (U_0^\ell, V_0^\ell) = (u,v)\right)}{|E|^m} + P(X_0 \in \bar{E_R}^m)$$

Hence, we have proven that the probability $P(X_1^\ell = y)$ tends to zero when ℓ and m tend to $+\infty$ for all populations that just contain the unfeasible solutions. We have to remind that we have proven at the beginning of this subsection, that this result is also true for populations containing at least one unfeasible and one feasible solutions for any m. Finally, we have proven that :

Theorem 1. $\forall\, \varepsilon > 0\ \exists m_\varepsilon > 0\ ,\ \exists\, \ell_\varepsilon > 0 : \forall\, y \in E^m,\ \{y\} \cap \bar{E_R}^m \neq \emptyset,$
$\forall\, m \geq m_\varepsilon,\ \forall \ell \geq \ell_\varepsilon$

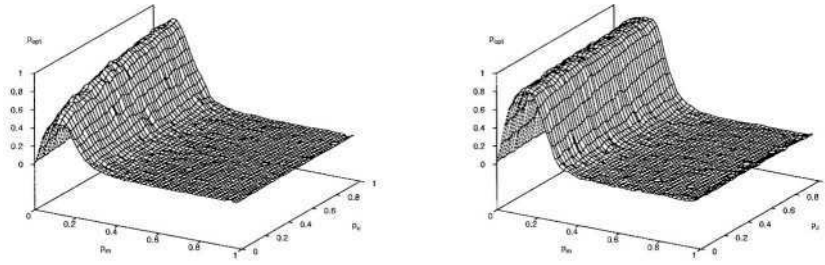
$$P(X_1^\ell = y) \leq \varepsilon \ \square$$



(a) Classical Selective GA

4.3 Numerical Results

In order to illustrate the effectiveness of our new GAs for constrained RDP problem, we will deal with a numerical example described in [8]. This example is a problem for distributing redundant units in order to maximize the reliability of a system subject to three non-linear and separable constraints. The choice of this example is motivated by the fact that this problem belongs to the well-know class of RDP [7] where the reliability expression are available [7]. We propose to apply three different GAs for solving this example the Modified GA with a large value of the selection force parameter (ℓ = 150), the Modified GA with a very large value of the selection force parameter (ℓ = 1000) and the "Classical Selective GA" which is very similar to a GA frequently used to solve these problems and described in [8]. These two GAs have exactly the same selection process. This selection process uses a static penalty method which sets the fitness value of any



(b) Modified GA with ℓ=150 (c) Modified GA with ℓ=1000

Fig. 1. Probabilities to obtain the optimal solution after 20 generations when p_m and p_c vary from 0 to 1 - parameters: population size = 100 - number of trials = 1000

unfeasible solution equal to –9999. The process of selecting the chromosomes in order to form the population of the next generation is completely deterministic. It selects m chromosomes from the current population and the set of offspring together in decreasing order of the fitness values. The mutation and crossing over process are described respectively in the Subsection 2.1 and in the Subsection 2.2. The Fig.1(b), 1(c) and 1(a) show the respective probabilities for the Modified GA with $\ell = 150$, the Modified GA with $\ell = 1000$, and the "Classical Selective GA" to obtain the optimal solution in a given number of generations when the mutation and crossing-over probabilities vary from 0 to 1. To obtain these results, the experiments are carried out by applying this three GAs with a population size of 100 chromosomes, the number of generations is 20 for given p_m, p_c and the same codage as proposed in [8]. The probability to obtain the optimal solution has been estimated by computing the average of successful experiments on 1000 trials. The feasible space is composed of 262144 solutions. The proportion of the feasible solutions is 20.8%. Thus, it is strongly possible that in a population of 100 chromosomes, there is at least one feasible solution. The solutions are coded with 18 bits and more than 90% of the feasible solutions have between six and nine bits coded by one. Thus the mutation and crossing-over process apply on feasible solutions seem to be able to generate new feasible solutions. Consequently, to explore the space of feasible solutions it seems to be more appropriated to have a population constituated only by feasible solutions after the selection process. However, the population of the "Classical Selective GA" after the selection process are composed by many unfeasible solutions, because of the deterministic aspect of the selection process. On the contrary, for any sufficiently large value of the selection parameter, Modified GAs allows to obtain a population after the selection process only composed of feasible solutions. Thus, Modified GA seems to be more appropriated for this numerical problem than the

"Classical Selective GA". This is illustrated by the results which demonstrate for sufficiently large value of ℓ, that modified GA obtain a better performance than the "Classical Selective GA" for any value of p_c and any value of p_m in the interval [0,0.2]. Thus, the application of the modified GA is more easier because of the determination of the appropriated values of p_m and p_c. It is interesting to underline that the superiority of these new GAs should be more important for larger problems. We also remark, when the value of ℓ increases, that the results of Modified GA are improved. Obviously when ℓ is equal to 150 all the unfeasible solutions in the population of Modified GA are quickly eliminated. Thus the increasing of the selection parameter value to 1000, seems to have no consequence on the elimination of the unfeasible solutions. On the contrary this increasing seems to have an important impact on the strategy of the feasible space exploration. The selective strategy seems to be the most appropriated strategy of the feasible space exploration.

5 Conclusion

In this paper, we have proven that an adapted choice of the selection parameter ℓ allows to obtain the elimination of the unfeasible solutions with a very high probability, near to one, and allows to obtain an efficient strategy of the feasible space exploration. This last aspect, when the size of the feasible space is large, makes possible to decrease the time of computation. Consequently, for RDP, where the proportion of feasible solutions is not too small, an adapted choice of the selection parameter seems to lead GA more quickly to the convergence. In addition the selection parameter is used exactly in the same way that the mutation and the crossing-over probability. Consequently, for the users of GA, it is very easy to change the strategy of the space exploration by tuning different values of the selection parameter. Nevertheless, it is difficult to determine exactly the value of the selection parameter. Thus, it turns out to be clear that further theoretical and experimental studies could improve the performance of GA for this kind of problems. Finally, it is also interesting to extend this work to optimization problems which have a solution space which contains several disjoint feasible subspaces of solutions.

References

1. D. Goldberg, Segrest: Finite Markov Chain analysis of genetic algorithms. Proceedings of the Second international Conference on Genetic Algorithms. (1987) 1–8.
2. J. Horn: Finite Markov Chain Analysis of Genetic Algorithms with Niching. S. Forrest. (1993) 110–117
3. M.I. Freidlin, A.D. Wentzell: Random perturbations of dynamical systems. Springer-Verlag. (1984)
4. O. Cantoni: Rough large deviations estimates for simulated annealing. The Annals of Probablity. (1992)

5. A. Trouve: Parallelisation massive du recuit simule PhD Thesis University paris XI. (1993), in French.
6. R. Cerf: Une theorie asymptotique des algorithmes genetiques. PhD Thesis University Montpellier II. (1994), in French.
7. F.A. Tillman: Optimization by integer programming constrained reliability problems with several modes of failure. IEEE Transaction on Reliability. (1969)
8. T. Yokota, M. Gen, K. Ida: System Reliability Optimization Problems with several Failure Modes by Genetic Algorithm. Japanese Journal of fuzzy theory and systems. **Vol.7 no. I** (1995)
9. W. Kuo, F.A. Tillman, C.L. Hwang, V.R. Prasad: Optimal reliability design. Cambridge University Press. (1992)
10. D.W. Davis, A.E. Smith: Penalty guided genetic search for reliability design optimization Computers ind. Engng.**Vol.30 no. 4** (1996) 895–904
11. T.E. Davis, J.C. Principe: A simulated annealing like convergence theory for the simple genetic algorithm. Proceedings of the fourth Int. Conf. on Genetic Algorithms. (1991) 174–181
12. Nix , Vose: Modeling genetic algorithms with Markov chains. it Annals of Mathematics and Artificial Intelligence. **Vol.5 no. 1** (1992)
13. D. Whitley: A genetic algorithm Tutorial. Technical report Colorado State University. (1993)
14. G. Rudolph: Convergence Analysis of Canonical Genetic Algorithms. IEE transaction on Neural Networks. (1994)
15. M. Schoenauer, S. Xanthakis: Constrained GA optimization. Proceedings of the 5th International Conference of Genetic Algorithms. (1993)
16. Z. Michalewicz and N. Attia: Evolutionary optimization of constrained problems. A.V. Sebald and L.J Fogel, editors. Proceedings of the 3rd Annual Conference on Evolutionary Programming. (1994)
17. S. Ben Hamida and M. Schoenauer: An adaptive Algorithm for constrained optimization problems. Parallel Problem Solving from Nature - PPSN VI, 6th International Conference, Paris, France. (2000) 529–538
18. A.E. Eiben, R. Hinterding and Z. Michalewicz: Parameter Control in Evolutionary Algorithms IEEE Trans. on Evolutionary Computation. (2000)

Improving the Performance of a Genetic Algorithm Using a Variable-Reordering Algorithm

Eduardo Rodriguez-Tello[1] and Jose Torres-Jimenez[2]

[1] LERIA, Université d'Angers
2 Boulevard Lavoisier, 49045 Angers, France
ertello@info.univ-angers.fr
[2] ITESM Campus Cuernavaca, Computer Science Department.
Av. Paseo de la Reforma 182-A. 62589 Temixco Morelos, Mexico
jtj@itesm.mx

Abstract. Genetic algorithms have been successfully applied to many difficult problems but there have been some disappointing results as well. In these cases the choice of the internal representation and genetic operators greatly conditions the result.
In this paper a GA and a reordering algorithm were used for solve SAT instances. The reordering algorithm produces a more suitable encoding for a GA that enables a GA performance improvement. The attained improvement relies on the building-block hypothesis, which states that a GA works well when short, low-order, highly-fit schemata (building blocks) recombine to form even more highly fit higher-order schemata. The reordering algorithm delivers a representation which has the most related bits (i.e. Boolean variables) in closer positions inside the chromosome.
The results of experimentation demonstrated that the proposed approach improves the performance of a simple GA in all the tests accomplished. These experiments also allow us to observe the relation among the internal representation, the genetic operators and the performance of a GA.

1 Introduction

A genetic algorithm (GA) is based on three elemental parts: an internal representation, an external evaluation function and an evolutionary mechanism. The GA is considered to be successful if a population of highly fit individuals evolves as a result of iterating this procedure.

The GAs, unlike other optimization algorithms, use in the search process of a solution not only the fitness values, but also the similarities that exist between certain patterns with high aptitude, that is to say, they change the emphasis of searching complete strings by discovering partially adapted strings. Given this fact it is useful to study the similarities of these patterns along with its corresponding fitness values, and specially to investigate the structural correlations of

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 102–113, 2004.

strings with exceptional fitness values. Formally, the similarities between strings are defined by means of a *schema.* A schema (on the binary alphabet, this does not mean lost of generality) is a string of the following type: $(a_1, a_2, ..., a_i, ..., a_l)$, $a_i \in \{0, 1, *\}$.

The character "$*$" is used for representing a wildcard which can take the values 0 or 1. A schema describes a subspace of strings [12]. For example $S_1 = (11*00*)$ is a schema in a population of strings with length $k = 6$ which represents the strings: 111001,111000,110001,110000. The schema's order $o(S)$, can be defined as the length of the string minus the total number of wildcard characters, in the previous example $o(S_1) = 4$; and the schema's length $\delta(S)$ is the distance between the two, non wildcard, most separated characters of the schema ($\delta(S_1) = 4$).

Given that a string is an instance of 3^l possible schemata, where l is the length of the string, when its fitness value is verified, it is also derived an important amount of implicit information related with its schemata. Holland called this fact *implicit parallelism*[19], because the searching process is directed simultaneously in many hyperplanes of the search space.

The schemata theorem, which is believed to be a primary source of the GA's search power, states that a GA works well when short, low-order, highly-fit schemata *(building blocks)* recombine to form even more highly fit higher-order schemata.

In this sense when examining GA hardness it is important to realize that the hardness is essentially due to the internal representation of the search space, and the way genetic operators act upon it [28]. For this reason the representation of a problem must be designed in such a way that the gene interaction (*epistasis*) is kept as low as possible and the building blocks may form to guide the convergence process towards the global optimum.

In an internal representation each *locus* (bit within the chromosome) discriminates between certain subsets of the search space. Only in one extreme case (zero epistasis, [6]), all loci are of equal importance. Most often, though, certain loci are more important than others in that they discriminate between larger chromosome chunks. Or, correlations between different loci are explicitly or implicitly present. Ideally we would like to order the loci by their discriminating power, and we would like correlated loci to be grouped together. In this paper an algorithm which tries to achieve this is presented.

This algorithm is implemented by a simulated annealing algorithm (SA), which transforms the representation of the problem by reordering its variables in such a way that the most related ones can be placed in closer positions inside the chromosome.

The rest of this paper is organized as follows: Section 2, presents a historical resume of the GAs performance on the satisfiability problem. In section 3, a detailed description of the GA representation issue for the satisfiability problem is presented. Section 4 focuses on the proposed reordering algorithm and its implementation details. In section 5, the experimental results and comparisons are presented. Finally, in last section, the conclusions of this work are discussed.

2 GAs and the Satisfiability Problem

We are specially interested in applying GAs to the satisfiability (SAT) problem given its great importance in computer science both in theory and in practice. In theory, SAT was the first problem which has been shown to be NP-complete [4]. In practice, it has many applications in different research fields such as planning, formal verification, and knowledge representation; to mention only some.

The SAT problem involves finding an assignment to a set of Boolean variables $x_1, x_2, ..., x_N$ that satisfies a set of constraints represented by a well-formed Boolean formula in CNF format $F : \mathbb{B}^N \rightarrow \mathbb{B}$, $\mathbb{B} = \{0, 1\}$, i.e., a conjunction of clauses, each of which is a disjunction of variables. If such assignment exists the instance is called *satisfiable* and *unsatisfiable* otherwise.

Due to its theoretical and practical relevance the SAT problem has been extensively studied and many exact and heuristic algorithms have been introduced. Theoretically, exact methods are able to find the solution, or to prove that no solution exists provided that no time constraint is present. However, given the combinatorial nature of the problem, they have an exponential wort-case complexity. In contrast heuristic algorithms can find solutions to satisfiable instances quickly, but they do not guarantee to give a definitive answer to all problem instances.

Existing heuristic algorithms for SAT are essentially based on local search methods [1,27] and evolutionary strategies [20,8].

Even though the GAs have been successfully applied to many NP-complete problems, some disappointing results question the ability of GAs to solve SAT. De Jong and Spears proposed a classical GA for SAT and concluded that their GA could not outperform highly tuned, problem-specific algorithms [20]. This result was confirmed experimentally in [9], they reported scarce performance of classical GAs when compared with local search methods. In addition, Rana and Whitley showed pure GAs being unsuitable for the MAXSAT fitness function, which counts the number of satisfied clauses, because the corresponding domain contains misleading low-order schema information, and the search space tends to result in similar schema fitness averages [26]. Recent results showed that GAs can nevertheless yield good results for SAT if equipped with additional techniques. These techniques include adaptive fitness functions, problem-specific genetic operators, and local optimization [8,9,24,13].

We have strong reasons to think that the negative results in the use of classical GA for SAT is caused by the effect of an inappropriate representation. In the next section, the representation issue will be described in greater detail.

3 The Representation Issue

SAT has a natural representation in GAs: binary strings of length N in which the $i-th$ bit represents the truth value of the $i-th$ Boolean variable present in the SAT formula, surprisingly, no efficient classical genetic algorithm has been found yet using this representation alone [14]. We have strong reasons to think that this poor performance is caused by the effect of an inappropriate representation.

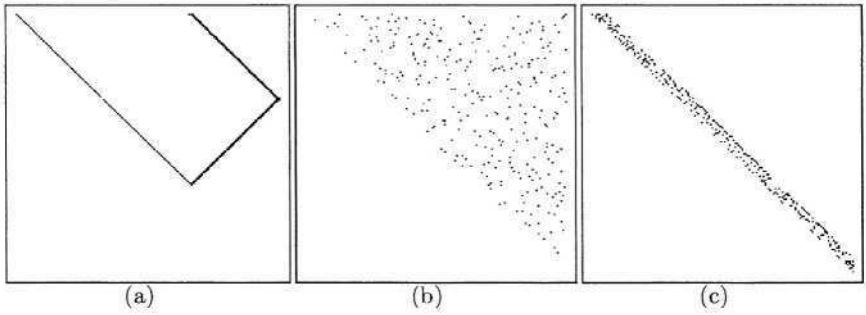


Fig. 1. (a) Closeness' graph for Dubois50.cnf original variable-ordering, (b) Closeness' graph for the Dubois50.cnf with a bad variable-ordering, (c) Closeness' graph for the Dubois50.cnf with a good variable-ordering.

An easy extension of the building block hypothesis [12] is that a chromosome representation must satisfy the fact that related variables are coded in closer positions in the chromosome representation. As a consequence, it is very desirable that two variables that participate in the same clause appear close in the chromosome representation, and through a variable reordering algorithm it is possible to control the "closeness" of SAT instance related variables.

A graphical view of variables' "closeness" can be constructed using the graph of a binary matrix R s.t. $R_{i,j} = 1$ if the $i-th$ variable appears related with the $j-th$ variable at least one time in a clause and $(i < j)$, and otherwise $R_{i,j} = 0$. The graph of the binary matrix R contains a dot in position (i, j) just in case $R_{i,j} = 1$,

The SAT instance *Dubois*50.*cnf* [1] will be used for presenting graphs that correspond to: a) the original variable-ordering, b) a bad variable-reordering, and c) a good variable-reordering (a good reordering means that related variables appear in closer positions within the chromosome representation saw by a GA). In Figure 1(a) the graph that corresponds to the original ordering of the SAT instance *Dubois*50.*cnf* is presented, in this ordering two related variables are separated at most 101 positions (i.e. if the two related variables are i and j, then $\max(abs(i-j)) = 101$). In Figure 1(b) the graph corresponding to a bad reordering for the SAT instance *Dubois*50.*cnf* is presented, in this case two related variables are separated at most 149 positions (this corresponds to a worst reordering given that two related variables appear in the most far positions within the chromosome, i.e. first chromosome position and last chromosome position). In Figure 1(c) the graph corresponding to a good reordering is presented, in this case two related variables are separated at most 10 positions (i.e. $\max(abs(i-j)) = 10$, where i and j are two related variables). As a general guideline if the dots appear near the main diagonal of the closeness' graph that indicates a variable-ordering, it implies that the representation is good.

[1] It consists of 150 variables and 400 clauses and can be found at http://www.satlib.org/benchm.html

4 The Reordering Algorithm

The proposed algorithm has as input a CNF instance, this CNF formula is then converted into a weighted graph, where each variable is represented by a node, and each weighted edge denotes how many times the variable i is related with the variable j. (see Figure 2(a)). A bandwidth minimization algorithm [30] is applied to the resulting weighted graph to produce a more suitable ordering of the weighted graph nodes. This ordering is translated back into an ordering of variables in the original CNF instance. After that, the preprocessed CNF formula can be used as input to a classical GA (see Figure 2(b)). Note that this preprocessing algorithm does not change the original problem only rename its variables.

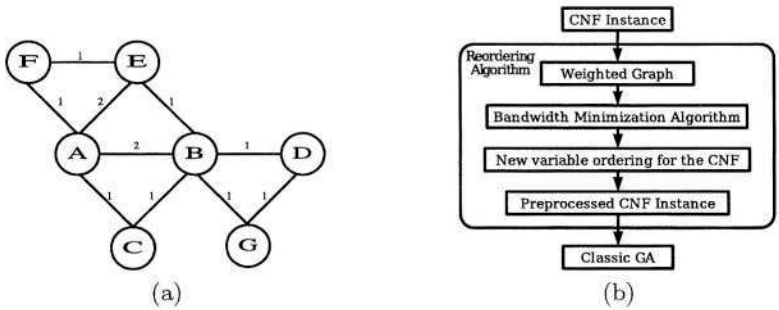


Fig. 2. (a) Weighted graph corresponding to the SAT instance: $(\sim A \vee B \vee \sim C) \wedge (A \vee B \vee E) \wedge (A \vee E \vee \sim F) \wedge (\sim B \vee D \vee \sim G)$ (b) Reordering variable heuristic based on a Bandwidth Minimization Algorithm.

4.1 Some Important Definitions

There are essentially two ways in which the bandwidth minimization problem (BMP) can be approached, whether as a graph or as a matrix. The equivalence of a graph and a matrix is made clear by replacing the nonzero entries of the matrix by $1's$ and interpreting the result as the adjacency matrix of a graph.

The Matrix Bandwidth Minimization Problem seems to have been originated in the 1950's when structural engineers first analyzed steel frameworks by computer manipulation of their structural matrices [22,23]. In order that operations like inversion and finding determinants take the least time as possible, many efforts were made to discover an equivalent matrix in which all the nonzero entries would lay within a narrow band near the main diagonal (hence the term "bandwidth")[3].

The BMP for graphs (BMPG) was proposed independently by Harper [18] and Harary [17]. This can be defined as finding a labeling for the vertices of a graph, where the maximum absolute difference between labels of each pair of connected vertices is minimum.

Formally, Let $G = (V, E)$ be a finite undirected graph, where V defines the set of vertices (labeled from 1 to N) and E is the set of edges. And a linear layout $\tau = \{\tau_1, \tau_2, ..., \tau_N\}$ of G is a permutation over $\{1, 2, ...N\}$, where τ_i denotes the label of the vertex that originally was identified with the label i. The bandwidth β of G for a layout τ is:

$$\beta_\tau(G) = Max_{\{u,v\}\in E}|\tau(u) - \tau(v)|. \quad (1)$$

Then the BMPG can be defined as finding a layout τ for which $\beta_\tau(G)$ is minimum.

The first work that helped to understand the computer complexity of the BMPG, was developed by Papadimitriou, who demonstrated that the decision problem associated to the BMPG is a NP-complete problem [25]. Later, it was demonstrated that the BMPG is NP-complete even for trees with a maximum degree of three [10].

There are several algorithms reported to solve the BMPG, they can be divided into two classes: exact and approximate algorithms. Exact algorithms, guaranteed always to discover the optimal bandwidth, example of exact algorithm is the one published by Gurari and Sudborough [15], it solves the BMPG in $O(N^k)$ steps, where k is the bandwidth searched for that graph. Approximate algorithms are best known as approximate in the sense that they do not guarantee to find the actual bandwidth of the graph, examples of this sort of algorithms are [5,11,16].

All algorithms mentioned in last paragraph use as a measure of the quality for a solution β. The only reported exceptions are: Dueck's work in which not only β is used, but he also takes into account differences among adjacent vertices close to β [7], and [30] where a new measure, namely γ, is proposed. The advantage of this new measure is the ability to distinguish even the smallest improvements that not necessarily lower the value of β.

Based in this previously reported work, we have implemented an algorithm for improving the representation of a problem used for a GA. The new representation produced by our algorithm satisfies the condition that the most related variables of the problem are placed in closer positions inside the chromosome. Next the implementation details of this algorithm are presented.

4.2 Implementation Details

The reordering algorithm approximates the bandwidth of a graph G by examining randomly generated layouts τ of G. These new layouts are generated by interchanging a pair of distinct labels of the set of vertices V. This interchanging operation is called a *move*.

The reordering algorithm begins initializing some parameters as the temperature, the maximum number of accepted moves at each temperature, and the cooling rate. The algorithm continues by randomly generating a move and then calculating the change in the cost function for the new labelling of the graph. If the cost decreases then the move is accepted. Otherwise, it is accepted with

probability $P(\Delta C) = e^{-\Delta C/T}$ where T is the temperature and ΔC is the increase in cost that would result from a particular move. The next temperature is obtained by using the relation $T_n = T_{n-1} * 0.85$. The minimum bandwidth of the labellings generated by the algorithm up that point is stored. The number of accepted moves is counted and if it falls below a given limit then the system is *frozen.*

The parameters of the SA algorithm were chosen experimentally, and taking into account some related work reported in [21,30,29]. It is important to remark that the maximum number of accepted moves at each temperature, depends directly on the number of edges of the graph, because more moves are required for denser graphs. Next the reordering algorithm is presented:

```
Procedure SA(G)
  Set initial temperature (T) and all the parameters
  Generate an initial random labelling for G
  Calculate its bandwidth (old)
  Repeat
     Repeat
       Generate a random move
       Calculate its bandwidth (new)
       If ((new-old) < 0) Or (Random[0,1) < Exp(-(new-old)/T))
          Accept the move
          If (new < old)
             old = new
          EndIf
       EndIf
     Until pre-defined number of iterations for this
temperature
     Reduce the temperature (Tn = Tn-1* 0.85)
  Until temperature > minimum temperature
EndSA
```

5 Experimental Results

To evaluate the effectiveness the proposed reordering algorithm, a classic GA was implemented. Its internal representation is based on binary strings of length N in which the $i-th$ bit represents the truth value of the $i-th$ Boolean variable of the problem. This chromosome definition has the following advantages: It is of fixed length, binary, context independent in the sense that the meaning of one bit is unaffected by changing the value of other bits, and permits the use of classic genetic operators defined by Holland [19], which have strong mathematical foundations.

Recombination is done in the standard way using the two-point crossover operator, and the bit flipping operator. The former has been applied at a 70% rate, and the latter at a 0.01% rate (these parameter values were adjusted through

experimentation). Selection operator is similar to the tournament selection reported in [2]; randomly three elements of the population are selected and the one with the better fitness is chosen.

The choice of the fitness function is an important aspect of any genetic optimization procedure. Firstly, in order to efficiently test each individual and determine if it is able to survive, the fitness function must be as simple as possible. Secondly, it must be sensitive enough to locate promising search regions on the space of solutions. Finally, the fitness function must be consistent: a solution that is better than others must have a better fitness value.

For the SAT problem the fitness function used is the simplest and most intuitive one, the fraction of the clauses that are satisfied by the assignment. More formally:

$$f(chromosome) = \frac{1}{M}\sum_{i=1}^{M} f(C_i) \tag{2}$$

Where M is the number of clauses in the problem and the contribution of each clause $f(C_i)$ is 1 if the clause is satisfied and 0 otherwise.

All the experiments were done using a population size fixed at: $\lfloor 1.6 * N \rfloor$, where N is the number of variables in the SAT problem. The termination condition used for this algorithm is either when a solution that satisfies all the clauses is found, or when the maximum number of 500 generations is reached.

The proposed reordering algorithm was tested on several classes of satisfiable and unsatisfiable benchmark instances as those of the second DIMACS challenge and flat graph coloring instances. They are easily available from the SATLIB web site[2]. For each of the selected instances 40 independent runs were executed, 20 using the reordering algorithm plus the classic GA (R+GA), and 20 solving the problem with the use of the same basic GA. The results reported here, are data averaged over the 20 corresponding runs.

Both the reordering algorithm and the GA have been implemented in C programming language and ran into a Pentium 4 1.7 Ghz. with 256 MB of RAM.

Table 1 presents the comparison between R+GA and the GA. Data included in this comparison are: name of the formula; number of variables N; number of clauses M; β_i and β_f represent the initial and final bandwidth obtained with the reordering algorithm; the CPU time in seconds used by the reordering algorithm is T_β. Additionally, for each approach it is presented: the number of satisfied clauses SC, and the CPU time in seconds T. It is important to remark that the times presented for the combined approach in the column T_{Total} take into account the CPU time used for the preprocessing algorithm.

The results of experimentation presented showed that the combined approach R+GA outperforms the simple GA in all the tests accomplished, not only in solution quality but also in computing time. The set of four satisfiable instances[3]

[2] http://www.satlib.org/benchm.html

[3] In Table 1 a $\star$ indicates a satisfiable formula.

Table 1. Comparative results between R+GA and GA.

			R			R+GA			GA	
Formulae	**N**	**M**	β_i	β_f	$\mathbf{T}_\beta$	**SC**	**T**	$\mathbf{T}_{\mathbf{Total}}$	**SC**	**T**
aim100-1_6y⋆	100	160	94	37	3.1	160	4.95	8.05	159	16.81
aim100-2_0y⋆	100	200	96	42	5.2	200	4.36	9.56	198	10.11
dubois28	84	224	77	9	1.0	223	2.08	3.08	221	7.25
dubois29	87	232	81	9	1.0	231	1.09	2.09	231	19.89
dubois30	90	240	87	9	1.0	239	13.35	14.35	237	27.81
dubois50	150	400	144	10	2.1	397	1.93	4.03	395	39.94
flat30-1⋆	90	300	85	26	2.2	300	16.61	18.81	299	27.41
flat30-2⋆	90	300	81	24	2.2	300	11.44	13.64	300	17.82
pret60-75	60	160	56	10	1.0	159	1.75	2.75	159	14.67
pret150-40	150	400	143	24	2.1	399	33.63	35.73	396	23.84
pret150-60	150	400	140	27	3.1	399	12.75	15.85	397	21.31
pret150-75	150	400	142	23	3.0	397	35.82	38.82	397	65.76

used in the experiments were solved by R+GA while only one of them could be solved by GA. Better quality solutions for the R+GA is possible thanks to the representation used by the GA ran after the reordering algorithm. This new representation has two very important characteristics: it keeps the gene interaction as low as possible and promotes the creation of building blocks. Smaller computing time for the R+GA approach is possible because the reordering algorithm is able to find good solutions in reasonable time and also because the GA using the new representation takes less CPU time. This can be clearly appreciated in the Figure 3. It shows the convergence process of the two compared algorithms. The *X* axis represents CPU time in seconds used by the algorithms, while the *Y* axis indicates the number of satisfied clauses. It is important to point out that the R+GA curve initiates at the time T_β, which is the CPU time in seconds consumed in the reordering stage.

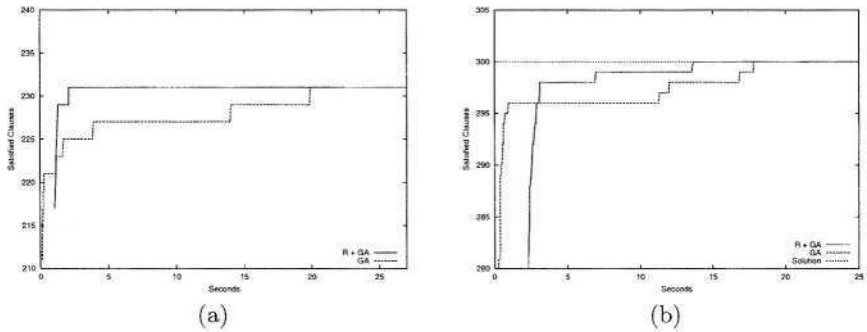


Fig. 3. Average best-so-far curves for R+GA and GA on the problems dubois29 (a) and flat30-2 (b).

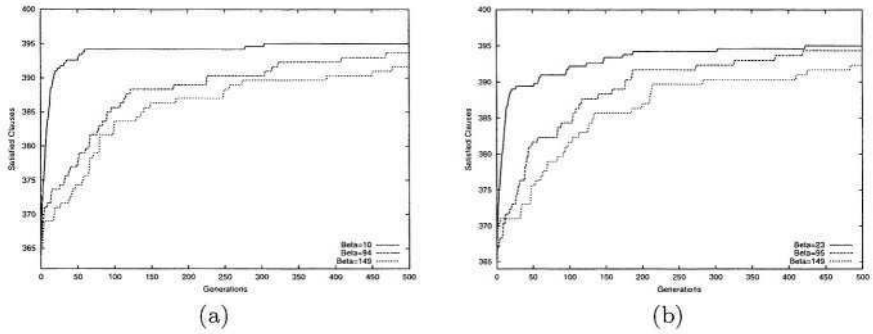


Fig. 4. Average best-so-far curves for a classic GA ran with different bandwidths on the problems dubois50 (a) and Pret l50-75 (b).

The behavior of a classic GA ran on the same problem with different bandwidths of the associated graph (Beta) has been explored. In Figure 4 it can be seen that the performance of the GA is better when the bandwidth is low, i.e. the most related variables are placed in closer positions inside the chromosome.

Next section presents the conclusions of this work.

6 Conclusions

In this paper a reordering algorithm was used to improve the performance of a classic GA when used to solve SAT problems. This combined approach has been compared versus a simple GA using a set of SAT instances and it outperformed the GA in all the tests accomplished. Additionally it has been observed during the experimentation, that as the size of the problem increases, the advantage to use the proposed reordering algorithm also increases. It allows us to conclude that it pays to make a preprocessing of the problems in order to obtain a more suitable representation.

The R+GA method reported in this paper is a preliminary version. Studies are in the way to have a better understanding of its behavior with respect of different classes of SAT instances. Another pending issued is to improve its performance; specially, a better stop criteria.

We think that the R+GA method is very promising and worthy to more research. In particular it will be interesting and important to identify the classes of SAT instances where is appropriate to apply our method and if this reordering technique could be used in other kind of problems which will be solved by using GAs.

References

1. Y. Kambayashi B. Cha, K. Iwama and S. Miyasaki, *Local search algorithms for partial maxsat,* Proceedings of the AAAI 97, July 27-31 1997, pp. 263–268.

2. T. Blickle and L. Thiele, *A mathematical analysis of tournament selection,* Proceedings of the Sixth ICGA, Morgan Kaufmann Publishers, San Francisco, Ca., 1995, pp. 9–16.
3. P.Z. Chinn, J. Chvatalova, A.K. Dewdney, and N.E. Gibbs, *The bandwidth problem for graphs and matrices - a survey,* Journal of Graph Theory **6** (1982), no. 3, 223–254.
4. S. A. Cook, *The complexity of theorem proving procedures,* 3rd Annual ACM Symposium on the Theory of Computing, 1971, pp. 151–158.
5. E. Cutchill and J. McKee, *Reducing the bandwidth of sparse symmetric matrices,* Proceedings 24th National of the ACM (1969), 157–172.
6. Y. Davidor, *Epistasis Variance: A Viewpont of GA-Hardness,* Proceedings of the Second Foundations of Genetic Algorithms Workshop, Morgan Kaufmann, 1991, pp. 23–35.
7. G. Dueck and J. Jeffs, *A heuristic bandwidth reduction algorithm,* Journal of combinatorial mathematics and computers (1995), no. 18, 97–108.
8. Agoston E. Eiben, Jan K. Van Der Hauw, and Jan Van Hemert, *Graph coloring with adaptative evolutionary algorithms,* Journal of Heuristics **4** (1998), no. 1, 25–46.
9. C. Fleurent and J. Ferland, *Object-oriented implementation of heuristic search methods for graph coloring, maximum clique, and satisfiability,* Second DIMACS Challenge, Special Issue, AMS, Providence, Rhode Island, 1996, pp. 619–652.
10. M.R. Garey, R.L. Graham, D.S. Johnson, and D.E. Knuth, *Complexity results for bandwidth minimization,* SIAM Journal of Applied Mathematics 34 (1978), 477–495.
11. N.E. Gibbs, W.G. Poole, and P.K. Stockmeyer, *An algorithm for reducing the bandwidth and profile of a sparse matrix,* SIAM Journal on Numerical Analysis **13** (1976), 235–251.
12. David E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning,* Addison-Wesley Publishing Company, Inc., 1989.
13. J. Gottlieb and N. Voss, *Improving the performance of evolutionary algorithms for the satisfiability problem by refining functions,* Proceedings of the Sixth International Conference on Parallel Problem Solving from Nature (Berlin, Germany), Lecture Notes in Computer Science, vol. Volumen 1917, Springer, 2000, pp. 621–630.
14. Jens Gottlieb, Elena Marchiori, and Claudio Rossi, *Evolutionary algorithms for the satisfiability problem,* Evolutionary Computation **10** (2002), no. 1, 35–50.
15. E.M. Gurari and I.H. Sudborough, *Improved dynamic programming algorithms for bandwidth minimization and the min-cut linear arrangement problem,* Journal of Algorithms 5 (1984), 531–546.
16. J. Haralambides, F. Makedon, and B. Monien, *An aproximation algorithm for caterpillars,* Journal of Mathematical Systems Theory **24** (1991), 169–177.
17. F. Harary, *Theory of graphs and its aplications,* Czechoslovak Academy of Science, Prague, 1967, M. Fiedler.
18. L.H. Harper, *Optimal assignment of numbers to vertices,* Journal of SIAM **12** (1964), 131–135.
19. J. Holland, *Adaptation in natural and artificial systems,* Ann Arbor: The University of Michigan Press, 1975.
20. Kenneth A. De Jong and William M. Spears, *Using genetic algorithms to solve NP-complete problems,* Proceedings of the Third ICGA (Fairfax, Virginia), 1989, pp. 124–132.

21. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, *Optimization by simulated annealing,* Science **220** (1983), 671–680.
22. E. Kosko, *Matrix inversion by partitioning - Part 2,* The Aeronautical Quarterly (1956), no. 8, 157.
23. R.R. Livesley, *The analysis of large structural systems,* Computer Journal **3** (1960), no. 1, 34–39.
24. E. Marchiori and C. Rossi, *A flipping genetic algorithm for hard 3-SAT problems,* Proceedings of Genetic and Evolutionary Computation Conference (San Francisco, California), Morgan Kaufmann, 1999, pp. 393–400.
25. C.H. Papadimitriou, *The NP-Completeness of the bandwidth minimization problem,* Journal on Computing **16** (1976), 263–270.
26. S. Rana and D. Whitley, *Genetic algorithm behavior in the MAXSAT domain,* Proceedings of the Fifth International Conference on Parallel Problem Solving from Nature (Berlin, Germany), Lecture Notes in Computer Science, vol. Volume 1498, Springer, Berlin, Germany, 1998, pp. 785–794.
27. B. Mazure; L. Sais and E. Gregoire, *Tabu search for SAT,* Proc. National Conference on Artificial Intelligence (AAAI-97), 1997, pp. 281–285.
28. Jim Smith, *On Appropriate Adaptation Levels for the Learning of Gene Linkage,* Journal of Genetic Programming and Evolvable Machines **3** (2002), 129–155.
29. William M. Spears, *Simulated Annealing for Hard Satisfiability Problems,* Tech. Report AIC-93-015, AI Center, Naval Research Laboratory, Washington, DC 20375, 1993.
30. Jose Torres-Jimenez and Eduardo Rodriguez-Tello, *A new measure for the bandwidth minimization problem,* Proceedings of the IBERAMIA-SBIA 2000, Number 1952 in LNAI (Antibaia SP, Brazil), Springer-Verlag, November 2000, pp. 477–486.

Designing Competent Mutation Operators Via Probabilistic Model Building of Neighborhoods

Kumara Sastry[1,2] and David E. Goldberg[1,3]

[1] Illinois Genetic Algorithms Laboratory (IlliGAL)
[2] Department of Material Science & Engineering
[3] Department of General Engineering
University of Illinois at Urbana-Champaign, Urbana IL 61801
{ksastry,deg}@uiuc.edu

Abstract. This paper presents a *competent* selectomutative genetic algorithm (GA), that adapts linkage and solves hard problems quickly, reliably, and accurately. A probabilistic model building process is used to automatically identify key building blocks (BBs) of the search problem. The mutation operator uses the probabilistic model of linkage groups to find the best among competing building blocks. The competent selectomutative GA successfully solves additively separable problems of bounded difficulty, requiring only subquadratic number of function evaluations. The results show that for additively separable problems the probabilistic model building BB-wise mutation scales as $\mathcal{O}(2^k m^{1.5})$, and requires $\mathcal{O}(\sqrt{k}\log m)$ less function evaluations than its selectorecombinative counterpart, confirming theoretical results reported elsewhere [1].

1 Introduction

One of the key challenges in designing an effective mutation operator is ensuring that it searches in the correct neighborhood. Existing mutation operators usually search in the local neighborhood of an individual, without taking into account the *global* neighborhood information. Recently, it was shown that a selectomutative algorithm that performs hillclimbing in building-block space can successfully solve boundedly-difficult problems in polynomial time as opposed to exponential time of simple mutation operators [1]. The results also showed that for additively separable search problems with deterministic fitness functions, building-block-wise mutation provided significant speed-up over recombination. The analysis assumed that both mutation and crossover operators had linkage information.

While several *competent* recombination operators that adapt linkage have been successfully and systematically designed, little attention has been paid to the development of competent mutation operators. Similarly, in local-search literature, while the importance of using a good neighborhood operator is often highlighted [2,3], there are no systematic methods for designing neighborhood operators that can solve a broad class of bounding problems. Therefore, the objective of this paper is to propose a methodology of automatically discovering *global* neighborhood information (which was *assumed* to be known *a priori* in [1])) from a population of sampled candidate solutions.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 114–125, 2004.

This paper systematically designs a *competent* mutation operator that adapts linkage and performs local search in the building-block space. The important substructures are automatically identified using probabilistic models developed via machine-learning techniques. Specifically, we use the probabilistic-model-building procedure of extended compact genetic algorithm (eCGA) [4] to identify the linkage groups (or *global* neighborhood) of a search problem.

This paper is organized as follows. The next section gives a brief literature review, followed by a discussion on the relation between neighborhood operators and linkage groups. We then introduce eCGA followed by a description of the BB-wise mutation algorithm and provide empirical results on the scalability of each algorithm. Finally, we outline future research directions followed by conclusions.

2 Literature Review

One of the key challenges in the area of genetic and evolutionary algorithms is the systematic design of genetic operators with demonstrated scalability. One such design-decomposition theory for developing effective GA designs has been proposed [5,6,7]. Based on the design-decomposition theory many competent GAs have been designed, which can be broadly classified into three categories:

Perturbation techniques include the messy GA [8], fast messy GA [9], gene expression messy GA [10], linkage identification by nonlinearity check GA, and linkage identification by monotonicity detection GA [11], dependency structure matrix driven genetic algorithm (DSMDGA) [12], and linkage identification by limited probing [13].

Linkage adaptation techniques such as linkage learning GA [14,15].

Probabilistic model building techniques [16,17] such as population-based incremental learning [18], the bivariate marginal distribution algorithm [19], the extended compact GA (eCGA) [4], iterated distribution estimation algorithm [20], Bayesian optimization algorithm (BOA) [21].

While the above techniques are selectorecombinative GAs, little attention has been paid to the systematic design of selectomutative GAs that utilize linkage (or neighborhood) information. Recently, we demonstrated that a mutation operator that performs local search in building-block neighborhood takes problems that were intractable by a fixed mutation operator and renders them *tractable* [1], requiring only polynomial number of function evaluations. In the study, we assumed that linkage information was known, which is not usually the case. Therefore, in this paper, we present a technique to identify the neighborhood information as a probabilistic model using a population of sampled solutions.

3 Neighborhood Operators and Linkage Groups

In local search literature, researchers have often recognized the importance of a good neighborhood operator in determining the effectiveness of a search algorithm [2,3,22,23,24]. A neighborhood operator that is capable of not only efficiently sampling the local neighborhood, but also able to jump to new less

sampled neighborhoods, has often yielded good results [23,24]. Oftentimes the neighborhood operators are designed for a particular search problem on an ad-hoc basis. There are no systematic procedures or analytical guidance for designing a good neighborhood operator that can work on a broad class of search problems.

In genetic algorithms, while significant attention is paid to the design of recombination operators, little or no attention is paid to the design of mutation operators. In GAs, mutation is usually a secondary search operator which performs random walk *locally* around a solution. On the other hand, in evolution strategies (ES) [25], in which mutation is the primary search operator, significant attention has been paid to the development of mutation operators. Several mutation operators, including adaptive techniques, have been proposed [25,26,27, 28,29]. While the mutation operators used in ES are powerful search operators, the neighborhood information is still *local* around a single or few solutions.

However, for solving boundedly difficult GA-hard problems, local neighborhood information is not sufficient, and a mutation operator which uses local neighborhoods requires exponential time [30]. Therefore, we utilize *machine-learning* tools and a population of candidate solutions of the search problem for automatically building *global* neighborhood (or linkage) information into the mutation operator. Unlike, adaptive mutation techniques in ES, which usually have local neighborhood information adapted over time, our method leads to a more global induction of the neighborhood. Specifically, we build probabilistic models of *global* neighborhood information by sampling candidate solutions to the search problem. The mutation operator proposed in this paper, utilizes the *global* neighborhood information to search among competing sub-solutions.

The procedure used to build the neighborhood information is based on the model-building procedure of eCGA, which is explained in the following section.

4 Extended Compact Genetic Algorithm (eCGA)

The extended compact GA proposed by Harik [4] is based on a key idea that the choice of a good probability distribution is equivalent to linkage learning. The measure of a good distribution is quantified based on minimum description length (MDL) models. The key concept behind MDL models is that all things being equal, simpler distributions are better than more complex ones. The MDL restriction reformulates the problem of finding a good distribution as an optimization problem that minimizes both the model complexity as well as model inaccuracy. The probability distribution used in eCGA is a class of probability models known as marginal product models (MPMs). MPMs are formed as a product of marginal distributions on a partition of the genes. MPMs also facilitate a direct linkage map with each partition separating tightly linked genes. For example, the following MPM, `[1,3] [2] [4]` , for a four-bit problem represents that the 1^{st} and 3^{rd} genes are linked and 2^{nd} and 4^{th} genes are independent.

The eCGA can be algorithmically outlined as follows:

1. Initialization: The population is usually initialized with random individuals.

2. Evaluate the fitness value of the individuals
3. Selection: The eCGA uses an s-wise tournament selection [8].
4. Build the probabilistic model: In eCGA, both the structure and the parameters of the model are searched. A greedy search heuristic is used to find an optimal model of the selected individuals in the population.
5. Create new individuals: In eCGA, new individuals are created by sampling the probabilistic model.
6. Replace the parental population with the offspring population.
7. Repeat steps 2–6 until some convergence criteria are met.

Two things need further explanation, one is the identification of MPM using MDL and the other is the creation of a new population based on MPM. The identification of MPM in every generation is formulated as a constrained optimization problem, that minimizes the sum of the model complexity, C_m, which represents the complexity of the model and compressed population complexity, C_p, which represents the inaccuracy of the model.

In essence, the model complexity, C_m, quantifies the model representation size in terms of number of bits required to store all the marginal probabilities. Let, a given problem of size ℓ with binary alphabets, have m partitions with k_i genes in the i^{th} partition, such that $\sum_{i=1}^{m} k_i = \ell$. Then each partition i requires $2^k - 1$ independent frequencies to completely define its marginal distribution. Furthermore, each frequency is of size $\log_2(n)$, where n is the population size. Therefore, the model complexity (or the representation size), C_m, is given by

$$C_m = \log_2(n) \sum_{i=1}^{m} \left(2^{k_i} - 1\right). \tag{1}$$

The compressed population complexity, C_p, quantifies the data compression in terms of the entropy of the marginal distribution over all partitions. Therefore, C_p is evaluated as

$$C_p = n \sum_{i=1}^{m} \sum_{j=1}^{2^{k_i}} -p_{ij} \log_2 \left(p_{ij}\right) \tag{2}$$

where p_{ij} is the frequency of the j^{th} gene sequence of the genes belonging to the i^{th} partition. In other words, $p_{ij} = N_{ij}/n$, where N_{ij} is the number of chromosomes in the population (after selection) possessing bit-sequence $j \in [1, 2^{k_i}]$ [1]for i^{th} partition.

The following greedy search heuristic is used to find an optimal model:

1. Assume each variable is independent of each other.
2. Compute the model complexity and population complexity values of the current model.
3. Consider all possible $\frac{1}{2}\ell(\ell - 1)$ merges of two variables.
4. Evaluate the model and compressed population complexity values for each model structure.

[1] Note that a BB of length k has 2^k possible sequences where the first sequence denotes be $00\cdots0$ and the last sequence $11\cdots1$

5. Select the merged model with lowest combined complexity.
6. If the combined complexity of the best merged model is better than the combined complexity of the model evaluated in step 2., replace it with the best merged model and go to step 2.
7. If the combined complexity of the best merged model is less than or equal to the combined complexity, the model cannot be improved and the model of step 2. is the probabilistic model of the current generation.

The offspring population are generated by randomly choosing subsets from the current individuals according to the probabilities of the subsets as calculated in the probabilistic model.

Analytical models have been developed for predicting the population-sizing and the scalability of probabilistic model building GAs [31,32]. The models predict that the population size required to solve a problem with m building blocks of size k with a failure rate of $\alpha = 1/m$ is given by

$$n \propto 2^k \left(\frac{\sigma_{BB}}{d}\right) m \log m, \tag{3}$$

and the number of function evaluations is given by

$$n_{fe} \propto \left(\frac{\sigma_{BB}}{d}\right) \sqrt{k} \cdot 2^k m^{1.5} \log m, \tag{4}$$

where σ_{BB} is fitness-variance of a BB and d is the signal difference between competing BBs [6].

Equations 3 and 4 are verified with empirical results for the m k-deceptive function [33] with *loose linkage* in Figures 1(a) and 1(b). By loose linkage we mean that the components of a BB are located far apart from each other in the chromosome. Fixed recombination operators such as one-point crossover or uniform crossover need exponential time to solve such loosely-linked deceptive problems [34]. The results show that eCGA automatically identifies the linkage groups and solve additively separable GA-hard problems, requiring only sub-quadratic number of function evaluations.

In obtaining the empirical results, we use a tournament selection with tournament size of 8. An eCGA run is terminated when all the individuals in the population converge to the same fitness value. The average number of BBs correctly converged are computed over 30 independent runs. The minimum population size required such that $m-1$ BBs converge to the correct value is determined by a bisection method [35]. The results of population-size is averaged over 30 such bisection runs, while the results for the function-evaluation ratio is averaged over 900 independent eCGA runs.

5 Probabilistic Model Building BB-Wise Mutation

As explained in the previous section, eCGA builds marginal product models that yields a direct mapping of linkage groups among successful individuals. Therefore, for BB identification purposes, we use the model-building procedure

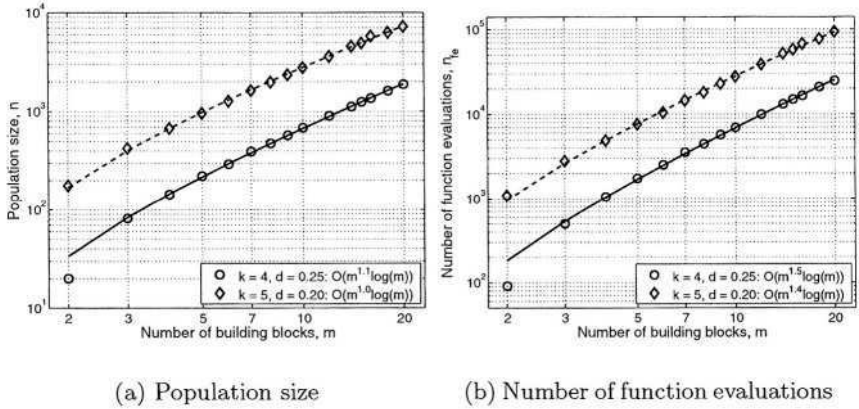


Fig. 1. Population size (Equation 3) and number of function evaluations (Equation 4) required by eCGA to successfully solve m k-Trap function. The results are averaged over 900 eCGA runs for the function evaluations and 30 bisection runs for the population size. The relative deviation for the empirical results is less than 1%. The results show that the population size scales as $\mathcal{O}(2^k m \log m)$ and the number of function evaluations scales as $\mathcal{O}(2^k m^{1.5} \log m)$.

of eCGA. Once the linkage-groups are identified, we use an *enumerative BB-wise mutation* operator as used elsewhere [1]. For example, if model builder identifies m BBs of size k each, the BB-wise algorithm will select the best BB out of 2^k possible ones in each of the m partition. The detailed procedure of the competent selectomutative GA is given in the following:

1. Initialize the population with random individuals and evaluate their fitness.
2. Selection: This procedure is similar to that of eCGA as described in Section 4.
3. Build the probabilistic model as explained in Section 4 to obtain linkage-group information.
4. Use the best individual from the population for BB-wise mutation.
5. Consider the first non-mutated BB. Here the BB order is chosen arbitrarily from left-to-right, however, different schemes can be—or may required to be—chosen to decide the order of BBs. For example, BB partitions that contain most *active* variables might be mutated before those that contain less active variables.
6. Create $2^k - 1$ unique individuals with all possible schemata in the chosen BB partition. Note that the schemata in other partitions are the same as the original individual (from step 2).
7. Evaluate all $2^k - 1$ individuals and retain the best for mutation of BBs in other partitions.
8. Repeat steps 5–7 till BBs of all the partitions have been mutated.

Steps 1–3 are identical to the ones used in eCGA (Section 4) and steps 4–8 are similar to the BB-wise mutation operator used elsewhere [1].

Note that the performance of the BB-wise mutation can be slightly improved by using a greedy heuristic to search for the best among competing BBs, however, as shown later, the scalability of the probabilistic model building BB-wise mutation operator is determined by the population-size required to accurately identify the building blocks.

It should also be noted that while eCGA can only build linkage groups with non-overlapping variables, the mutation procedure can be easily used with other linkage identification techniques that can handle overlapping BBs such as BOA [21] or DSMDGA [12]. However, since the effect of overlapping interactions between variables is similar to that of an exogenous noise [7], crossover is likely to be more effective than mutation [1].

Finally, we perform linkage identification only once in the initial generation. This offline linkage identification works well on problems with BBs of nearly equal salience. However, for problems with BBs of non-uniform salience, we would have to perform linkage identification and update BB information in regular intervals. Furthermore, it might be more efficient to utilize both BB-wise mutation and eCGA model sampling simultaneously or sequentially along the lines of *hybridization* [36,37,38] and *time-continuation* [39,40] techniques.

However, the objective of this paper is to couple linkage identification with a mutation operator that performs local search in the BB neighborhood and to verify its effectiveness in solving boundedly difficult additively separable problems. Moreover, the aforementioned enhancements can be designed on the proposed competent selectomutative GA.

5.1 Scalability of the BB-Wise Mutation

The scalability of the selectomutative GA depends on two factors: (1) The population size required to build accurate probabilistic models of the linkage groups, and (2) the total number of evaluations performed by the BB-wise mutation operator to find optimum BBs in all the partitions.

Pelikan and Sastry [32] developed facetwise models for predicting the critical and maximum population-size required to correctly identify good interactions among variables. They showed that the minimum population size scales as

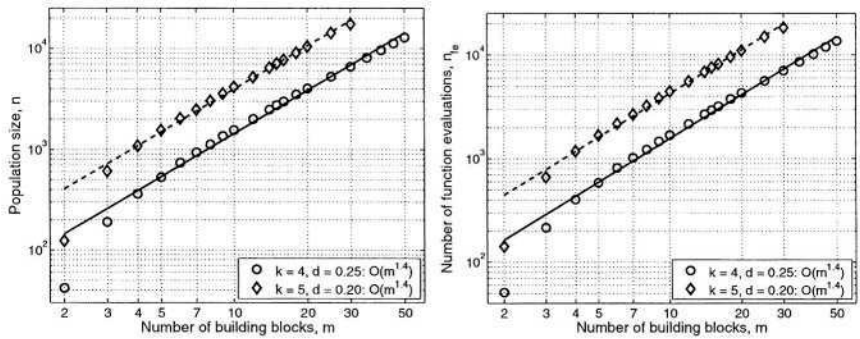
$$n_{\min} = \mathcal{O}\left(2^k m^{1.05}\right), \tag{5}$$

and the maximum population size which avoids discovery of false dependencies between independent variables is given by

$$n_{\max} = \mathcal{O}\left(2^k m^{2.1}\right), \tag{6}$$

In other words, to avoid incorrect identification of BBs, the population size should be less than $n_{\max}$. Since we require that *all* the BBs be correctly identified in the first generation itself, the population size required should be greater than $n_{\min}$, but less than $n_{\max}$. That is,

$$\mathcal{O}\left(2^k m^{1.05}\right) \leq n \leq \mathcal{O}\left(2^k m^{2.1}\right). \tag{7}$$



(a) Population size (b) Number of function evaluations

Fig. 2. Population size (Equation 7 and the number of function evaluations (Equation 10) required by BB-wise mutation for solving m k-Trap function. The results are averaged over 900 runs for the function evaluations and 30 bisection runs for the population size. The relative deviation for the empirical results is less than 0.2%. The population size and the number of function evaluations both scale as $\mathcal{O}(2^k m^{1.5})$.

Since the model building is performed only once, the total number of function evaluations scales as the population size. That is,

$$\mathcal{O}\left(2^k m^{1.05}\right) \leq n_{\text{fe},1} \leq \mathcal{O}\left(2^k m^{2.1}\right). \tag{8}$$

During BB-wise mutation, we evaluate $2^k - 1$ individuals for determining the best BBs in each of the m partitions. Therefore, the total number of function evaluations used during BB-wise mutation is

$$n_{\text{fe},2} = \left(2^k - 1\right) m = \mathcal{O}\left(2^k m\right). \tag{9}$$

From Equations 8 and 9, the total number of function evaluations scales as

$$\mathcal{O}\left(2^k m^{1.05}\right) \leq n_{\text{fe}} \leq \mathcal{O}\left(2^k m^{2.1}\right). \tag{10}$$

We now empirically verify the scale-up of the population size and the number of function evaluations required for successfully solving the m k-trap problem with loose linkage in Figures 2(a) and 2(b), respectively. In contrast to fixed mutation operators which require $\mathcal{O}(m^k \log m)$ (exponential) number of function evaluations to solve additively separable GA-hard problems [30], the proposed eCGA-based BB-wise mutation operator that automatically identifies the linkage groups requires only $\mathcal{O}(2^k m^{1.5})$ (polynomial) number of evaluations.

6 eCGA vs. Building-Block-Wise Mutation

The previous two sections demonstrated the scalability of eCGA and the competent selectomutative GA. In this section, we analyze the relative computational costs of using eCGA or the BB-wise mutation algorithm for successfully solving additively separable problems of bounded difficulty.

The results from the above sections (Equations 4 and 10) indicate that while the scalability of eCGA is $\mathcal{O}\left(2^k\sqrt{k}m^{1.5}\log m\right)$, the BB-wise mutation scales as $\mathcal{O}\left(2^k m^{1.5}\right)$. Therefore, the probabilistic model building BB-wise mutation operator is $\mathcal{O}\left(\sqrt{k}\log m\right)$ faster than eCGA in solving boundedly difficult additively separable problems. That is, the speed-up—which is defined as the ratio of number of function evaluations required by eCGA to that required by the selectomutative GA—is given by

$$\eta = \frac{n_{\text{fe}}(\text{eCGA})}{n_{\text{fe}}(\text{BBwise Mutation})} = \mathcal{O}\left(\sqrt{k}\log m\right). \tag{11}$$

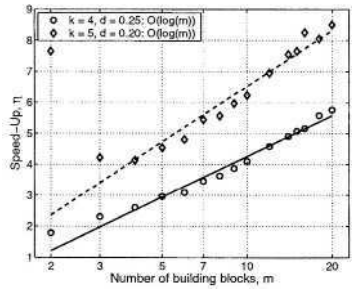


Fig. 3. Empirical verification of the speed-up (Equation 11) obtained by using the probabilistic model building BB-wise mutation over eCGA for the m k-Trap function. The results show that the speed-up scales as $\mathcal{O}(\sqrt{k}\log m)$.

Empirical results shown in Figure 3 agrees with the above equation. The results show that the probabilistic model building BB-wise mutation is $\mathcal{O}(\sqrt{k}m)$ times faster than the extended compact GA. The results are also in agreement with the analytical results derived for an *ideal* BB-wise operator [1].

7 Future Work

We demonstrated the potential of inducting *global* neighborhood information into mutation operations via the automatic discovery of linkage groups by probabilistic model-building techniques. The results are very encouraging and warrants further research in one of more of the following avenues:

Hybridization of competent crossover and mutation: While we considered a bounding case of crossover vs. mutation, it is likely to be more efficient to use an efficient hybrid of competent crossover and mutation operators. For example, we can consider a hybrid GA with *oscillating* populations. A large population is used to gather linkage information and used for crossover, while a small population is used for searching in BB neighborhood.

Problems with overlapping building blocks: While this paper considered problems with non-overlapping building blocks, many problems have different building blocks that share common components and such problems have to be analyzed. Since the effect of overlapping variable interactions is similar

to that of exogenous noise [7], based on our recent analysis [1], a crossover is likely to be more useful than mutation.

Problems with non-uniform BB salience: In this paper, we considered additively separable problems with uniform sub-solution salience. Unlike uniformly-scaled problems, in non-uniformly scaled problems BBs are identified sequentially over time. Therefore, in such cases, we would need to regularly update the BB information and develop theory to predict the updating schedule.

Hierarchical problems: One of the important class of nearly decomposable problems is hierarchical problems, in which the building-block interactions are present at more than a single level. Further investigation is necessary to analyze the performance of BB-wise mutation on hierarchical problems.

8 Summary and Conclusions

In this paper, we have introduced a systematic procedure for the automatic induction of *global* neighborhood information into the mutation operator via the discovery of linkage groups of a problem. We used probabilistic model building techniques to develop a probabilistic model of linkage information of a search problem. The BB-wise mutation operators uses the linkage (or neighborhood) information to perform local search among competing sub-solutions.

We derived an analytical bound and empirically verified the scalability of the competent mutation operator on boundedly-difficult additively separable problems. The results showed that the BB-wise mutation operator successfully solves GA-hard problems, requiring only subquadratic number of function evaluations. That is, for an additively separable problem with m BBs of size k each, the number of function evaluations scales as $\mathcal{O}(2^k m^{1.5})$. We also compared the probabilistic model-building mutation with probabilistic model-building crossover head to head. For deterministic additively separable problems, we showed that BB-wise mutation provides significant advantage over crossover. The results show that the speed-up of using BB-wise mutation over crossover on deterministic problems is $\mathcal{O}(\sqrt{k}\log m)$, which is in agreement with analytical results [1].

Acknowledgments. This work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant F49620-03-1-0129, the National Science Foundation under ITR grant DMR-99-76550, and ITR grant DMR-0121695, and the Dept. of Energy under grant DEFG02-91ER45439.

References

1. Sastry, K., Goldberg, D.: Let's get ready to rumble: Crossover versus mutation head to head. Proceedings of the Genetic and Evolutionary Computation Conference (2004) (To appear) (Also IlliGAL Report No. 2004005).
2. Barnes, J.W., Dimova, B., Dokov, S.P.: The theory of elementary landscapes. Applied Mathematical Letters **16** (2003) 337–343

3. Watson, J.P.: Empirical Modeling and Analysis of Local Search Algorithms For The Job-Shop Scheduling Problem. PhD thesis, Colorado State University, Fort Collins, CO (2003)
4. Harik, G.: Linkage learning via probabilistic modeling in the ECGA. IlliGAL Report No. 99010, University of Illinois at Urbana-Champaign, Urbana, IL (1999)
5. Goldberg, D.E.: Theory tutorial (1991) (Tutorial presented with G. Liepens at the 1991 International Conference on Genetic Algorithms, La Jolla, CA).
6. Goldberg, D.E., Deb, K., Clark, J.H.: Genetic algorithms, noise, and the sizing of populations. Complex Systems **6** (1992) 333–362 (Also IlliGAL Report No. 91010).
7. Goldberg, D.E.: Design of innovation: Lessons from and for competent genetic algorithms. Kluwer Academic Publishers, Boston, MA (2002)
8. Goldberg, D.E., Korb, B., Deb, K.: Messy genetic algorithms: Motivation, analysis, and first results. Complex Systems **3** (1989) 493–530 (IlliGAL Rep. No. 89003).
9. Goldberg, D.E., Deb, K., Kargupta, H., Harik, G.: Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. Proceedings of the International Conference on Genetic Algorithms (1993) 56–64 (IlliGAL Rep. 93004).
10. Kargupta, H.: The gene expression messy genetic algorithm. Proceedings of the International Conference on Evolutionary Computation (1996) 814–819
11. Munetomo, M., Goldberg, D.E.: Linkage identification by non-monotonicity detection for overlapping functions. Evolutionary Computation **7** (1999) 377–398
12. Yu, T.L., Goldberg, D.E., Yassine, A., Chen, Y.P.: A genetic algorithm design inspired by organizational theory: Pilot study of a dependency structure matrix driven genetic algorithm. Artificial Neural Networks in Engineering (2003) 327–332 (Also IlliGAL Report No. 2003007).
13. Heckendorn, R.B., Wright, A.H.: Efficient linkage discovery by limited probing. Proceedings of the Genetic and Evolutionary Computation Conference (2003) 1003–1014
14. Harik, G., Goldberg, D.E.: Learning linkage. Foundations of Genetic Algorithms **4** (1997) 247–262 (Also IlliGAL Report No. 96006).
15. Chen, Y.P., Goldberg, D.E.: Introducing start expression genes to the linkage learning genetic algorithm. Parallel Problem Solving from Nature **7** (2002) 351–360 (Also IlliGAL Report No. 2002007).
16. Pelikan, M., Lobo, F., Goldberg, D.E.: A survey of optimization by building and using probabilistic models. Computational Optimization and Applications **21** (2002) 5–20 (Also IlliGAL Report No. 99018).
17. Larrañaga, P., Lozano, J.A., eds.: Estimation of Distribution Algorithms. Kluwer Academic Publishers, Boston, MA (2002)
18. Baluja, S.: Population-based incremental learning: A method of integrating genetic search based function optimization and competitive learning. Technical Report CMU-CS-94-163, Carnegie Mellon University (1994)
19. Pelikan, M., Mühlenbein, H.: The bivariate marginal distribution algorithm. In Roy, R., Furuhashi, T., Chawdhry, P.K., eds.: Advances in Soft Computing - Engineering Design and Manufacturing, London, Springer-Verlag (1999) 521–535
20. Bosman, P., Thierens, D.: Linkage information processing in distribution estimation algorithms. Proceedings of the Genetic and Evolutionary Computation Conference (1999) 60–67
21. Pelikan, M., Goldberg, D.E., Cantú-Paz, E.: Linkage learning, estimation distribution, and Bayesian networks. Evolutionary Computation **8** (2000) 314–341 (Also IlliGAL Report No. 98013).

22. Vaughan, D., Jacobson, S., Armstrong, D.: A new neighborhood function for discrete manufacturing process design optimization using generalized hill climbing algorithms. ASME Journal of Mechanical Design **122** (2000) 164–171
23. Armstrong, D., Jacobson, S.: Polynomial transformations and data independent neighborhood functions. Discrete Applied Mathematics (2004) (Submitted).
24. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers, Boston, MA (1997)
25. Rechenberg, I.: Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution. Frommann-Holzboog, Stuttgart (1973)
26. Schwefel, H.P.: Numerische optimierung von computer-modellen mittels der evolutionsstrategie. Interdisciplinary Systems Research **26** (1977)
27. Bäck, T.: Evolutionary Algorithms in Theory and Practice. Oxford University Press, New York (1996)
28. Beyer, H.G.: Toward a theory of evolution strategies: Self-adaptation. Evolutionary Computation **3** (1996) 311–347
29. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation **9** (2001) 159–195
30. Mühlenbein, H.: How genetic algorithms really work: Mutation and hillclimbing. Parallel Problem Solving from Nature II (1992) 15–26
31. Pelikan, M., Goldberg, D.E., Cantú-Paz, E.: Bayesian optimization algorithm, population sizing, and time to convergence. Proceedings of the Genetic and Evolutionary Computation Conference (2000) 275–282 (Also IlliGAL Report No. 2000001).
32. Pelikan, M., Sastry, K., Goldberg, D.E.: Scalability of the Bayesian optimization algorithm. International Journal of Approximate Reasoning **31** (2003) 221–258 (Also IlliGAL Report No. 2001029).
33. Deb, K., Goldberg, D.E.: Analyzing deception in trap functions. Foundations of Genetic Algorithms **2** (1993) 93–108 (Also IlliGAL Report No. 91009).
34. Thierens, D., Goldberg, D.E.: Mixing in genetic algorithms. Proceedings of the International Conference On Genetic Algorithms (1993) 38–45
35. Sastry, K.: Evaluation-relaxation schemes for genetic and evolutionary algorithms. Master's thesis, University of Illinois at Urbana-Champaign, General Engineering Department, Urbana, IL (2001) (Also IlliGAL Report No. 2002004).
36. Goldberg, D.E., Voessner, S.: Optimizing global-local search hybrids. Proceedings of the Genetic and Evolutionary Computation Conference (1999) 220–228 (Also IlliGAL Report No. 99001).
37. Sinha, A., Goldberg, D.E.: A survey of hybrid genetic and evolutionary algorithms. IlliGAL Report No. 2003004, University of Illinois at Urbana-Champaign, General Engineering Department, Urbana, IL (2003)
38. Sinha, A.: Designing efficient genetic and evolutionary algorithm hybrids. Master's thesis, University of Illinois at Urbana-Champaign, General Engineering Department, Urbana, IL (2003) (Also IlliGAL Report No. 2003020).
39. Goldberg, D.E.: Using time efficiently: Genetic-evolutionary algorithms and the continuation problem. Proceedings of the Genetic and Evolutionary Computation Conference (1999) 212–219 (Also IlliGAL Report No. 99002).
40. Srivastava, R.: Time continuation in genetic algorithms. Master's thesis, University of Illinois at Urbana-Champaign, General Engineering Department, Urbana, IL (2002) (Also IlliGAL Report No. 2001021).

Let's Get Ready to Rumble: Crossover Versus Mutation Head to Head

Kumara Sastry[1,2] and David E. Goldberg[1,3]

[1] Illinois Genetic Algorithms Laboratory (IlliGAL)
[2] Department of Material Science & Engineering
[3] Department of General Engineering
University of Illinois at Urbana-Champaign, Urbana IL 61801
{ksastry,deg}@uiuc.edu

Abstract. This paper analyzes the relative advantages between crossover and mutation on a class of deterministic and stochastic additively separable problems. This study assumes that the recombination and mutation operators have the knowledge of the building blocks (BBs) and effectively exchange or search among competing BBs. Facetwise models of convergence time and population sizing have been used to determine the scalability of each algorithm. The analysis shows that for additively separable deterministic problems, the BB-wise mutation is more efficient than crossover, while the crossover outperforms the mutation on additively separable problems perturbed with additive Gaussian noise. The results show that the speed-up of using BB-wise mutation on deterministic problems is $\mathcal{O}(\sqrt{k}\log m)$, where k is the BB size, and m is the number of BBs. Likewise, the speed-up of using crossover on stochastic problems with fixed noise variance is $\mathcal{O}(m\sqrt{k}/\log m)$.

1 Introduction

Great debate between crossover and mutation has consumed much ink and many trees over the years. When mutation works it is lightening quick and uses small or non-extent populations. Crossover when it works, seems to be able to tackle more complex problems, but getting the population size and other parameters set is a challenge. Comparisons between the two are usually written by a researcher with an axe to grind. Comparisons are usually empirical, the basis for comparison is implicitly or explicitly unfair, and theory is non-existent. Wouldn't it be nice to compare our two favorite genetic operators on a fair basis in an interesting class of problems and let them slug it out head to head.

That's what we do here. Assuming that both the recombination and mutation operators possess linkage (or neighborhood) knowledge, we pit them against each other for solving boundedly difficult additively separable problems with and without the presence of additive exogenous noise. We use a recombination operator that exchanges building blocks (BBs) without disrupting them and a mutation operator that performs local search among competing building-block neighborhoods. The motivation for this study also comes from recent local-search

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 126–137, 2004.

literature, where authors have highlighted the importance of using a good *neighborhood* operator [1,2]. However, a systematic method of designing a good neighborhood operator for a class of search problems is still an open question. We investigate whether using a neighborhood operator that searches among competing BBs of a problem would be advantageous and if so under what circumstances.

This paper is organized as follows. The next section gives a brief review of related literature. We provide an outline of the crossover-based and mutation-based genetic algorithms (GAs) in Section 3. Facetwise models are developed to determine the scalability of the crossover and the BB-wise mutation-based GAs for deterministic fitness functions in Section 4 and for noisy fitness functions in Section 5. Finally, we discuss future research directions followed by conclusions.

2 Literature Review

Over the last few decades many researchers have empirically and theoretically studied where GAs excel. An exhaustive literature review is out of the scope of this paper, and therefore we present a brief review of related theoretical studies.

Several authors have analyzed the scalability of a mutation based hillclimber and compared it to scalability of different forms of genetic algorithms, such as breeder genetic algorithm [3,4], an ideal genetic algorithm [5], and a genetic algorithm with *culling* [6]. Spears [7] compared uniform crossover and bit-wise mutation in terms of schemata disruption and construction probabilities. He suggested that crossover had higher probability of maintaining and creating higher-order schemata than mutation. Goldberg [8] gave a theoretical analysis of deciding between a single run with a large population GA and multiple runs with several small population GAs, under the constraint of fixed computational cost. He showed that for uniformly-scaled problems a single run of large population GA was advantageous, while for exponentially-scaled problems small population GAs with multiple restarts were better. Srivastava and Goldberg [9, 10] empirically verified and analytically enhanced the *time-continuation* theory put forth by Goldberg [8]. Recently, Cantú-Paz and Goldberg [11] investigated scenarios under which multiple runs of a GA are better than a single GA run. For an exhaustive review of studies on the advantages/disadvantages of multiple populations both under serial and parallel GAs over a single large-population GA, the reader is referred elsewhere [10,12,13,14] and to the references therein.

Many of the related studies [3,4,5,6,7,8,9,10,11] assumed fixed genetic operators, with no knowledge of building-block structure. Furthermore, some of the aforementioned studies also considered problems where building-block identification and mixing are not critical for obtaining the optimal solution. In this paper, we assume that the recombination and mutation operators have linkage (or neighborhood) knowledge and consider test problems where building-block identification and exchange are critical to GA success. While the linkage information is usually unknown for a given search problem, a variety of linkage identification methods can be used to design the operators (see Goldberg [15], Sastry and Goldberg [16], and references therein).

3 Preliminaries

The objective of this paper is to predict the relative computational costs of a crossover and an ideal-mutation based algorithm for additively separable problems with and without additive Gaussian noise. Before developing models for estimating the computational costs, we briefly describe the algorithms and the assumptions used in the paper.

3.1 Selectorecombinative Genetic Algorithms

We consider a generationwise selectorecombinative GA with non-overlapping populations of fixed size [17,18]. We apply crossover with a probability of 1.0 and do not use any mutation. We assume binary strings of fixed length as the chromosomes. To ease the analytical burden, the selection mechanism assumed throughout the analysis is binary tournament selection [19]. However, the results can be extended to other tournament sizes and other selection methods in a straightforward manner. The recombination method used in the analysis is a uniform building-block-wise crossover [20]. In uniform BB-wise crossover, two parents are randomly selected from the mating pool and their building blocks in each partition are exchanged with a probability of 0.5. Therefore, none of the building blocks are disrupted during a recombination event. The offspring created through crossover entirely replace the parental individuals.

3.2 Building-Block-Wise Mutation Algorithm (BBMA)

In this paper, we consider an *enumerative BB-wise mutation* operator, in which we start with a random individual and evaluate all possible schemas in a given partition. That is, for a building-block of size k, we evaluate all 2^k individuals. The best out of 2^k individuals is chosen as a candidate for mutating BBs of other partitions. In other words, the BBs in different partitions are mutated in a sequential manner. For a problem with m BBs of size k each, the BBMA can be described as follows:

1. Start with a random individual and evaluate it.
2. Consider the first non-mutated BB. Here the BB order is chosen arbitrarily from left-to-right, however, different schemes can be—or may required to be—chosen to decide the order of BBs.
3. Create $2^k - 1$ unique individuals with all possible schemata in the chosen BB partition. Note that the schemata in other partitions are the same as the original individual (from step 2).
4. Evaluate all $2^k - 1$ individuals and retain the best for mutation of BBs in other partitions.
5. Repeat steps 2–4 till BBs of all the partitions have been mutated.

We use an enumerative BB-wise mutation for simplifying the analysis. A greedy BB-wise method can improve the performance of the mutation-based algorithm.

A straightforward Markov process analysis—along the lines of [3,4]—of a greedy BB-wise mutation algorithm indeed shows that the greedy method is on an average better than the enumerative one. However, the analysis also shows that differences between the greedy and enumerative BB-wise mutation approaches are little, especially for moderate-to-large problems. Moreover, the computational costs of an enumerative BB-wise mutation bounds the costs of a greedy BB-wise mutation.

4 Crossover vs. Mutation: Deterministic Fitness Functions

In this section, we analyze the relative computational costs of using a selectorecombinative GA or a BB-wise mutation algorithm for successfully solving deterministic problems of bounded difficulty. The objective of the analysis is to answer whether a population-based selectorecombinative GA is computationally advantageous over a BB-wise-mutation based algorithm. If one algorithm is better than the other, we are also interested in estimating the savings in computational time. Note that unlike earlier studies, we assume that the building-block structure is known to both the crossover and mutation operators.

We begin our analysis with the scalability of selectorecombinative genetic algorithms followed by the scalability of the BB-wise mutation algorithm.

4.1 Scalability of Selectorecombinative GA

Two key factors for predicting the scalability and estimating the computational costs of a genetic algorithm are the convergence time and population sizing. Therefore, in the following subsections we present facetwise models of convergence time and population sizing.

Population-Sizing Model. Goldberg, Deb, & Clark [21] proposed population-sizing models for correctly deciding between competing BBs. They incorporated noise arising from other partitions into their model. However, they assumed that if wrong BBs were chosen in the first generation, the GAs would be unable to recover from the error. Harik, Cantú-Paz, Goldberg, and Miller [22] refined the above model by incorporating cumulative effects of decision making over time rather than in first generation only. Harik et al. [22] modeled the decision making between competing BBs as a gambler's ruin problem. Here we use an approximate form of the gambler's ruin population-sizing model [22]:

$$n = \frac{\sqrt{\pi}}{2} \frac{\sigma_{BB}}{d} 2^k \sqrt{m} \log m, \tag{1}$$

where k is the BB size, m is the number of BBs, d is the size signal between the competing BBs, and σ_{BB} is the fitness variance of a building block. The above equation assumes a failure probability (per BB), $\alpha = 1/m$.

Convergence-Time Model. Mühlenbein and Schlierkamp-Voosen [23] derived a convergence-time model for the breeder GA using the notion of *selection intensity* [24] from population genetics. Thierens and Goldberg [20] derived convergence-time models for different selections schemes including binary tournament selection. Bäck [25] derived estimates of selection intensity for s-wise tournament and (μ, λ) selection. Miller and Goldberg [26] developed convergence-time models for s-wise tournament selection and incorporated the effects of external noise. Bäck [27] developed convergence-time models for (μ, λ) selection. Even though the selection-intensity-based convergence-time models were developed for the OneMax problem, Miller and Goldberg [28] observed that they are generally applicable to additively decomposable problems of bounded order. Here, we use the convergence-time model of Miller and Goldberg [26]:

$$t_c = \frac{\pi}{2I}\sqrt{\ell}, \tag{2}$$

where I is the selection intensity, and $\ell = mk$ is the string length. For binary tournament selection, $I = 1/\sqrt{\pi}$.

Using equations 1 and 2, we can now predict the scalability, or the number of function evaluations required for successful convergence, of GAs as follows:

$$n_{\text{fe,GA}} = n \cdot t_c = \frac{\pi^2}{4}\frac{\sigma_{BB}}{d}\sqrt{k}\log m \cdot 2^k \cdot m. \tag{3}$$

4.2 Scalability of BB-Wise Mutation Algorithm

Since the initial point is evaluated once and after that for each of the m BBs, $2^k - 1$ individuals are evaluated, the total number of function evaluations required for the BBMA is

$$n_{\text{fe,BBMA}} = \left(2^k - 1\right) m + 1. \tag{4}$$

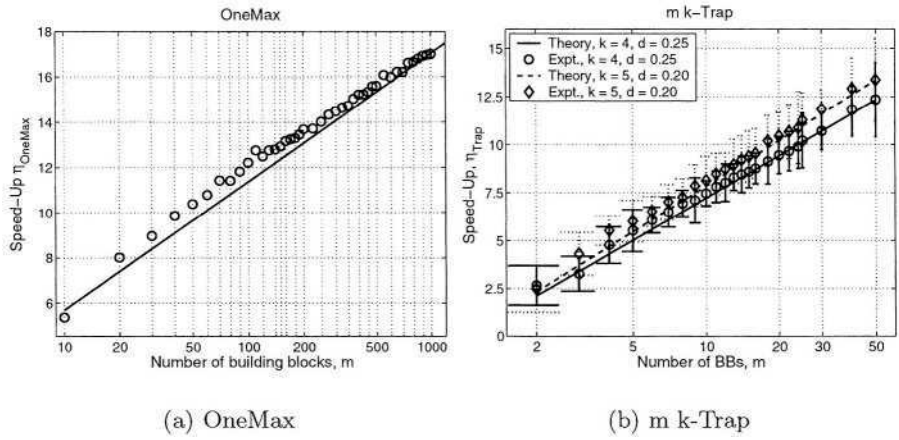
The results from the above subsections (Equations 3 and 4) indicate that while the scalability of a selectorecombinative GA is $\mathcal{O}\left(2^k m \log m\right)$, the scalability of the BBMA is $\mathcal{O}\left(2^k m\right)$. This is in contrast to a random-walk mutation algorithm with no BB knowledge which scales as $\mathcal{O}\left(m^k \log m\right)$ [4]. By searching among building-block neighborhoods, the selectomutative algorithm scales-up significantly better than a mutation operator with no linkage information and provides a savings of $\mathcal{O}(\sqrt{k}\log m)$ evaluations over the GA. The savings comes from the extra evaluation required for the convergence and decision-making in the selectorecombinative GAs.

The speed-up—which is defined as the ratio of number of function evaluations required by a GA to that required by BBMA—obtained by using a BB-wise mutation algorithm over a selectorecombinative GA is given by

$$\eta = \frac{n_{\text{fe,GA}}}{n_{\text{fe,BBMA}}} = \mathcal{O}\left(\sqrt{k}\log m\right). \tag{5}$$

In particular, the speed-up for the OneMax problem $(k = 1)$ is given by

$$\eta_{\text{OneMax}} = \frac{\frac{\pi^2}{4} m \log m}{m + 1} \approx \frac{\pi^2}{4}\log m, \tag{6}$$



(a) OneMax (b) m k-Trap

Fig. 1. Empirical verification of the speed-up predicted for using BB-wise mutation over a selectorecombinative GA by Equations 6 and 7. The empirical results are averaged over 900 independent runs. The results show that the speed-up obtained by BB-wise mutation algorithm over a GA is $\mathcal{O}(\sqrt{k}\log m)$.

and for the GA-haxd m k-Trap function [29], the speed-up is given by

$$\eta_{\text{Trap}} = \frac{\frac{\pi^2}{4}\frac{\sigma_{BB}}{d}\sqrt{k}2^k m\log m}{(2^k-1)m+1} \approx \frac{\pi^2}{4}\frac{\sigma_{BB}}{d}\sqrt{k}\log m. \tag{7}$$

The speed-up predicted by Equations 6 and 7 are verified with empirical results in Figures 1(a) and 1(b), respectively. The results are averaged over 900 independent runs. The results show that there is a good agreement between the predicted and observed speed-up. The results show that for deterministic additively separable problems, a BB-wise mutation algorithm is about $\mathcal{O}(\sqrt{k}m)$ times faster than a selectorecombinative genetic algorithm.

5 Crossover vs. Mutation: Noisy Fitness Functions

In the previous section, we observed that BB-wise mutation scales-up better than a crossover on deterministic additively separable problems. Furthermore, a selectomutative algorithm was able to overcome *deception,* one of the key factors influencing problem difficulty, using linkage (neighborhood) information and enumeration within the neighborhood. In this section, we introduce another dimension of problem difficulty in *extra-BB noise* [15] and analyze if the BB-wise mutation maintains its edge over crossover. That is, we analyze whether a selectorecombinative or a selectomutative GA works better on additively separable problems with additive external Gaussian noise.

We follow the same approach outlined in the previous section and consider the scalability of crossover and mutation.

5.1 Scalability of Selectorecombinative GAs

Again we use the convergence-time and population-sizing models to determine the scalability of GAs under the presence of unbiased Gaussian noise. We use an approximate form of the gambler's ruin population-sizing model for noisy environments:

$$n = \frac{\sqrt{\pi}}{2}\frac{\sigma_{BB}}{d}2^k\sqrt{m}\log m\sqrt{\left(1+\frac{\sigma_N^2}{\sigma_f^2}\right)}, \tag{8}$$

where σ_N^2 is the variance of the noise, and $\sigma_f^2 = m\sigma_{BB}^2$ is the fitness variance.

We use an approximate form of Miller and Goldberg's [26] convergence-time model:

$$t_c = \frac{\pi}{2I}\sqrt{m}\sqrt{1+\frac{\sigma_N^2}{\sigma_f^2}}. \tag{9}$$

A detailed derivation of the above equation and other approximations are given elsewhere [15,30].

The population-sizing and convergence-time models indicate that the exogenous noise increases the population size and elongates the convergence time. Using equations 1 and 2, we can now predict the scalability, or the number of function evaluations required for successful convergence, of GAs as follows:

$$n_{\text{fe,GA}} = \frac{\pi^2}{4}\frac{\sigma_{BB}}{d}\sqrt{k}\log m\cdot\left(1+\frac{\sigma_N^2}{\sigma_f^2}\right)\cdot 2^k\cdot m. \tag{10}$$

5.2 Scalability of BB-Wise Mutation Algorithm

Unlike the deterministic case where a BB was perturbed and evaluated once, in the presence of exogenous noise we cannot rely on only a single evaluation. In other words, in the presence of noise, an average of multiple samples of the fitness should be used in deciding between competing building blocks. Now the question remains as to exactly how many samples have to be considered. This issue of exact samples of fitness required to correctly decide between competing building blocks in the presence of noise has been addressed elsewhere [21]:

$$n_s = 2c\sigma_N^2, \tag{11}$$

where n_s is the number of independent fitness samples, and c is the square of the ordinate of a one-sided standard Gaussian deviate at a specified error probability α. For low error values, c can be obtained by the usual approximation for the tail of a Gaussian distribution: $\alpha \approx \exp(-c/2)/(\sqrt{2c})$. In this paper, we have used $\alpha = 1/m$. Equation 11 is empirically verified for the Noisy-OneMax problem in Figure 2. The results show a good agreement between the model and experiments.

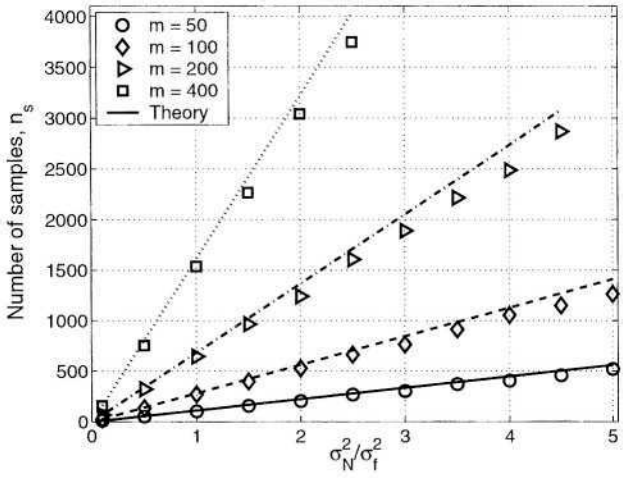


Fig. 2. Comparison of the number of samples of fitness evaluations per individual required to correctly decide between competing building blocks as predicted by Equation 11 with empirical results.

Since the initial point is evaluated n_s times and after that for each of the m BBs, $2^k - 1$ individuals are evaluated n_s times, the total number of function evaluations required for the BBMA for noisy fitness functions is given by

$$\begin{aligned} n_{\text{fe,BBMA}} &= n_s \left[\left(2^k - 1\right) m + 1 \right], \\ &= \left(2c \frac{\sigma_N^2}{\sigma_f^2} \cdot m\sigma_{BB} \right) \left[\left(2^k - 1\right) m + 1 \right]. \end{aligned} \tag{12}$$

The results from the above subsections (Equations 10 and 12) indicate that under the presence of exogenous noise, a selectorecombinative GA scales as $\mathcal{O}\left(2^k m \log m(1 + \sigma_N^2/\sigma_f^2)\right)$. On the other hand, the BB-wise mutation scales as $\mathcal{O}\left(2^k m^2(\sigma_N^2/\sigma_f^2)\right)$. Therefore, for constant values of σ_N^2/σ_f^2, a selectorecombinative GA is $\mathcal{O}(\sqrt{k}m/\log m)$ times faster than the BB-wise mutation. By implicitly averaging out the exogenous noise, crossover is able to overcome the extra effort needed for the convergence and decision-making. On the other hand the explicit averaging via multiple fitness samples by the BB-wise mutation leads to quadratic scale-up—in terms of number of building blocks—in the number of function evaluations.

The speed-up—which is defined as the ratio of number of function evaluations required by mutation to that required by crossover—obtained by using a

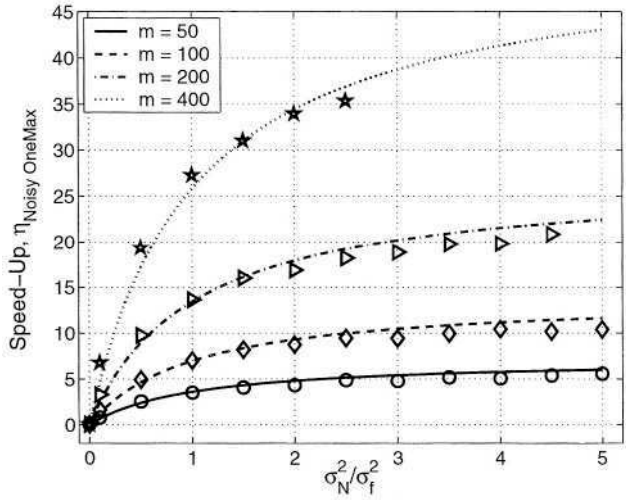


Fig. 3. Empirical verification of the speed-up predicted for using BB-wise mutation over a selectorecombinative GA by Equation 14 for the OneMax problem with exogenous noise. The empirical results are averaged over 900 independent runs. The results show that a selectorecombinative GA uses significantly less number of function evaluations than the BB-wise mutation algorithm.

selectorecombinative over selectomutative GA is given by

$$\eta_{\text{Noise}} = \frac{n_{\text{fe,BBMA}}}{n_{\text{fe,GA}}} = \mathcal{O}\left[\sqrt{k}\frac{m}{\log m}\left(\frac{\frac{\sigma_N^2}{\sigma_f^2}}{1+\frac{\sigma_N^2}{\sigma_f^2}}\right)\right]. \tag{13}$$

In particular, the speed-up for the OneMax problem ($k = 1$) is given by

$$\eta_{\text{Noisy OneMax}} = \frac{4c}{\pi^2}\frac{m}{\log m}\left(\frac{\frac{\sigma_N^2}{\sigma_f^2}}{1+\frac{\sigma_N^2}{\sigma_f^2}}\right). \tag{14}$$

The speed-up predicted by Equation 14 is verified with empirical results in Figure 3. The results are averaged over 900 independent runs. The results show that there is a good agreement between the predicted and observed speed-up. The results show that for stochastic additively separable problems with constant noise variance, a selectorecombinative GA is about $\mathcal{O}(\sqrt{k}m/\log m)$ times faster than the BB-wise mutation algorithm.

6 Future Work

The results of this paper indicate that there are significant advantages of using a mutation operator that performs hillclimbing in the BB space and indicates many avenues of future research some of which are listed in the following:

Hybridization of crossover and BB-wise mutation: While this paper consider a bounding case of crossover vs. mutation, it might be (more likely it is) more effective to use an efficient hybrid of crossover *and* mutation.

Designing BB-wise Mutation: In this paper, we assumed that the BB information was known, which generally is not the case. Over the last few years, effective recombination operators that adapt linkage have been developed in a systematic manner [15]. On the other hand, most mutation operators, including adaptive ones, search in the local neighborhood of a solution. Furthermore, there has been growing evidence of the importance of using good neighborhood operators in determining the effectiveness of local-search methods [1,2]. Despite the importance of having good neighborhood information, a general methodology for designing operators with good neighborhood information is non-existent. That is, little attention has been paid to systematically design effective mutation operators that performs local search in the building-block space [16]. The results of this paper indicate that the dividends obtained by designing BB-wise mutation operators that adaptively identify and utilize good neighborhood information can be significant.

Problems with overlapping building blocks: While this paper considered problems with non-overlapping building blocks, many problems have different building blocks that share common components. An analysis similar to the one presented in this paper can be performed to predict which of the two algorithms excel. However, since the effect of overlapping variable interactions is similar to that of exogenous noise [15], based on the results of this paper crossover is likely to be more useful than the mutation for solving problems with overlapping building blocks.

Hierarchical problems: One of the important class of nearly decomposable problems is hierarchical problems, in which the building-block interactions are present at more than a single level. Further investigation is necessary to analyze if BB-wise mutation can help speed-up the scalability of selectorecombinative GAs.

7 Summary and Conclusions

In this paper, we have introduced a building-block-wise mutation operator which efficiently searches among the competing building block (BB) neighborhood. We also compared the computational costs BB-wise mutation algorithm with a selectorecombinative genetic algorithm for both deterministic and stochastic additively separable problems. Our results show that while the BB-wise mutation provides significant advantage over crossover for deterministic problems, crossover maintains significant edge over the BB-wise mutation on stochastic problems. The results show that the speed-up of using BB-wise mutation on deterministic problems is $\mathcal{O}(\sqrt{k}\log m)$, where k is the BB size, and m is the number of BBs. Likewise, the speed-up of using crossover on stochastic problems with fixed noise variance is $\mathcal{O}(m\sqrt{k}/\log m)$.

Acknowledgments. This work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant F49620-03-1-0129, the National Science Foundation under ITR grant DMR-99-76550 (at Materials Computation Center), and ITR grant DMR-0121695 (at CPSD), and the Dept. of Energy through the Fredrick Seitz MRL (grant DEFG02-91ER45439) at UIUC. The U.S. Government is authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research, the National Science Foundation, or the U.S. Government.

References

1. Barnes, J.W., Dimova, B., Dokov, S.P.: The theory of elementary landscapes. Applied Mathematical Letters **16** (2003) 337-343
2. Watson, J.P.: Empirical Modeling and Analysis of Local Search Algorithms For The Job-Shop Scheduling Problem. PhD thesis, Colorado State University, Fort Collins, CO (2003)
3. Mühlenbein, H.: Evolution in time and space- the parallel genetic algorithm. Foundations of Genetic Algorithms (1991) 316–337
4. Mühlenbein, H.: How genetic algorithms really work: Mutation and hillclimbing. Parallel Problem Solving from Nature II (1992) 15–26
5. Mitchell, M., Holland, J., Forrest, S.: When will a genetic algorithm outperform hill-climbing. Advances in Neural Information Processing Systems **6** (1994) 51–58
6. Baum, E.B., Boneh, D., Garrett, C.: Where genetic algorithms excel. Evolutionary Computation **9** (2001) 93–124
7. Spears, W.M.: Crossover or mutation. Foundations of Genetic Algorithms **2** (1993) 221–237
8. Goldberg, D.E.: Using time efficiently: Genetic-evolutionary algorithms and the continuation problem. Proceedings of the Genetic and Evolutionary Computation Conference (1999) 212–219 (Also IlliGAL Report No. 99002).
9. Srivastava, R., Goldberg, D.E.: Verification of the theory of genetic and evolutionary continuation. Proceedings of the Genetic and Evolutionary Computation Conference (2001) 551–558 (Also IlliGAL Report No. 2001007).
10. Srivastava, R.: Time continuation in genetic algorithms. Master's thesis, University of Illinois at Urbana-Champaign, General Engineering Department, Urbana, IL (2002) (Also IlliGAL Report No. 2001021).
11. Cantú-Paz, E., Goldberg, D.E.: Are multiple runs of genetic algorithms better than one? Proceedings of the Genetic and Evolutionary Computation Conference (2003) 801–812
12. Cantú-Paz, E.: Efficient and accurate parallel genetic algorithms. Kluwer Academic Pub, Boston, MA (2000)
13. Luke, S.: When short runs beat long runs. Proceedings of the Genetic and Evolutionary Computation Conference (2001) 74–80
14. Fuchs, M.: Large populations are not always the best choice in genetic programming. Proceedings of the Genetic and Evolutionary Computation Conference (1999) 1033–1038

15. Goldberg, D.E.: Design of innovation: Lessons from and for competent genetic algorithms. Kluwer Academic Publishers, Boston, MA (2002)
16. Sastry, K., Goldberg, D.: Designing competent mutation operators via probabilistic model building of neighborhoods. Proceedings of the Genetic and Evolutionary Computation Conference (2004) (To appear) (Also IlliGAL Report No. 2004006).
17. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, MI (1975)
18. Goldberg, D.E.: Genetic Algorithms in Search Optimization and Machine Learning. Addison-Wesley, Reading, MA (1989)
19. Goldberg, D.E., Korb, B., Deb, K.: Messy genetic algorithms: Motivation, analysis, and first results. Complex Systems **3** (1989) 493–530 (Also IlliGAL Report No. 89003).
20. Thierens, D., Goldberg, D.E.: Convergence models of genetic algorithm selection schemes. Parallel Problem Solving from Nature **3** (1994) 116–121
21. Goldberg, D.E., Deb, K., Clark, J.H.: Genetic algorithms, noise, and the sizing of populations. Complex Systems **6** (1992) 333–362 (Also IlliGAL Report No. 91010).
22. Harik, G., Cantú-Paz, E., Goldberg, D.E., Miller, B.L.: The gambler's ruin problem, genetic algorithms, and the sizing of populations. Evolutionary Computation **7** (1999) 231–253 (Also IlliGAL Report No. 96004).
23. Mühlenbein, H., Schlierkamp-Voosen, D.: Predictive models for the breeder genetic algorithm: I. continuous parameter optimization. Evolutionary Computation **1** (1993) 25–49
24. Bulmer, M.G.: The Mathematical Theory of Quantitative Genetics. Oxford University Press, Oxford (1985)
25. Bäck, T.: Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. Proceedings of the First IEEE Conference on Evolutionary Computation (1994) 57–62
26. Miller, B.L., Goldberg, D.E.: Genetic algorithms, tournament selection, and the effects of noise. Complex Systems **9** (1995) 193–212 (Also IlliGAL Report No. 95006).
27. Bäck, T.: Generalized convergence models for tournament—and (μ, λ)—selection. Proceedings of the Sixth International Conference on Genetic Algorithms (1995) 2–8
28. Miller, B.L.: Noise, Sampling, and Efficient Genetic Algorithms. PhD thesis, University of Illinois at Urbana-Champaign, General Engineering Department, Urbana, IL (1997) (Also IlliGAL Report No. 97001).
29. Goldberg, D.E.: Simple genetic algorithms and the minimal, deceptive problem. In Davis, L., ed.: Genetic algorithms and simulated annealing. Morgan Kaufmann, Los Altos, CA (1987) 74–88
30. Sastry, K.: Evaluation-relaxation schemes for genetic and evolutionary algorithms. Master's thesis, University of Illinois at Urbana-Champaign, General Engineering Department, Urbana, IL (2001) (Also IlliGAL Report No. 2002004).

Classification with Scaled Genetic Algorithms in a Coevolutionary Setting

Lothar M. Schmitt

The University of Aizu, Aizu-Wakamatsu City, Fukushima Pref. 965-8580, Japan
lothar@u-aizu.ac.jp

Abstract. This work discusses asymptotic convergence of scaled genetic algorithms in a coevolutionary setting where the underlying population contains fixed numbers of creatures of various types. These types of creatures can act on each other in cooperative or competitive manner. The genetic algorithm uses common mutation and crossover operators as well as proportional fitness selection. By a scaled genetic algorithm, we mean that the mutation and crossover rates have to be annealed to zero in proper fashion over the course of the algorithm, and power-law scaling is used for the fitness function with (unbounded) logarithmic growth in the exponent. In the case that a scaled (non-elitist, non-memory) genetic algorithm is used as function optimizer, it has been shown [Theoret. Comput. Sci. 310, p. 181] that such an algorithm is able to find global optima asymptoticly with probability one. However, in the case of a coevolutionary setting, global optima need not exist. Based upon a properly defined, population-dependent fitness function, the set of creatures of a specific type can be canonically grouped into equivalence classes such that all members of one equivalence class are either inferior or superior to all members of another class. We show that in the situation of a coevolutionary setting, a scaled genetic algorithm at least retains the property of converging to a probability distribution over such populations that contain only copies of one creature from the *top-class* for every or a selected group of types while on the other hand maintaining a noisy field of test-cases.

1 Introduction

This work discusses a scaled, non-elitist, non-memory genetic algorithm in a coevolutionary setting that, in principle, is able to find global optima asymptotically. The algorithm, which is a significant extension of algorithms described in [21,22,24] is new in that: (1) it allows for parts of the population to be optimized while other parts remain "noisy" and provide for a "challenging" environment in which the optimization takes place; and (2) some "unnatural" condition on the fitness function used in [21,22] and, in case of a coevolutionary interpretation, also in [24, Thm. 3.3.2, Cors. 3.3.3–4] is removed such that a classification-algorithm is obtained in a very general situation.

1.1. Optimization Tasks in a Coevolutionary Setting. In order to give a precise account what is to be optimized in a coevolutionary setting in contrast to,

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 138–149, 2004.

e.g., investigations of population dynamics as in [8,12], let us start the discussion by listing a few examples:

Example 1.1.1 (Single-species competitive setting): A typical example for a single-species setting where a coevolutionary optimization is performed is that of deterministic game-playing strategies $\mathcal{C}$. Such strategies (agents, programs) are supposed to be encoded here as strings of symbols over a finite alphabet[1] $\mathcal{A}$ which are bounded in length with fixed upper bound $\ell \in \mathbb{N}$. If $c, d \in \mathcal{C}$ are game-playing strategies, then they determine a number $<c,d> \in \{0, \pm 1\}$ depending upon the outcome of the game. We set $<c,d> = -1$, if c wins; $<c,d> = 0$ in case of a draw; $<c,d> = 1$, if d wins. The simplest definition for a positive, population-dependent fitness function $f(c,p)$,—where p is a population in which c resides—, is then given by:

$$f(c,p) = \sum_{c \neq d \in p} (1 - <c,d>). \tag{1}$$

While there may be no strategy $c \in \mathcal{C}$ that is *not inferior to all other* strategies in the above setting (*i.e.,* a global maximum), there likely exists a minimal subset $\mathcal{C}^{\max} \neq \mathcal{C}$ of "superior" strategies such that for every $c \in \mathcal{C}^{\max}$, $d \notin \mathcal{C}^{\max}$ and every population p, one has $f(c,p) > f(d,p)$, *i.e.,* elements of $\mathcal{C}^{\max}$ are in any population p superior to elements not in $\mathcal{C}^{\max}$. Our approach to optimization in this situation is to search for elements of $\mathcal{C}^{\max}$. What we shall outline below is a scaled genetic algorithm that asymptotically finds elements of $\mathcal{C}^{\max}$ essentially without further uncontrollable assumptions on f or the coevolutionary setting.

Example 1.1.2 (Two-species competitive setting): A typical example for a two-species competitive setting is obtained, if we distinguish in Example 1.1.1 between first-move strategies $\mathcal{C}_1$ and second-move strategies $\mathcal{C}_2$. For $c_1 \in \mathcal{C}_1$ and $c_2 \in \mathcal{C}_2$ the fitness function can then be defined as:

$$f(c_1,p) = \exp(\gamma_1 \sum_{c_2 \in p \cap \mathcal{C}_2} <c_1,c_2>), \qquad \gamma_1 = -1, \text{ and} \tag{2}$$

$$f(c_2,p) = \exp(\gamma_2 \sum_{c_1 \in p \cap \mathcal{C}_1} <c_1,c_2>), \qquad \gamma_2 = 1. \tag{3}$$

We assume here that the number of elements $s_j > 0$ of creatures $c_j \in p \cap \mathcal{C}_j$ is fixed for $j = 1, 2$ over the course of the algorithm, *i.e.,* we do not allow for population-dynamics.

Another example for a two-species, adversary setting would be to measure the performance (*i.e.,* execution time) $<c_1,c_2>$ of sorting programs[2] $c_1 \in \mathcal{C}_1$ acting on unsorted tuples (test-instances) $c_2 \in \mathcal{C}_2$ of finite length $\ell_2 \in \mathbb{N}$. Here, we can define the fitness function as in lines *(2)* and *(3).* Sorting programs $c_1 \in \mathcal{C}_1$ aim for short execution times while unsorted tuples $c_2 \in \mathcal{C}_2$ aim for long execution times.

---

[1] A larger alphabet is favorable in several regards: (1) The length of creatures stays smaller; consequently, the population-size can stay smaller (*cf.,* lines *(22,24,26))* and the fitness-evaluation is computationally less extensive (*cf.,* lines *(2)* and *(3)* and the table in Sec. 2.2). (2) The mutation rate can be annealed to zero by a faster schedule (*cf.,* Def. 3.2.2). (3) The fitness-function can be exponentiated at a *slower* rate *(cf.,* lines *(23,25,27))* allowing the mixing operators more time in exploring the search space.

[2] Programs of length bounded by a fixed $\ell_1 \in \mathbb{N}$.

Example 1.1.3 (Two-species cooperative setting): A typical example for a two-species cooperative setting is the simultaneous performance-optimization of interacting components of a system. Specifically, one may be interested in optimizing the performance of a mechanical robot by computer-simulation in regard to minimizing energy $E = \langle \cdot, \cdot \rangle$ of motion. For example on the hardware-side $\mathcal{C}_1$, one could optimize the layout or placement of parts. See [25] for application of a genetic algorithm to such a setting. For example on the software-side $\mathcal{C}_2$, one could be interested in finding a neural network [29] which is optimized[3] to control/correct certain aspects of the motion but is required to be "robust", *i.e.,* handle various types of situations uniformly well. Here, we can define the fitness function as in lines *(2)* and (3) with $\gamma_1 = \gamma_2 = -1$.

In the settings of both Examples 1.1.2 and 1.1.3, we intend to find "superior" elements in the *top-classes* $\mathcal{C}_1^{\max} \subset \mathcal{C}_1$ and $\mathcal{C}_2^{\max} \subset \mathcal{C}_2$ similar to the case of Example 1.1.1 (see also footnote 4). Altogether, we have:

Consider a coevolutionary setting as in the above examples 1.1.2 and 1.1.3 where creatures of different types of species $\mathcal{C}_j$ interact via an $\mathbb{R}$-valued duality or evaluation procedure $\langle \cdot, \cdots \rangle$. Such a setting gives rise to a species- and population-dependent fitness function f in a canonical way as in lines (2) and (3). No global optima for f may exist within some or all of the species $\mathcal{C}_j$. In this presentation, we shall aim to describe a dynamically scaled genetic algorithm that finds creatures (candidate solutions) that are "optimal" in a simple and natural sense without uncontrollable restrictions on the coevolutionary setting.

1.2. Optimization with Genetic Algorithms. Genetic algorithms, a particular case of evolutionary algorithms, were invented by Holland [11] and are by now a well-established tool for search and optimization. Techniques using the operational framework of genetic algorithms have significant applications in mechanical and electrical engineering such as, *e.g.,* the construction of the turbine for the engine of the Boeing 777 airplane [26, discussion of p. 2357–8] or design of electric circuits [15] and antennas [16,26]. The common usage of a genetic algorithm as function optimizer for a(fitness-)function $f : \mathcal{C} \to \mathbb{R}^+$ as described, *e.g.,* in [10,27] or [20,24] is as follows: first a population p is initiated as an s-tuple of creatures $c_1, \ldots c_s \in \mathcal{C}$ $(s \in 2\mathbb{N})$, then three operations —crossover, mutation, and fitness-selection— are applied cyclically and iteratively to the creatures in the population until a termination condition is satisfied. The combined crossover-mutation phase of a genetic algorithm is also called the mixing-phase of the algorithm, *cf.* [27, p. 32]. Crossover is inspired by exchange of genetic information in living organisms, *e.g.,* during the process of sexual reproduction. Mutation is inspired by random change of genetic information in living organisms, *e.g.,* through the effects of radiation or chemical mismatch. Fitness selection models increased reproductive success of organisms c that are better adapted to their environment (*i.e.,* have higher value $f(c)$). Usually, fitness selection includes a random arrangement of selected creatures/individuals in the population.

[3] We note that the often-used back-propagation algorithm for neural networks [29, p. 187] is a gradient method which is not guaranteed to find a global minimum.

The introduction of [24, pp. 183-4, 186-7] lists a collection of references in regard to theoretical approaches to asymptotic convergence of genetic algorithms. Most of these approaches require satisfaction of some auxiliary condition on the algorithm or problem-setting in order to achieve convergence such as using the elitist strategy [18] or an infinite population limit [27, p. 147]. In [24, Thm. 3.3.2, Cors. 3.3.3-4] asymptotic convergence to global optima is shown for a genetic algorithm with arbitrary fitness function that satisfies *all* goals (1)-(4) formulated in [4, p. 270] and is much in the spirit of the simulated annealing algorithm. In particular, the scaled genetic algorithm described in [24] operates with a small population-size s that can be set by the user as low as $\ell+1$ (ℓ length of creatures). In addition, the techniques and results in [24] essentially solve the single-species coevolutionary optimization problem as described in Example (1.1.1) which, however, shall be considerably extended below.

1.3. De Jong's Challenge. In [7], the need for a theoretical framework for coevolutionary genetic algorithms and possible convergence theorems in regard to coevolutionary optimization ("arms races") was emphasized. Such a theoretical framework requires, in particular, treatment of a population-dependent fitness function. While there is a substantial amount of work in: (1) practical applications, and (2) experiments with the setting of coevolutionary algorithms, theoretical advance in regard to convergence seems to be limited. Some static aspects of coevolutionary genetic algorithms in regard to the order on creatures (*i.e.,* in our notation elements of $\mathcal{C}_1$) and test-instances (*i.e.,* elements of $\mathcal{C}_2$) have been investigated, *e.g.,* in [1,2]. Work in [6] discusses convergence of the evaluation procedure towards an ideal evaluation function. A thorough investigation of complexity/convergence of a (1 + 1) coevolutionary algorithm without crossover for maximization of pseudo-Boolean functions can be found in [14]. However, there seems to be no black-box-scenario, coevolutionary global optimization theorem in the literature except results in [20, Thm. 8.6, Rem. 8.7] for a population-dependent fitness function with single dominant creature in a single-species setting, [24, Thm. 3.3.2, Cors. 3.3.3-4] for a population-dependent fitness function with a *'set of strictly dominant creatures of equal fitness'* in a single-species setting, and —already inspired by [7]— results with distinct crossover operators in [21,22] for a population-dependent fitness function with a *'set of strictly dominant creatures of equal fitness for every species that is optimized'* in a multi-species setting.

The new algorithm described below addressees, in particular, the following two problems:

1. The condition of a *'set of strictly dominant creatures of equal fitness for every species that is optimized'* shall be removed. Thus, the algorithm shall asymptotically deliver creatures from the *top equivalence class(es)*[4] pertaining to the fitness function with probability one. This addresses and solves (or circumvents) the problem of *intransitivity superiority* [28, p. 702] in a reasonable manner.

[4] The fitness function f defines a canonical relation $\leq_f$ on every type of species: $c \leq_f d \Leftrightarrow \exists p\colon f(c,p) \leq f(d,p)$. If $\ll_f$ denotes the transitive closure of $\leq_f$, then the equivalence class of c is given by $[c]=\{d : c \ll_f d, \text{ and } d \ll_f c\}$.

2. Part of the population is kept "noisy" driven by constant, non-zero mutation. This part need not be governed by a selection mechanism that converges against the elitist strategy. This allows for maintenance of a larger diverse collection of good "test-instance" within the population. In principle, techniques proposed in [5], [6], and [9] could be incorporated in a properly designed enhancement/selection mechanism acting on the noisy part of the population.

2 Alphabets, Creatures, and Populations

In what follows, we shall restrict ourselves to the case of two species and shall leave consideration of a coevolutionary setting for more species to the reader along the discussion in [21,22]. Note also, that we do not treat here the version of mixing put forward in [27], which produces only one child per mixing operation. However, [21,22] and [24, Sec. 4.3] discuss how to incorporate this mixing in the present framework by doubling the population-size s of the model for the algorithm and considering "selector masks" for selection.

2.1. Alphabets and Creatures. Suppose that two disjoint alphabets

$$\mathcal{A}_j = \{a_j(0), a_j(1) \ldots a_j(\alpha_j - 1)\}, \qquad j=1,2, \tag{4}$$

are given which are used to encode creatures in $\mathcal{C}_j$ as strings over $\mathcal{A}_j$ of length ℓ_j, *i.e.*, $\mathcal{C}_j=(\mathcal{A}_j)^{\ell_j}$, $j=1,2$.

As outlined in [20, Sec. 3.1], it may be advantageous to consider larger alphabets representing finite, equidistant sets of real numbers in applications where real parameters are optimized in a compact domain of $\mathbb{R}^{\ell_j}$. Such a point of view is also supported in [17]. In addition, see footnote 1. Let $\mathcal{V}_j^{(1)} \equiv \mathbb{C}^{\alpha_j}$ be the free vector space over $\mathcal{A}_j$. Let $\mathcal{A}=\mathcal{A}_1 \cup \mathcal{A}_2$ be the combined set of letters considered here.

Let $n_j \in \mathbb{N}$ such that $n_j < \alpha_j/2$. We shall say that $a_j(\iota), a_j(\iota') \in \mathcal{A}_j$ are *close neighbors,* if $\iota \neq \iota'$ and $\min\{|\iota - \iota'|, \alpha_j - |\iota - \iota'|\} \leq n_j$.

Let $\mathcal{C}=\mathcal{C}_1 \cup \mathcal{C}_2$ be the set of all possible creatures considered here. In contrast to, *e.g.,* [5], creatures (phenotypes) are identified with their genetic information (genotypes). We suppose that an evaluation procedure or duality

$$<\cdot,\cdot> : \mathcal{C}_1 \times \mathcal{C}_2 \to \mathbb{R} \tag{5}$$

exists which lets creatures of different types interact and gives rise to a fitness function f defined in lines *(2)* and *(3)* above.

2.2. Populations. Let $s_1, s_3 \in 2\mathbb{N}+2$, $s_2, s_4 \in 2\mathbb{N}_0$. Let $\wp_{2j-n} = (\mathcal{C}_j)^{s_{2j-n}}$ for $j=1,2$, $n=0,1$. The set of populations is now given by

$$\wp = \wp_1 \times \wp_2 \times \wp_3 \times \wp_4. \tag{6}$$

Thus, $s=s_1+s_2+s_3+s_4$ is the number of creatures in a population. The length of a population as word over $\mathcal{A}$ is given by $L=(s_1+s_2)\ell_1+(s_3+s_4)\ell_2$.

We shall call the population $p = (p_1, p_2, p_3, p_4)$ *'multi-uniform at dedicated positions'* or **'partly uniform'** for short, if the sub-populations p_1 and p_3 contain only copies of a single creature $c_1 \in \mathcal{C}_1$ and $c_2 \in \mathcal{C}_2$ respectively.

Let $\mathcal{V}_\wp = \mathcal{V}(\wp)$ be the free complex vector space over $\wp$. The defintion of $\wp$ in line *(6)* yields a canonical tensor-product decomposition of $\mathcal{V}_\wp$:

$$\mathcal{V}_\wp = \mathcal{V}(\wp_1) \otimes \mathcal{V}(\wp_2) \otimes \mathcal{V}(\wp_3) \otimes \mathcal{V}(\wp_4). \tag{7}$$

Let $\mathcal{S}_\wp$ be the set of probability distributions over $\wp$ which is the positive part of the unit sphere of $\mathcal{V}_\wp$ with respect to the ℓ^1-norm.

Let $\mathcal{U} \subset \mathcal{V}_\wp$ be the free vector space over all populations which are partly uniform. Thus, $\wp \cap \mathcal{U}$ is the set of partly uniform populations. In addition, $P_\mathcal{U}$ shall denote the orthogonal projection onto $\mathcal{U}$.

A population p and the action of the genetic algorithm described below are then structured as shown in the following table:

Popul. $p=$	p_1	p_2	p_3	p_4
Creatures in	$\mathcal{C}_1$	$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C}_2$
Position σ of creatures	$1 \dots s_1$	$s_1{+}1 \dots s_1{+}s_2$	$s_1{+}s_2{+}1 \dots s_1{+}s_2{+}s_3$	$s_1{+}s_2{+}s_3{+}1 \dots s$
Position $\hat{\lambda}$ of letters	$1 \dots s_1\ell_1$	$s_1\ell_1{+}1 \dots (s_1{+}s_2)\ell_1$	$(s_1{+}s_2)\ell_1{+}1 \dots (s_1{+}s_2)\ell_1{+}s_3\ell_2$	$(s_1{+}s_2)\ell_1{+}s_3\ell_2{+}1 \dots L$
Purpose	**optimize**	noisy test-set	**optimize**	noisy test-set
Mutation rate	scaled $\mu{\to}0$	constant $\mu'{>}0$	scaled $\mu{\to}0$	constant $\mu'{>}0$
Crossover rate	scaled $\chi{\to}0$	constant $\chi'{>}0$	scaled $\chi{\to}0$	constant $\chi'{>}0$
Fitness evaluation	unbounded scaled fitness-prop.	any standard method	unbounded scaled fitness-prop.	any standard method

3 The Genetic Operators

The genetic operators used in this exibition allow for (1) multi-spot mutation $M_{\hat{\mu},\mu}$ with a local action (*i.e.,* the spot mutation matrix) on the alphabet level that implements a localized search; (2) practically any know crossover operator $C(\chi)$; and (3) selection S_t with scaled fitness-proportional selection on the parts of the population that the user wants to optimize.

3.1. Spot Mutation Matrices. Let $\hat{\mu} \in [0,1]$. For $1 \le \hat{\lambda} \le L$, the spot mutation matrices $\mathbf{m}_j(\hat{\mu})$ model *change* within the alphabets $\mathcal{A}_j$ for the letter $a_j(\iota)$ at single spot $\hat{\lambda}$ in the combined genome of a population ($j{=}1,2$). Let $a_j(\iota), a_j(\iota') \in \mathcal{A}_j$ such that $\iota \neq \iota'$, $0 \le \iota, \iota' \le \alpha_j - 1$. In what follows, we shall use the stochastic matrix $\mathbf{m}_j(\hat{\mu})$ with zero-entries on the diagonal given by:

$$<a_j(\iota'), \mathbf{m}_j(\hat{\mu}) a_j(\iota)> = (1-\hat{\mu})/(2n_j) + \hat{\mu}/(\alpha_j - 1), \tag{8}$$

if $a_j(\iota')$ and $a_j(\iota)$ are close neighbors in the sense of Sec. 2.1, and otherwise

$$<a_j(\iota'), \mathbf{m}_j(\hat{\mu}) a_j(\iota)> = \hat{\mu}/(\alpha_j - 1). \tag{9}$$

Discussion of other choices for $\mathbf{m}_j(\hat{\mu})$ is left to the reader. Local change determined by $\mathbf{m}_j(\hat{\mu})$ is a continuous function of $\hat{\mu}$. The case $\hat{\mu}{=}0$ corresponds to

uniform change within the preferred "small set of close neighbors" of the current letter $a_j(\iota)$ at spot $\hat{\lambda}$ in the combined genome of a population in the spirit of the simulated annealing algorithm. The case $\hat{\mu}=1$ corresponds to uniform random change in $\mathcal{A}_j$.

Note that by [19, p. 5: eq. (7')] or [23, eq. (7)] any stochastic matrix has operator norm 1 with respect to the ℓ^1-norm on the underlying vector space. Hence, its spectrum is contained[5] in the closed unit disk in $\mathbb{C}$. Elementary spectral calculus yields that a matrix of the form $(1-\mu)\mathbf{1}+\mu\mathbf{m}_j(\hat{\mu})$, $\mu\in(0,1/2)$, is invertible since its spectrum cannot contain 0.

3.2. Mutation. Let $\hat{\mu},\hat{\mu}'\in(0,1]$ and $\mu,\mu'\in(0,1/2)$. We shall keep $\hat{\mu}'$ and μ' fixed — a discussion of possible annealing schedules for $\hat{\mu}'$ and μ' is left to the reader. The mutation operator $M_{\hat{\mu},\mu}$ is then given by the following procedure:

Definition 3.2.1 (Mutation operator): For $\hat{\lambda}=1\ldots L$, execute the next two steps: (STEP 1) Decide probabilistically whether or not to change the letter at spot $\hat{\lambda}$ in the current population. The decision for change is made positively with probability μ for $\hat{\lambda}\in[1,s_1\ell_1]\cup[(s_1+s_2)\ell_1+1,(s_1+s_2)\ell_1+s_3\ell_2]$ in which case we set $\tau=\hat{\mu}$. Otherwise, the decision for change is made positively with constant probability μ', and we set $\tau=\hat{\mu}'$. (STEP 2) If the decision has been made positively in step 1, then the letter at spot $\hat{\lambda}$ is altered in accordance with the transition probabilities for letters set by the spot mutation matrix $\mathbf{m}_j(\tau)$ where $j\in\{1,2\}$ is chosen appropriately. ⌋

Let $M_{\hat{\mu},\mu}$ also denote the fully positive, stochastic matrix associated with multiple-spot mutation that acts on $\mathcal{V}_{\wp}$ and describes transition probabilities for entire populations. It is easy to see that the coefficients of $M_{\hat{\mu},\mu}$ are greater than $K\cdot(\hat{\mu}\mu)^{L_o}$ where $K>0$ is a fixed constant and $L_o=s_1\ell_1+s_3\ell_2$.

Since $\mathcal{V}_{\wp}$ is an L-fold tensor-product of spaces $\mathcal{V}_j^{(1)}$, $j\in\{1,2\}$, the matrix $M_{\hat{\mu},\mu}$ is the L-fold tensor-product of invertible matrices of type $(1-\mu)\mathbf{1}+\mu\mathbf{m}_j(\hat{\mu})$ and is therefore invertible (see, *e.g.,* [24, line (7), Prop. 2.2.2.2 and Sec. 5] for details). Using [24, Lemma 1.4.2.2], we can conclude that the steady state probability distribution w_t of a single step G_t of the scaled genetic algorithm as defined in line (*21*) is uniquely determined. This allows to compute w_t via Cramer's rule as in [24, p. 213: proof of Thm. 3.3.2], and makes it relatively easy to establish strong ergodicity of the the inhomogeneous Markov chain which describes the probabilistic behavior of the scaled genetic algorithm considered in this work.

Definition 3.2.2 (Mutation rate annealing schedule): Let $\varphi>0$ and $t_o\in\mathbb{N}$ be such that $\varphi t_o^{-1/(\kappa_o L_o)}<1/2$, where $\kappa_o\in[1,\infty)$ is set by the user as described below. Now, the annealing schedule for the mutation rate is given by:

$$\mu=\mu(t)=\varphi\cdot t^{-1/(\kappa_o L_o)},\quad t\in\mathbb{N}\cap[t_o,\infty).$$

In addition, let $\hat{\mu}=\hat{\mu}(t)$ be defined by one of the following annealing schedules:

1. *Constant local noise.* Set $\kappa_o=1$. $\hat{\mu}\in(0,1]$ is kept constant.
2. *Decreasing local noise.* Choose $\kappa_o\in(1,\infty)$ and $\hat{\varphi}\in(0,\mu(t_o)^{1-\kappa_o}]$. Define: $\hat{\mu}(t)=\hat{\varphi}\cdot\mu(t)^{\kappa_o-1}\in(0,1]$ ⌋

[5] For a proof, consider stretching eigenvectors with respect to the ℓ^1-norm.

As a consequence of Def. 3.2.2, we obtain by [24, Lemma 1.3.1] that the inhomogeneous Markov chain as defined in line *(21)* which describes the probabilistic behavior of the scaled genetic algorithm considered in this work is *weakly ergodic.* To show *strong ergodicity,* one employs [13, p. 160: Thm. V.4.3] or [23, Thm. 3.3.2]. The prerequisites of these Theorems are verified using the facts that: (1) the matrix-entries of the stochastic matrices $C(\chi_t)$, $M_{\hat{\mu}(t),\mu(t)}$, and S_t representing the genetic operators crossover, mutation and selection have a "nice" functional form[6]; (2) the steady state probability distribution of a single step G_t of the scaled genetic algorithm as defined in line *(21)* can be computed via Cramer's rule as in [24, p. 213: proof of Thm. 3.3.2]; and (3) techniques established in [24, Lemma 3.3.1, p. 213 bottom: proof of Thm. 3.3.2].

Finally, let us discuss the *mutation-flow inequality* associated with the setting described thus far. First, define:

$$\beta = \beta(\hat{\mu}, \mu) = \min\{||P_{\mathcal{U}} M_{\hat{\mu},\mu} p||_1 : p \in \wp \cap \mathcal{U}\} \in (0,1). \tag{10}$$

Then, one obtains the mutation flow inequality with the argument in [24, proof of Prop. 2.2.3, second part] (Recall that $\mathcal{U}$ refers to 'partly uniform' here.):

$$\forall v \in \mathcal{S}_\wp\colon\ ||(\mathbf{1} - P_{\mathcal{U}}) M_{\hat{\mu},\mu} v||_1 \leq 1 - \beta + \beta ||(\mathbf{1} - P_{\mathcal{U}}) v||_1. \tag{11}$$

The mutation flow inequality is one of two key ingredients to show convergence to partly uniform populations as $\mu \to 0$ in the course of the scaled genetic algorithm.

3.3. Crossover. We consider a similar framework for crossover as in [24]. For crossover rate $\chi \in [0,1]$, the crossover operator is represented by a stochastic matrix $C = C(\chi)$ with entries that are rational functions in χ. The matrix $C(\chi)$ acts on $\mathcal{V}_\wp$ and describes transition probabilities for entire populations. It satisfies:

$$C(0) = \mathbf{1}_{\mathcal{V}(\wp_1)} \otimes C_2^o \otimes \mathbf{1}_{\mathcal{V}(\wp_3)} \otimes C_4^o, \quad \text{and} \quad \forall p \in \wp \cap \mathcal{U}\colon C(\chi) p \in \mathcal{U}. \tag{12}$$

Here, $\mathcal{V}(\wp_1)$ and $\mathcal{V}(\wp_3)$ and the tensor product refer to the decomposition of $\mathcal{V}_\wp$ given in line (7). If no further information about the crossover operator is known, then let the annealing schedule for the crossover rate be given by:

$$\chi_t = \phi_c \mu(t)^{\kappa_o(\ell_1+\ell_2)+1}, \quad \phi_c \in (0, \mu(t_o)^{-\kappa_o(\ell_1+\ell_2)-1}]. \tag{13}$$

Compare the settings in line *(13)* with [24, Thm. 3.3.2, eqs. (41), (45)].

Now suppose that either the crossover operation C_n^{loc} on sub-populations $\wp_n$, $n=1,2,3,4$, is given by regular one-, two-, or multiple-cutpoint crossover, *i.e.,* the creatures are paired sequentially (c_1, c_2), (c_3, c_4), ... (c_{s-1}, c_s), and with probability χ for every pair $(c_{2\sigma-1}, c_{2\sigma})$ letters (genes) are exchanged within the pairs of creatures in accordance with randomly chosen cutpoints; or suppose that C_n^{loc} is given by regular uniform or gene-lottery crossover. See, *e.g.,* [10, pp. 16–17], [27, p. 43] or [24, Sec. 2.4-5], Suppose that $\chi, \chi' \in (0,1]$ are crossover rates, χ' is kept fixed over the course of the algorithm, and $C(\chi)$ is given by:

$$C(\chi) = C_1^{\text{loc}}(\chi) \otimes C_2^{\text{loc}}(\chi') \otimes C_3^{\text{loc}}(\chi) \otimes C_4^{\text{loc}}(\chi') \tag{14}$$

[6] See the second sentence of Sec. 3.3 for crossover; see the discussion after Def. 3.2.1 in regard to tensor products for mutation, or consult [24, Prop. 2.2.2.1]; and see [24, line (33)] for selection.

in regard to the tensor product decomposition of $\mathcal{V}_\wp$ given in line (7). For reason of simplicity, we shall keep $C_1^{\text{loc}} = C_3^{\text{loc}}$. Discussion of mixed cases shall be left to the reader.

Let $m \in [1, \infty)$. We shall set the annealing schedule for the crossover rate as:

$$\chi_t = \phi_c \mu(t)^{1/m}, \qquad \phi_c \in (0, \mu(t_o)^{-1/m}]. \tag{15}$$

3.4. Selection and Convergence. We shall suppose that the (raw) fitness function f is given as in lines (2) and (3) with $\gamma_{1,2} \in \{\pm 1\}$. The fitness function f shall be power-law scaled, *i.e.,* exponentiated over the course of the algorithm. In fact, we set:

$$f_t(c,p) = (f(c,p))^{g(t)}, \qquad g(t) = B \cdot \log(t - t_o + 2), \tag{16}$$

for $p \in \wp$, $c \in p \cap \mathcal{C}$, $t \in \mathbb{N} \cap [t_o, \infty)$, and fixed $B > 0$. In addition, set $f(c,p) = f_t(c,p) = 0$, if $p \in \wp$ and $c \in \mathcal{C} \setminus p$. In order to define the selection procedure, we set in accordance with the table in Sec. 2.2, and the positions of creatures in the sub-populations p_1, p_2, p_3, p_4 of a population $p \in \wp$ defined there:

$\rho_1 = 1$	$\rho_1' = s_1$	$\rho_3 = \rho_2' + 1$	$\rho_3' = \rho_2' + s_3$	$F_{1,t} = F_{3,t} = f_t$	scaled
$\rho_2 = \rho_1' + 1$	$\rho_2' = \rho_1' + s_2$	$\rho_4 = \rho_3' + 1$	$\rho_4' = s$	$F_{2,t} = F_{4,t} = f$	unscaled

Now, suppose that $p = (c_1, c_2, \ldots, c_s)$, $c_\sigma \in \mathcal{C}$, $1 \leq \sigma \leq s$ is the current population. At time or step $t \in \mathbb{N} \cap [t_o, \infty)$, we attempt to select creatures randomly with probability proportional to their individual fitness-strength $F_{n,t}$, $n = 1, 2, 3, 4$, within the particular sub-population $p_n = (c_\sigma : \sigma = \rho_n \ldots \rho_n')$ of p in which the creature resides. If $c \in \mathcal{C}$, then let let $\#(c, p_n)$ denote the number of copies of c in p_n. Now, we can define the selection procedure:

Definition 3.4.1 (Selection operator S_t): With the current population p as above assemble the new population $q = (d_1, d_2, \ldots, d_s) \in \wp$ with $d_\sigma \in \mathcal{C}$, $1 \leq \sigma \leq s$, in the following manner: For $n = 1, 2, 3, 4$ do: for $\sigma = \rho_n, \ldots, \rho_n'$ do:
Select creature $d_\sigma \in q$ probabilisticly among the creatures in p_n such that a particular $c \in p_n$ has relative probability for being selected as d_σ given by:

$$\left(\sum_{\rho_n \leq \sigma' \leq \rho_n'} F_{n,t}(c_{\sigma'}, p)\right)^{-1} \cdot \#(c, p_n)\, F_{n,t}(c,p). \qquad \rfloor$$

Note that our definition of selection includes unsealed fitness proportional selection for segments p_2 and p_4 of the population p. This was chosen here for reason of simplicity of presentation. In fact, most standard methods for selection such as tournament selection can be used for segments p_2 and p_4; and this shall not alter the main results of the discussion below.

Let S_t also denote the stochastic matrix associated with scaled proportional fitness selection. S_t acts on $\mathcal{V}_\wp$ and describes transition probabilities for entire populations. It is easy to obtain an explicit formula for the coefficients of S_t as in [24, line (33)] which is needed for the proofs of some of the statements made in this exposition. However, we shall only list below some key properties of S_t which differ from corresponding statements in [24, p. 206]:

$$\forall p \in \wp \cap \mathcal{U}\colon\ S_t p \in \mathcal{U}. \tag{17}$$

$$S_t P_{\mathcal{U}} = P_{\mathcal{U}} S_t P_{\mathcal{U}} \Rightarrow (\mathbf{1} - P_{\mathcal{U}}) S_t = (\mathbf{1} - P_{\mathcal{U}}) S_t (\mathbf{1} - P_{\mathcal{U}}). \tag{18}$$

$$\forall p \in \wp \cap \mathcal{U}\colon\ ||P_{\mathcal{U}} S_t p||_1 \geq (s_1)^{-s_1+1} (s_3)^{-s_3+1} =_{\text{def}} 1 - \theta. \tag{19}$$

$$\forall v \in \mathcal{S}_\wp\colon \|(\mathbf{1} - P_{\mathcal{U}})S_t v\|_1 \leq \theta \cdot \|(\mathbf{1} - P_{\mathcal{U}})v\|_1. \tag{20}$$

The properties established in lines *(11), (12), (18)* and *(20)* together with an adaptation of the techniques in the proof of [24, Thm. 3.1.1] show the following:
• *The genetic algorithm considered here converges asymptotically to a probability distribution w_∞ over partly uniform populations only.*

Finally, we can consider the inhomogeneous Markov chain that describes the probabilistic behavior of the scaled genetic algorithm considered in this exposition. In fact, a single step at time $t \in \mathbb{N} \cap [t_o, \infty)$ is described by the following stochastic matrix (t_o is the intial value for t):

$$G_t = S_t \cdot C(\chi_t)^{1-k} \cdot M_{\hat{\mu}(t),\mu(t)} \cdot C(\chi_t)^k, \text{ where } k{=}0, \text{ or } k{=}1. \tag{21}$$

Note that we do NOT suppose that crossover $C(\chi_t)$ and mutation $M_{\hat{\mu}(t),\mu(t)}$ commute. Let $w_t{=}G_t w_t \in \mathcal{S}_\wp$ denote the uniquely determined steady-state distribution of an individual step G_t of the scaled genetic algorithm (*cf.*, the discussion following Def. 3.2.1). Let $H_t{=}\prod_{\tau=t}^{t_o} G_\tau$. We have already established that $(H_t)_{t \in \mathbb{N} \cap [t_o,\infty)}$ is strongly ergodic in the discussion following Def. 3.2.2. Hence, for every $v_o \in \mathcal{S}_\wp$: $\lim_{t\to\infty} H_t v_o = \lim_{t\to\infty} w_t =_{\text{def}} w_\infty$.

We have outlined in the discussion after line *(20)* how to obtain that w_∞ is positive only over partly uniform populations. What remains to show is that:
• *w_∞ is positive only over such populations that contain elements from the 'top equivalence classes' $\mathcal{C}_1^{\max} \subset \mathcal{C}_1$ and $\mathcal{C}_2^{\max} \subset \mathcal{C}_2$ as defined in footnote 4.*
In order to achieve this, one establishes a *'steady-state flow inequality'* for the w_t considered here similar to [24, lines (45–7)]. This yields the following conditions:

General type crossover:	$\kappa_o(\ell_1 + \ell_2) < \min(s_1, s_3)$	*(22)*
General type crossover:	$\kappa_o(\ell_1 + \ell_2) < \kappa_o L_o B \log(\rho_2(f)) + 1$	*(23)*
Regular crossover:	$2m\kappa_o(\ell_1 + \ell_2) < \min(s_1, s_3)$	*(24)*
Regular crossover:	$\kappa_o(\ell_1 + \ell_2) < \kappa_o L_o B \log(\rho_2(f)) + 1/m$	*(25)*
Gene-lottery crossover:	$m\kappa_o(\ell_1 + \ell_2) < \min(s_1, s_3)$	*(26)*
Gene-lottery crossover:	$\kappa_o(\ell_1 + \ell_2) < \kappa_o L_o B \log(\rho_2(f)) + 1/m$	*(27)*

Here, $\rho_2(f)$ is a constant measuring the strength of 'second-to-best elements' in populations containing elements of $\mathcal{C}_1^{\max}$ and $\mathcal{C}_2^{\max}$. In fact, $\rho_2(f)$ is given by:

$$T = \{(p, \hat{c}, c) : p \in \wp,\ p_{2j-1} \cap \mathcal{C}_j^{\max} \neq \emptyset,\ p_{2j-1} \setminus \mathcal{C}_j^{\max} \neq \emptyset \text{ for both } j{=}1,2, \text{ and } \hat{c} \in p_{2j-1} \cap \mathcal{C}_j^{\max},\ c \in p_{2j-1} \setminus \mathcal{C}_j^{\max} \text{ for one } j{=}1,2\}$$

$$\rho_2(f) = \min\{f(\hat{c}, p)/f(c, p) : (p, \hat{c}, c) \in T\} \quad (\text{assume } T \neq \emptyset). \tag{28}$$

$\rho_2(f)$ is easy to determine, if one employs rank ($=f$) for the fitness selection mechanism based upon an originally-given raw fitness function f_r.

Note that lines *(24)* and *(26)* show (with mathematical theory and not experimentally) the remarkable effect that with increasing population size, one is allowed to use a more relaxed cooling schedule for crossover. Thus, for larger population size, the part of the algorithm-design, *i.e.,* definition of creatures (data-structures), which is exploited by crossover plays a more important role.

Conclusion

We have obtained a new, all-purpose (coevolutionary genetic algorithm suitable for optimization in a multi-species setting for which a *global optimization theorem* holds. The proposed algorithm is very much in the spirit of the simulated annealing algorithm. It is realistic in that the population-size and consequently evaluation-time for the fitness function stay relatively small. Explicite annealing schedules for crossover/mutation and exponentiation schedules for the fitness-function scaling are given such that the proposed algorithm, in fact, can be readily implemented. The selection operator which uses fitness-proportional selection applies the scaled fitness function only on part of the population in order to keep the complementary part as a "noisy", non-converging field of "test-instances". Thus, the proposed algorithm improves and generalizes a previously published approach for a revolutionary genetic algorithm in a multi-species setting by removing some technical conditions on the latter and introducing new features such as maintaining a noisy field of test-instances without loosing convergence to global optima. Such convergence is here understood as convergence toward populations contain only elements of canonical equivalence classes of "superior" creatures ($\Leftrightarrow$ global optima) formed in relation to the fitness function. One important aspect of future research on this algorithm should be the incorporation of various enhancement procedures for maintenance of "balanced and rich" sets of test-instances within the population that have been proposed in the literature.

References

1. A. Bucci, J.B. Pollack: A Mathematical Framework for the Study of Coevolution. *Proc. FOGA 7,* K. De Jong, *et al.* (eds.), Morgan Kaufmann, San Francisco, CA, USA (2003)
2. A. Bucci, J.B. Pollack: Focusing versus Intransitivity: Geometrical Aspects of Coevolution. IN: [3], pp. 250–261
3. E. Cantu-Paz *et al.* (eds.): *Proc. GECCO 2003,* LNCS 2723, Springer Verlag, Berlin, Germany (2003)
4. T.E. Davis, J.C. Principe: A Markov Chain Framework for the Simple Genetic Algorithm. *Evolut. Comput.* **1** (1993), pp. 269–288
5. E.D. De Jong: Representation Development from Pareto-Coevolution. IN: [3], pp. 262–273
6. E.D. De Jong, J.B. Pollack: Learning the Ideal Evaluation Function. IN: [3], pp. 277–288
7. K. De Jong: Lecture on Coevolution. IN: Seminar *"Theory of Evolutionary Computation",* H.-G. Beyer *et al.* (chairs), Max Planck Inst. for Comput. Sci. Conf. Cntr., Schloß Dagstuhl, Saarland, Germany (2002)
8. S.G. Ficci, J.B. Pollack: Effects of Finite Populations on Evolutionary Stable Strategies. *Proc. GECCO 2000,* D. Whitley *et al.* (eds.), Morgan Kaufmann, San Francisco, CA, USA (2000), pp. 927–934
9. S.G. Ficici, J.B. Pollack: A Game-Theoretic Memory Mechanism for Coevolution. IN: [3], pp. 286–297
10. D.E. Goldberg: *Genetic Algorithms, in Search, Optimization & Machine Learning.* Addison-Wesley Publishers, Boston, MA, USA (1989)

11. J.H. Holland: *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, MI, USA(1975), MIT Press, Cambridge, MA, USA (1992)
12. J. Horn, J. Cattron: The Paradox of the Plankton: Oscillations and Chaos in Multispecies Evolution. IN: [3], pp. 298–309
13. D.L. Isaacson, R.W. Madsen: *Markov Chains: Theory and Applications.* Prentice-Hall Publishers, Upper Saddle River, NJ, USA (1961)
14. T. Jansen, R.P. Wiegand: Exploring the Explorative Advantage of the Cooperative Coevolutionary (1 + 1) EA. IN: [3], pp. 310–321
15. J.R. Koza, M.A. Keane, M.J. Streeter: *Genetic Programming IV: Routine Human-Competitive Machine Intelligence.* Kluwer Academic Publishers, Dordrecht, The Netherlands (2003)
16. D.S. Linden: Antenna Design Using Genetic Algorithms. *Proc. GECCO 2002,* W.B. Langdon *et al.* (eds.), Morgan Kaufmann Publishers, San Francisco, CA, USA (2002), pp. 1133–1140
17. A. Márkus, G. Renner, J. Vanza. Spline Interpolation with Genetic Algorithms. *Proc. Int. Conf. Shape Modeling,* Aizu Univ. (1997), T.L. Kunii *et al.* (chairs), IEEE Computer Soc. Press, Los Alamitos, CA, USA (1997), pp. 47–54
18. G. Rudolph: Convergence Analysis of Canonical Genetic Algorithms. *IEEE Trans. Neural Networks* **5** (1994), pp. 96–101
19. H.H. Schaefer. *Banach Lattices and Positive Operators.* Springer Verlag, Berlin, Germany (1974)
20. L.M. Schmitt: Theory of Genetic Algorithms. *Theoret. Comput. Sci.* **259** (2001), pp. 1–61
21. L.M. Schmitt: Optimization with Genetic Algorithms in Multi-Species Environments. *Proc. ICCIMA 2003,* Xidian Univ., L. Jiao, *et al.* (eds.), IEEE Computer Soc. Press, Los Alamitos, CA, USA, pp. 194–199
22. L.M. Schmitt: Theory of Coevolutionary Genetic Algorithms. *Proc. ISPA 2003,* Aizu Univ., M. Guo, L.T. Yang (eds.), LNCS 2745, Springer Verlag, Berlin, Germany (2003), pp. 285–293
23. L.M. Schmitt: Asymptotic Convergence of Scaled Genetic Algorithms to Global Optima —A gentle introduction to the theory—. IN: *Frontiers of Evolutionary Computation,* A. Menon (ed.), Genetic Alg. and Evol. Comput. Ser. **11,** Kluwer Publishers, Dordrecht, The Netherlands (2004), pp. 157–192
24. L.M. Schmitt: Theory of Genetic Algorithms II — Models for genetic operators over the string-tensor representation of populations and convergence to global optima for arbitrary fitness function under scaling. *Theoret. Comput. Sci.* **310** (2004), pp. 181–231
25. L.M. Schmitt, T. Kondoh: Optimization of Mass Distribution in Articulated Figures with Genetic Algorithms. *Proc. IASTED Int. Conf. ASM 2000,* Banff, Alberta, Canada, M.H. Hamza (ed.), IASTED-ACTA Press, Anaheim-Calgary-Zürich (2000), pp. 191–197
26. S. Tong, D.J. Powell: Genetic Algorithms: A Fundamental Component of an Optimization Toolkit for Improved Engineering Designs. IN: [3], pp. 2346–2359
27. M.D. Vose: *The Simple Genetic Algorithm: Foundations and Theory.* MIT Press, Cambridge, MA, USA (1999)
28. R.A. Watson, J.B. Pollack: Coevolutionary Dynamics in a Minimal Substrate. *Proc. GECCO 2001,* L. Spector *et al.*(eds.), Morgan Kaufmann Publishers, San Francisco, CA, USA (2001), pp. 702–709
29. J.M. Zurada: *Introduction to Artificial Neural Systems.* West Publ. Co., St. Paul, MN, USA (1992)

New Epistasis Measures for Detecting Independently Optimizable Partitions of Variables

Dong-Il Seo, Sung-Soon Choi, and Byung-Ro Moon

School of Computer Science & Engineering, Seoul National University
Sillim-dong, Gwanak-gu, Seoul, 151-744 Korea
{diseo, sschoi, moon}@soar.snu.ac.kr
http://soar.snu.ac.kr/~{diseo, sschoi, moon}/

Abstract. An optimization problem is often represented with a set of variables, and the interaction between the variables is referred to as epistasis. In this paper, we propose two new measures of epistasis: *internal epistasis* and *external epistasis*. Then we show that they can quantify the decomposability of a problem, which has a theoretical meaning about how strongly the problem is independently optimizable with a partition of variables. We present examples of the problem decomposition and the results of experiments that support the consistency of the measures.

1 Introduction

An optimization problem is specified by a set of problem instances, each of which is a pair $(\mathcal{U}, f)$, where the universe $\mathcal{U}$ is the set of feasible solutions and the fitness function f is a mapping $f : \mathcal{U} \rightarrow \mathbb{R}$ [1]. A solution set is often represented by a set of variables, whose values can have certain ranges. If the ranges are discrete, the problem is said to be combinatorial [2]. The interaction between the variables is referred to as epistasis, which implies that the contribution of a variable to the fitness depends on the values of other variables. The epistasis is one of the main reasons that one cannot solve problems by naive approaches, such as steepest ascent method, which try to optimize each variable independently.

This difficulty was addressed recently with the techniques relevant to the "building blocks" or the "factorization" in the evolutionary algorithms. A building block is a specific assignment to a subset of variables that contributes to a high fitness, which is believed to give a shortcut to the optimal solution. Such building blocks are detected and proliferated in the population by the genetic operators in topological linkage-based genetic algorithms (TLBGAs) [3]. Thus the precise and efficient detection of the building blocks is critical in the black box optimization where no problem-specific knowledge is available. Instead of manipulating the building blocks by the genetic operators, the distribution of the variables is explicitly estimated in estimation-of-distribution algorithms (EDAs) [4,5]. EDAs do not use crossover nor mutation operator. Instead, the new population of individuals is sampled from a probability distribution, which

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 150–161, 2004.

is estimated from selected individuals from the previous generation. Since the exact estimation of the whole distribution of the variables is computationally prohibitive for large-scale problems, the variable set is factorized into a number of subsets in which variables are believed to have strong dependence with each other. The factorization is often based on probabilistic graphic model [6, 7] in recent EDAs. The building block detection and the factorization of the distribution are strongly correlated with the decomposition of a problem since both of them are, in some sense, to find the subgroups of the variables that are likely to be optimized individually. The approaches are, however, not free from specific algorithms and their running status since the building block detection and the model construction are based on the solutions selected by fitness from the population or the probability distribution at that point of time.

There are a number of algorithm-independent measures of epistasis including the epistasis variance by Davidor [8] and the entropic epistasis by Seo *et al.* [9,10]. The epistasis variance quantifies the nonlinearity lying in the fitness landscape based on experimental design [11], while the entropic epistasis measures, using Shannon's information theory [12,13], the amount of information shared by the variables about the fitness. In this paper, we extend the entropic epistasis to the problem decomposition. To do so, we first show that the epistasis can be split into two factors: *internal epistasis* and *external epistasis.* Then we formally define the theoretical concept of problem decomposition in the sense of independent optimization. Finally, we show the relationship between the two novel measures and the decomposability of a problem.

The reset of this paper is organized as follows. We provide a brief overview of the entropic epistasis and propose new epistasis measures in Section 2. Then we formally define the decomposition of a problem and show the relationship between the decomposability and the epistasis of the problem in Section 3. The examples of problem decomposition and the experimental results are presented in Section 4. The conclusions are given finally in Section 5.

2 Epistasis Measures

2.1 Probability Model

Let the variable indices of a given problem be $\mathcal{V} = \{1, 2, \ldots, n\}$, and the alphabet for each variable be $\mathcal{A}_i, i \in \mathcal{V}$. And let the universe and the fitness function be $\mathcal{U} \subseteq \mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_n$ and $f : \mathcal{U} \to \mathcal{F}$, respectively. We assume that the alphabet of each variable is finite. Then, the set of all fitness values $\mathcal{F} \subset \mathbb{R}$ is finite as the universe is finite.

Based on the uniform probability model on the universe, we define a random variable X_i for each variable $i \in \mathcal{V}$ and a random variable Y for the fitness value. Then, the joint probabilistic mass function (jpmf) $p : \mathcal{A}_1 \times \cdots \times \mathcal{A}_n \times \mathcal{F} \to \mathbb{R}$ of the random variables are defined as follows:

$$p(x_1, x_2, \ldots, x_n, y) = \begin{cases} \frac{1}{|\mathcal{U}|} & \text{if } (x_1, x_2, \ldots, x_n) \in \mathcal{U} \\ & \quad \text{and } y = f(x_1, x_2, \ldots, x_n) \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

It means that the probability of a solution $(x_1, x_2, \ldots, x_n)$ and a fitness value y is $\frac{1}{|\mathcal{U}|}$ if the fitness value of the solution $f(x_1, x_2, \ldots, x_n)$ is y, and the probability is zero otherwise. In this paper, we use a conventional notation X_V to denote $(X_{v_1}, X_{v_2}, \ldots, X_{v_k})$ for a variable set $V = \{v_1, v_2, \ldots, v_k\} \subseteq \mathcal{V}$.

It is practical to use a set of sampled solutions instead of the universe $\mathcal{U}$ in Equation (1) for large-scale problems because of the spatial or computational limitations. In the case, the size of the set must be not too small to get results of low levels of distortion (see [10] for details).

In general, the fitness function of a problem is defined on a continuous domain, while each variable has a discrete alphabet in combinatorial optimization problems. Although the set $\mathcal{F}$ of all fitness values of a problem instance is finite, it is less practical to consider each distinct value in $\mathcal{F}$ as a discrete symbol for the random variable Y since we can hardly talk about the statistics including fitness as one of the variables if the solutions rarely share the same fitness. Hence the fitness needs to be discretized into a number of intervals (see [10] for details).

2.2 Significance and Epistasis

The significance of a variable set is defined to be the mutual information between the corresponding random variables and the random variable Y. It is intuitive and natural because if we can get more information about the fitness from the values of the variables then we can say that the variables contribute more to the fitness. Formally, the *significance* $\xi(V)$ of a variable set $V = \{v_1, v_2, \ldots, v_k\} \subseteq \mathcal{V}$, $k \geq 1$, is defined as follows [9,10]:

$$\xi(V) = \frac{I(X_V; Y)}{H(Y)} \tag{2}$$

where I denotes the mutual information [13]. We do not consider the case when the fitness is constant, i.e., the case of $|\mathcal{F}| = 1$, because no optimization is required in the case. Hence we regard the entropy of Y as a nonzero value. The equation in the definition is a normalized formula, as is easily verified in the following. The significance $\xi(V)$ of a variable set $V \subseteq \mathcal{V}$ satisfies the following inequality:

$$0 \leq \xi(V) \leq 1. \tag{3}$$

If the significance is zero, then we cannot get any information about the fitness from the corresponding variables. On the contrary, if the significance is one, then we can fully identify the fitness from the variables. It is clear that $\xi(\mathcal{V}) = 1$ since $I(X_\mathcal{V}; Y) = H(Y)$, and $\xi(V) \leq \xi(W)$ for $V \subseteq W \subseteq \mathcal{V}$ since $I(X_V; Y) \leq I(X_W; Y)$ by the chain rule (see [13] p. 22).

The epistasis can be defined rigorously from the significance. From (2), the significance of a variable set $V \subseteq \mathcal{V}$ is denoted by $\xi(V)$ and the significance of each variable v in $\mathcal{V}$ is denoted by $\xi(v)$. We define the epistasis between the variables in V to be the difference between $\xi(V)$ and the summation of all $\xi(v)$'s,

$v \in V$. Formally, the *epistasis* $\varepsilon(V)$ of a variable set $V = \{v_1, v_2, \ldots, v_k\} \subseteq \mathcal{V}$, $k \geq 1$, is defined as follows [9,10]:

$$\varepsilon(V) = \begin{cases} \dfrac{\xi(V) - \sum_{i=1}^{k} \xi(v_i)}{\xi(V)} & \text{if } I(X_V; Y) \neq 0 \\ 0 & \text{otherwise,} \end{cases} \tag{4}$$

which is rewritten as

$$\varepsilon(V) = \frac{I(X_V; Y) - \sum_{i=1}^{k} I(X_{v_i}; Y)}{I(X_V; Y)} \tag{5}$$

when $I(X_V; Y) \neq 0$. The epistasis $\varepsilon(V)$ of a variable set $V \subseteq \mathcal{V}$ satisfies the following inequality:

$$1 - |V| \leq \varepsilon(V) \leq 1. \tag{6}$$

The epistasis has a positive value when $\xi(V)$ is greater than $\sum_{i=1}^{k} \xi(v_i)$ and it has a negative value when $\xi(V)$ is smaller than $\sum_{i=1}^{k} \xi(v_i)$. The former case means that the corresponding variables interact constructively with each other, and the latter case means that they interact destructively with each other.

The epistasis has a nonnegative value if the universe contains the whole combinations of the alphabets. Formally, if the universe $\mathcal{U}$ of a problem is defined to be $\mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_n$, then the epistasis $\varepsilon(V)$ of a variable set $V \subseteq \mathcal{V}$ is nonnegative, i.e.,

$$0 \leq \varepsilon(V) \leq 1. \tag{7}$$

The equation means that a set of variables always interact constructively in such a fitness function. If we get a negative epistasis from a sampled solution set, then it implies that the solution set was mis-sampled in the case. The epistasis has a zero value if and only if the corresponding variables are conditionally independent given Y, i.e., $p(x_{v_1}, x_{v_2}, \ldots, x_{v_k}|y) = \prod_{i=1}^{k} p(x_{v_i}|y)$ for all $y \in \mathcal{F}$.

2.3 Internal/External Epistasis

A set of disjoint nonempty subsets of a set V is said to be a partition of V if the union of the subsets is V. A partition composed of k subsets is said to be a k-way partition. Based on the epistasis measures shown in Section 2.2, we devise two novel epistasis measures for a partition, *internal epistasis* and *external epistasis.*

The internal epistasis of a partition is defined to be the weighted sum of the epistases of the subsets, where each weight corresponds to the relative significance of a subset.

Definition 1 (Internal Epistasis). *Let $\pi = \{V_1, V_2, \ldots, V_q\}$ be a partition of a variable set $V \subseteq \mathcal{V}, |V| \geq 1$. The internal epistasis $\zeta(\pi)$ of π is defined as follows:*

$$\zeta(\pi) = \sum_{i=1}^{q} \frac{\xi(V_i)\varepsilon(V_i)}{\xi(V)}, \tag{8}$$

which is rewritten as

$$\zeta(\pi) = \sum_{i=1}^{q} \frac{I(X_{V_i}; Y) - \sum_{v \in V_i} I(X_v; Y)}{I(X_V; Y)} \tag{9}$$

when $I(X_V; Y) \neq 0$.

The value of internal epistasis is bounded, which is verified in the following proposition.

Proposition 1. *Let $\pi = \{V_1, V_2, \ldots, V_q\}$ be a partition of a variable set $V = \{v_1, v_2, \ldots, v_k\} \subseteq \mathcal{V}, k \geq 1$. The internal epistasis $\zeta(\pi)$ of π satisfies the following inequality:*

$$q - k \leq \zeta(\pi) \leq q. \tag{10}$$

Proof. Omitted by space limitation [14]. □

The internal epistasis has a nonnegative value if the universe contains the whole combinations of the alphabets.

Proposition 2. *Let $\pi = \{V_1, V_2, \ldots, V_q\}$ be a partition of a variable set $V \subseteq \mathcal{V}, |V| \geq 1$. If the universe $\mathcal{U}$ of a problem is defined to be $\mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_n$, then the internal epistasis $\zeta(\pi)$ of π is nonnegative, i.e.,*

$$0 \leq \zeta(\pi) \leq 1. \tag{11}$$

Proof. Omitted by space limitation [14]. □

The external epistasis of a partition is defined similarly to the epistasis of a variable set. From (2), the significance of a variable set $V \subseteq \mathcal{V}$ is denoted by $\xi(V)$, and the significance of each subset V_i in a partition π of V is denoted by $\xi(V_i)$. We define the external epistasis of π to be the difference between $\xi(V)$ and the summation of all $\xi(V_i)$'s, $V_i \in \pi$.

Definition 2 (External Epistasis). *Let $\pi = \{V_1, V_2, \ldots, V_q\}$ be a partition of a variable set $V \subseteq \mathcal{V}, |V| \geq 1$. The* external epistasis *$\theta(\pi)$ of π is defined as follows:*

$$\theta(\pi) = \begin{cases} \dfrac{\xi(V) - \sum_{i=1}^{q} \xi(V_i)}{\xi(V)} & \text{if } I(X_V; Y) \neq 0 \\ 0 & \text{otherwise,} \end{cases} \tag{12}$$

which is rewritten as

$$\theta(\pi) = \frac{I(X_V;Y) - \sum_{i=1}^{q} I(X_{V_i};Y)}{I(X_V;Y)} \tag{13}$$

when $I(X_V;Y) \neq 0$.

The value of external epistasis is also bounded as in the following proposition.

Proposition 3. *Let* $\pi = \{V_1, V_2, \ldots, V_q\}$ *be a partition of a variable set* $V \subseteq \mathcal{V}, |V| \geq 1$. *The* external epistasis $\theta(\pi)$ *of* π *satisfies the following inequality:*

$$1 - q \leq \theta(\pi) \leq 1. \tag{14}$$

Proof. Omitted by space limitation [14]. □

The external epistasis has a nonnegative value like the internal epistasis if the universe contains the whole combinations of the alphabets.

Proposition 4. *Let* $\pi = \{V_1, V_2, \ldots, V_q\}$ *be a partition of a variable set* $V \subseteq \mathcal{V}, |V| \geq 1$. *If the universe* $\mathcal{U}$ *of a problem is defined to be* $\mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_n$, *then the external epistasis* $\theta(\pi)$ *of* π *is nonnegative, i.e.,*

$$0 \leq \theta(\pi) \leq 1. \tag{15}$$

Proof. Omitted by space limitation [14]. □

The summation of the internal epistasis and external epistasis of a partition is exactly the same as the epistasis of the variable set.

Theorem 1 (Internal/External Epistasis). *The following holds for a partition* π *of a variable set* $V \subseteq \mathcal{V}, |V| \geq 1$.

$$\varepsilon(V) = \theta(\pi) + \zeta(\pi) \tag{16}$$

Proof. Omitted by space limitation [14]. □

The theorem shows that the internal epistasis and the external epistasis of a partition can be interpreted as the intra-partition epistasis and the inter-partition epistasis, respectively. Figure 1 shows an illustration of the relationship between the epistasis, the internal epistasis, and the external epistasis. Since the epistasis is constant with a given variable set, the internal epistasis and the external epistasis are competitive with each other, i.e., the maximality of the internal epistasis implies the minimality of the external epistasis. This property is utilized in Section 3.2.

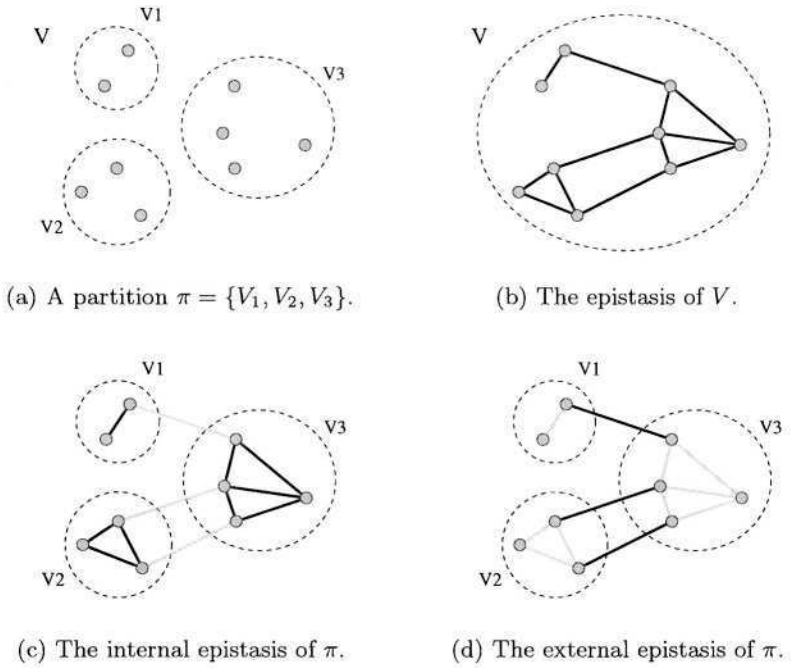


Fig. 1. An illustration of the internal epistasis and the external epistasis of a partition. A partition $\pi = \{V_1, V_2, V_3\}$ of a variable set V is shown denoting the epistatic variable pairs by edge connections.

3 Problem Decomposition

3.1 Decomposable Problem

The decomposability of a problem is formally defined in this section. At first, we define a schema as follows:

Definition 3 (Schema). *Let* $V = \{v_1, v_2, \ldots, v_k\} \subseteq \mathcal{V}$ *be an ordered set. A pair* (V, b) *is said to be a schema if* $b \in \mathcal{A}_{v_1} \times \mathcal{A}_{v_2} \cdots \mathcal{A}_{v_k}$.

The schema "$*10**$" expressed in Holland's notation [15], for example, is represented as ({2,3}, (1,0)).

The conjunction of a number of schemata, whose variable sets are disjoint with each other, is defined to be a schema composed of the union of the variable sets and the union of the assignments.

Definition 4 (Schema Conjunction). *Let* $\pi = \{V_1, V_2, \ldots, V_q\}$ *be a partition of* $V \subseteq \mathcal{V}$, *and* (V_i, b_i) *be a schema for each* $i = 1, 2, \ldots, q$. *A schema* (V, c) *is said to be a conjunction of* (V_i, b_i)*'s if* $c_{V_i} = b_i$ *for all* $i = 1, 2, \ldots, q$.

For example, given two schemata, $(\{1,2\}, (1,1))$ and $(\{4,5\}, (0, 0))$, which are expressed as "11***" and "***00" in Holland's notation, respectively, the conjunction of the two schemata is defined to be $(\{1, 2,4,5\}, (1,1, 0, 0))$, which is expressed as "11*00".

A solution $e \in \mathcal{U}$ is said to be an instance of a schema (V, b) if the solution contains the schema, i.e., $e_V = b$. Given $y \in \mathcal{F}$, a schema is said to be instantiable for y if there exists a solution which is an instance of the schema and whose fitness is y.

Definition 5 (Instantiable Schema). *A schema (V, b) is said to be instantiable for y if there exists a solution $e \in \mathcal{U}$ such that $e_V = b$ and $f(e) = y$.*

An optimization is in a sense the process of finding desirable schemata. Given a partition of the variables, we can individually optimize each subset of the variables if we can guarantee that the conjunction of the optimal schemata corresponding to the subsets is also optimal. In this context, we can define a decomposable problem as follows:

Definition 6 (Schema Independence Condition). *Let $\pi = \{V_1, V_2, \ldots, V_q\}$ be a partition of $\mathcal{V}$. The following statement is defined to be a schema independence condition for π and $y \in \mathcal{F}$.*

"If there exists an instantiable schema for y for each $V_i, i = 1, 2, \ldots, q$, then the conjunction of the schemata is also instantiable for y."

Definition 7 (Decomposable Problem). *Let $\pi = \{V_1, V_2, \ldots, V_q\}$ be a partition of $\mathcal{V}$. A problem is said to be decomposable with π if the schema independence condition is satisfied for $y = \max \mathcal{F}$.*

Unfortunately, we can not always assure of the optimality of the partial solutions obtained from the individual optimizations of the decomposed subproblems. Thus the schema independence condition only for the maximum $y \in \mathcal{F}$ do not guarantee that the problem is possibly solved independently. This is the reason why it is necessary to define the decomposability of a problem with more strong conditions. Hence we define a strongly decomposable problem as follows:

Definition 8 (Strongly Decomposable Problem). *Let $\pi = \{V_1, V_2, \ldots, V_q\}$ be a partition of $\mathcal{V}$. A problem is said to be strongly decomposable with π if the schema independence condition is satisfied for all $y \in \mathcal{F}$.*

The decomposition of a problem in this paper has different meaning from the concept of decomposition in additively decomposed functions (ADFs) [16]. The strong decomposability of a problem implies that the distributions of the variable subsets in the partition are conditionally independent given the fitness value, while the additive decomposability in ADFs means that the fitness function of a problem can be represented as a summation of subfunctions defined on the variable subsets.

3.2 Decomposition Theorem

The Conditional Independence Theorem and the Problem Decomposition Theorem, the main results of this paper, are shown in this section.

Firstly, we show that the strong decomposability of a problem is equivalent to the conditional independence of the variables given Y.

Theorem 2 (Conditional Independence). *Let $\pi = \{V_1, V_2, \ldots, V_q\}$ be a partition of $\mathcal{V}$. Given a problem of which universe $\mathcal{U}$ is defined to be $\mathcal{A}_1 \times \mathcal{A}_2 \cdots \mathcal{A}_n$, the problem is strongly decomposable with π if and only if X_{V_i}'s are conditionally independent given Y, i.e.,*

$$p(x_{\mathcal{V}}|Y = y) = \prod_{i=1}^{q} p(x_{V_i}|Y = y) \tag{17}$$

for all $y \in \mathcal{F}$.

Proof. Omitted by space limitation [14]. □

Secondly, we show that the strong decomposability of a problem with a partition is equivalent to the zero external epistasis of the partition.

Theorem 3 (Problem Decomposition). *Let $\pi = \{V_1, V_2, \ldots, V_q\}$ be a partition of $\mathcal{V}$. Given a problem of which universe $\mathcal{U}$ is defined to be $\mathcal{A}_1 \times \mathcal{A}_2 \cdots \mathcal{A}_n$, the problem is strongly decomposable with π if and only if the external epistasis $\theta(\pi)$ of π is zero.*

Proof. Omitted by space limitation [14]. □

We have the following corollary to Theorem 3.

Corollary 1. *Let $\pi = \{V_1, V_2, \ldots, V_q\}$ be a partition of $\mathcal{V}$. Given a problem of which universe $\mathcal{U}$ is defined to be $\mathcal{A}_1 \times \mathcal{A}_2 \cdots \mathcal{A}_n$, the problem is decomposable with π if the external epistasis $\theta(\pi)$ of π is zero.*

Proof. Omitted by space limitation [14]. □

Since the minimal external epistasis means the maximal internal epistasis by Theorem 1, the strong decomposability of a problem is equivalent to the maximality of the internal epistasis. Consequently, the previous theorems show that the internal epistasis and the external epistasis are capable of quantifying the decomposability of a problem with a given partition of variables.

4 An Example

We tested the proposed measures on a well-known problem, Royal Road function. This example is just for more concrete explanation of the measures and the concepts previously mentioned, not for providing a new optimization method.

Table 1. A decomposition example for the Royal Road function. Given a Royal Road function R, we can obtain three subproblem pairs (R_{11}, R_{12}), (R_{21}, R_{22}), and (R_{31}, R_{32}) from decomposing R with partitions $\pi_1 = \{\{1,2,3,4\},\{5,6,7,8\}\}$, $\pi_2 = \{\{1,2,3,8\},\{4,5,6,7\}\}$, and $\pi_3 = \{\{1,3,5,7\},\{2,4,6,8\}\}$, respectively.

Problem	Schema s_i	Coefficient c_i
	1 2 3 4 5 6 7 8	
R	1 1 * * * * * *	2
	* * 1 1 * * * *	2
	* * * * 1 1 * *	2
	* * * * * * 1 1	2
R_{11}	1 1 * * * * * *	2
	* * 1 1 * * * *	2
R_{12}	* * * * 1 1 * *	2
	* * * * * * 1 1	2
R_{21}	1 1 * * * * * *	2
R_{22}	* * * * 1 1 * *	2
R_{31}	–	–
R_{32}	–	–

The Royal Road function is suitable for this kind of test since it has explicit building blocks.

The Royal Road functions are special functions proposed by Forrest and Mitchell [17] to investigate how schema processing actually takes place inside evolutionary algorithms. To do so, the function was designed to have obvious building blocks and an optimal solution. A Royal Road function is defined as follows:

$$f(x_1, x_2, \ldots, x_n) = \sum_i c_i \delta_i(x_i, x_2, \ldots, x_n) \tag{18}$$

where c_i is a predefined coefficient corresponding to a schema s_i, and $\delta_i : \{0,1\}^n \rightarrow \{0,1\}$ is a function that returns 1 if the solution contains the schema s_i, and returns 0 otherwise. Generally, thecoefficient c_i is defined to be equal to the order of schema s_i.

Table 1 shows decomposition examples for the Royal Road function. In the table, a Royal Road function R, which contains four order-2 building blocks, is decomposed into (R_{11}, R_{12}), (R_{21}, R_{22}), and (R_{31}, R_{32}), respectively, with three different 2-way partitions $\pi_1 = \{\{1,2,3,4\}, \{5,6,7,8\}\}$, $\pi_2 = \{\{1,2,3,8\}, \{4,5,6,7\}\}$, and $\pi_3 = \{\{1,3,5,7\}, \{2,4,6,8\}\}$. It is notable that the subproblems (R_{11}, R_{12}) have more building blocks than (R_{21}, R_{22}), which have more building blocks than (R_{31}, R_{32}). We can solve the decomposed subproblems individually by an exhaustive search algorithm as shown in Table 2.

Table 3 shows the external epistasis $\theta(\cdot)$ and the internal epistasis $\zeta(\cdot)$ of the partitions, and the average fitness of the solutions obtained by joining the mutually exclusive partial solutions in Table 2. We can see that $\theta(\pi_1)$ is less than $\theta(\pi_2)$ and $\theta(\pi_2)$ is less than $\theta(\pi_3)$. By Theorem 3, it follows that the function

Table 2. The partial solutions obtained from solving the Royal Road sub-functions individually.

Subproblem	Partial solution	Subproblem	Partial solution
	1 2 3 4 5 6 7 8		1 2 3 4 5 6 7 8
R_{11}	1 1 1 1	R_{31}	0 0 0 0
R_{12}	1 1 1 1		1 0 0 0
R_{21}	1 1 0 0		0 1 0 0
	1 1 1 0		...
	1 1 0 1		1 1 1 1
	1 1 1 1	R_{32}	0 0 0 0
R_{22}	0 1 1 0		1 0 0 0
	1 1 1 0		0 1 0 0
	0 1 1 1		...
	1 1 1 1		1 1 1 1

Table 3. The epistasis and the average quality of the conjuncted partial solutions of the Royal Road sub-functions.

Partition π	Subproblems	$\epsilon(\mathcal{V})$	$\theta(\pi)$	$\zeta(\pi)$	Quality
$\pi_1 = \{\{1,2,3,4\},\{5,6,7,8\}\}$	(R_{11}, R_{12})	0.712	0.416	0.296	8.000
$\pi_2 = \{\{1,2,3,8\},\{4,5,6,7\}\}$	(R_{21}, R_{22})	0.712	0.524	0.188	4.500
$\pi_3 = \{\{1,3,5,7\},\{2,4,6,8\}\}$	(R_{31}, R_{32})	0.712	0.580	0.132	2.000

R is more decomposable with π_1 than with π_2, and more decomposable with π_2 than with π_3. This is consistent with the fact that π_1 preserves more building blocks than π_2 which preserves more building blocks than π_3. These results also agree with the fact that the average quality corresponding to π_1 is greater than that of π_2, and the average quality corresponding to π_2 is greater than that of π_3. The function R is not strongly decomposable with any of the example partitions since neither of them has external epistasis zero, although it is decomposable with all of them by the schema independence condition for $y = \max \mathcal{F} = 8$.

It is notable that the tests conducted on other combinatorial optimization problems, such as the MAXSAT problem [18], showed consistent results with the case of the Royal Road function.

5 Conclusions

We showed that the epistasis can be factorized into the internal epistasis and the external epistasis, and these new measures provide strong evidences for the decomposability of a problem. The theorems and the experimental results on an example function showed that the proposed measures were well defined and consistent with other properties of the function. We believe that the contribution of this paper is not merely providing new measures of the decomposability but also presenting a way to in-depth understanding about the epistatic behaviors of the

variables in combinatorial optimization problems. Future work includes applying these measures to the design of efficient hierarchical optimization methods.

Acknowledgments. This work was supported by Brain Korea 21 Project. The ICT at Seoul National University provided research facilities for this study.

References

1. E. H. Aarts and J. K. Lenstra. Introduction. In *Local Search in Combinatorial Optimization,* pages 1–17. John Wiley & Sons, 1997.
2. C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Dover Publications, 1998.
3. D. I. Seo and B. R. Moon. A survey on chromosomal structures and operators for exploiting topological linkages of genes. In *Genetic and Evolutionary Computation Conference,* 2003.
4. M. Pelikan, D. E. Goldberg, and F. Lobo. A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications,* 21(1):5–20, 2002.
5. P. Larrañaga and J. A. Lozano. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation.* Kluwer Academic Publishers, 2002.
6. S. L. Lauritzen. Graphical models. Oxford: Clarendon Press, 1996.
7. B. J. Fery. *Graphical Models for Machine Learning and Digital Communication.* Cambridge: MIT Press, 1998.
8. Y. Davidor. Epistasis variance: Suitability of a representation to genetic algorithms. *Complex Systems,* 4:369–383, 1990.
9. D. I. Seo, Y. H. Kim, and B. R. Moon. New entropy-based measures of gene significance and epistasis. In *Genetic and Evolutionary Computation Conference,* 2003.
10. D. I. Seo, S. S. Choi, Y. H. Kim, and B. R. Moon. New epistasis measures based on Shannon's entropy for combinatorial optimization problems. in preparation, 2004.
11. C. R. Reeves and C. C. Wright. An experimental design perspective on genetic algorithms. In *Foundations of Genetic Algorithms 3,* pages 7–22. 1995.
12. C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal,* 27:379–423, 623–656, 1948.
13. T. Cover and J. Thomas. *Elements of Information Theory.* John Wiley & Sons, 1991.
14. D. I. Seo, S. S. Choi, and B. R. Moon. The epistasis and problem decomposition for combinatorial optimization problems. in preparation, 2004.
15. J. Holland. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, 1975.
16. H. Mühlenbein and T. Mahnig. FDA — A scalable evolutionary algorithm for the optimization of additively decomposed functions. *Evolutionary Computation,* 7(4):353–376, 1999.
17. S. Forrest and M. Mitchell. Relative building-block fitness and the building-block hypothesis. In *Foundations of Genetic Algorithms 2,* pages 109–126. 1993.
18. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, 1979.

Clustering with Niching Genetic *K*-means Algorithm

Weiguo Sheng, Allan Tucker, and Xiaohui Liu

Department of Information System and Computing
Brunel University, Uxbridge, Middlesex, UB8 3PH
London, UK
{weiguo.sheng, allan.tucker, xiaohui.liu}@brunel.ac.uk

Abstract. GA-based clustering algorithms often employ either simple GA, steady state GA or their variants and fail to consistently and efficiently identify high quality solutions (best known optima) of given clustering problems, which involve large data sets with many local optima. To circumvent this problem, we propose Niching Genetic *K*-means Algorithm (NGKA) that is based on modified deterministic crowding and embeds the computationally attractive *k*-means. Our experiments show that NGKA can consistently and efficiently identify high quality solutions. Experiments use both simulated and real data with varying size and varying number of local optima. The significance of NGKA is also shown on the experimental data sets by comparing through simulations with Genetically Guided Algorithm (GGA) and Genetic *K*-means Algorithm (GKA).

1 Introduction

Clustering is useful in exploratory data analysis. Cluster analysis organizes data by grouping individuals in a population in order to discover structure or clusters in the data. In some sense, we would like the individuals within a group to be similar to one another, but dissimilar from individuals in other groups. Various types of clustering algorithms have been proposed to suit different requirements. For clustering large data sets, there is a general consensus that partitional algorithms are imperative. Partitional clustering algorithms generate a single partitioning, with a specified or estimated number of clusters of the data in an attempt to recover natural groups present in the data. Among the partitional clustering algorithms, the *k*-means [5] has been popularly used because of its simplicity and efficiency. However, it highly depends on the initial choice of cluster centers and may end up in a local optimum.

A possible way to deal with local optimality of clustering problems is to use stochastic optimization schemes, such as Genetic Algorithms (GAs), which are believed more insensitive to initial conditions. There have been many attempts to use GAs for clustering problems. Roughly, the attempts can be classified as GA approaches such as [12,4,7] and hybrid GA approaches such as [15,9]. In most cases, the above approaches are reported performing well on small data sets with few local optima. However, real clustering problems may involve large data sets with many local optima. On such clustering problems, both GA and hybrid GA approaches can run into problems. First, they have difficulties in consistently identifying high quality solutions mainly because they employ either the Simple GA (SGA) [6], the Steady State GA (SSGA)

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 162–173, 2004.

[16] or their variants, which may suffer from premature convergence to local optima. Second, they are either very expensive or not efficient enough to identify high quality solutions.

To overcome these problems, we propose a method called *Niching Genetic K-means Algorithm* (NGKA). In NGKA, instead of employing SGA or SSGA, modified deterministic crowding is proposed and the computationally attractive *k*-means is incorporated. Our experiments show that NGKA can consistently and efficiently identify high quality solutions of given clustering problems, which involve large data sets with many local optima. Experiments use both simulated and real data with varying size and varying number of local optima. The performance of NGKA is also shown on the experimental data sets by comparing through simulation with Genetically Guided Algorithm (GGA) [4] and Genetic *K*-means Algorithm (GKA) [9].

The outline of the paper is as follows. We briefly review deterministic crowding in section 2. Section 3 provides the details of our proposed NGKA. Data sets employed in this work are described in section 4. Section 5 contains a description of how various parameters for NGKA can be set. Section 6 details the experiments and compares NGKA against GGA and GKA. Lastly, section 7 presents our conclusions and future work.

2 Deterministic Crowding

Different niching genetic algorithms have been developed [3,13,10]. In this work we focus on one niching genetic algorithm known as Deterministic Crowding (DC) [10]. In DC, selection pressure at the selection stage is eliminated by allowing individuals to mate at random with any other individual in the population. After crossover and eventually mutation, each child replaces the nearest parent if it has higher fitness. It is expected DC can maintain the population diversity and permit the GA to investigate many peaks in parallel. On the other hand, it prevents the GA from being trapped in local optima of the search space. In this work, we modify the DC and incorporate the computationally attractive *k*-means for clustering.

3 Niching Genetic *K*-means Algorithm

3.1 Algorithm Overview

To consistently and efficiently identify high quality solutions of given clustering problems, which involve large data sets with many local optima, we propose NGKA as follows. After initialization, one parent p_1 is selected randomly from the population, and its mate p_2 is selected not from the entire population, but from a group of individuals called *Comparing Factor Group* (CFG), picked randomly (without replacement) from the population. The one most similar to p_1 (e.g., the one whose cluster centers encoded in the chromosome are the closest to p_1's) is chosen as mate p_2. This procedure is repeated until *P*/2 parent pairs are selected. Each of these parent pairs is crossed to form not two but only one single offspring, after undergoing mutation and applying one step of *k*-means the single offspring is then paired with a more

similar parent (a parent whose cluster centers encoded in the chromosome are closer to the offspring's), and if the fitness of the offspring is better than its paired parent, its parent is replaced.

The details of NGKA are shown as below. The algorithm is terminated when the stopping criterion is met. The output of the algorithm is the best solution encountered during the evolution.

Step 1. Randomly initialize P sets of k cluster centers using floating-point representation. Constrain the initial values to be within the space defined by the vectors to be clustered. Only valid individuals that have at least one data point in each cluster are considered to be included in the initial population.

Step 2. Calculate *MSE* according to equation (3) for each individual in the initial population and set the fitness value as $f=1/MSE$.

Step 3. Repeat following (a) to (e) until the *stopping criterion* is met.

a) One parent p_1 is selected randomly from the population, and its mate p_2 is selected from CFG (see section 5 for the size setting). The one most similar (determined by the Euclidean distance based on *phenotypic metric*) to p_1 is chosen as mate p_2. This procedure is repeated until $P/2$ parent pairs are selected.

b) Do real *arithmetic crossover* on each paired parent to form not two but only one single offspring with probability of one and then perform *Gaussian mutation* on each feature of the offspring with some low probability.

c) Run k-means one step on the new offspring and update the offspring (i.e. each data point in the data set is assigned to its closest cluster center encoded in offspring's chromosome after which the cluster centers are updated as the centers of mass of the data points that are assigned to the clusters).

d) Compare the offspring with both parents, and paired with a more similar parent.

e) Calculate MSE' according to equation (4) for the offspring and set fitness of the offspring equal to $1/MSE'$. If the fitness of offspring is better than its paired parent, then replace parent.

Step 4. Provide the cluster centers for the terminal population member with the best fitness.

3.2 Representation

In most of the GA clustering applications, the binary, integer or floating-point representation [14,12,2] are commonly used. In the binary and integer representation, binary codes or integers are usually used to represent the membership or permutation of data points and cluster assignment is done explicitly based on the value of binary codes or integers. In most cases, both binary and integer representations suffer from problems of redundancy and context insensitivity with traditional crossover and mutation methods. Furthermore, they are not a scalable representation for clustering large data sets due to the length of the genomes. In this work, we use a floating-point presentation, which represents the cluster centers. Cluster assignment is done implicitly based on distance. In this context, [11] showed that a real-valued representation

moves the problem closer to the problem representation which offers higher precision with more consistent results across replications.

Our representation consists of a vector of $k \times d$ features of real numbers, where d is the number of dimensions in the data and k is the number of clusters. The first d positions represent the d dimensions of the first cluster center, the next d positions represent those of the second cluster center, and so on.

3.3 Crossover and Mutation

The crossover and mutation operators are chosen after a number of experimental trials. Arithmetic crossover [11] and Gaussian mutation perform well and are selected as reproduction operators for NGKA. Traditional arithmetic crossover linearly combines two parent chromosome vectors to produce two new offspring according to the following equations:

$$Offspring1 = a*Parent1 + (1- a)*Parent2, \tag{1}$$

$$Offspring2 = (1 - a)*Parent1 + a*Parent2. \tag{2}$$

Where $a \in [0,1]$ is a random weighting factor. In NGKA, we use $a = 0.5$ as a weighting factor, however, to produce not two but only one single offspring.

After crossover, a very low probability of Gaussian mutation will apply on the offspring. Gaussian mutation adds a unit Gaussian distributed random value to the chosen feature. The new feature value is clipped if it falls outside of the lower or upper bounds of that feature.

3.4 *K*-means Hybridization

K-means is an iterative scheme attempting to minimize the sum of squared Euclidean distances between data points and cluster centers. Let x_i $i=1,2,\ldots n$ be the set of n data points, k, the number of clusters, m_j, the centroid of cluster C_j. Then the algorithm tries to minimize the cost function Mean Square Error (MSE)

$$MSE = \sum_{i=1}^{n} \sum_{j=1, x_i \in C_j}^{k} \left| x_i - m_j \right|^2 . \tag{3}$$

Starting from an initial distribution of cluster centers in the data space, each data point is assigned to the cluster with closest center, after which each center itself is updated as the center of mass of all data points belonging to that particular cluster. The procedure is repeated until convergence. This iterative scheme is known to converge sufficiently fast.

In order to improve computational efficiency dramatically, one step of *k*-means is applied to all new offspring during each generation, after the regeneration step. This is done by assigning each data point to one of the clusters with the nearest centre encoded in the individual's chromosome. After that, the cluster centers encoded in the chromosome are replaced by the mean points of the respective clusters.

3.5 Fitness Function

The fitness calculation of an individual is based on the clusters formed according to the centers encoded in the chromosome. It was defined as $f=1/MSE$, in which MSE is computed according to equation (3), so that maximization of the fitness leads to minimization of MSE.

During our experiments, sometimes the resulting individuals may represent a partitioning with empty clusters (called *illegal individuals*). To penalize illegal individuals, we use the following heuristic proposed in [4]. If a partitioning, defined by its cluster centers, has b clusters with no data points assigned to them and the MSE value for the partitioning evaluates to a value Tot, the new value will be $(b+1)*Tot$. The resulting fitness function will be $f = 1/MSE'$ where

$$MSE' = MSE + b*MSE, \tag{4}$$

$b \in \{0,\ldots,k\}$ is the integer number of empty clusters. The heuristic penalizes illegal partitionings by decreasing their fitness value. This makes them less likely to replace paired parents and less likely to enter the population pool for the next generation.

4 Data Set Description

4.1 Simulated Data

The simulated data created for our study comprises 3,300 data points with nine clusters. The nine clusters are generated according to a spherical bivariate normal distribution with given mean vector $u=(u_x,u_y)$ and standard deviation σ for both x and y according to:

Cluster1: 300 objects $u_x = 1.0$ $u_y = 0.5$ $\sigma = 0.2$
Cluster2: 300 objects $u_x = 2.0$ $u_y = 0.5$ $\sigma = 0.2$
Cluster3: 600 objects $u_x = 5.0$ $u_y = 0.5$ $\sigma = 0.6$
Cluster4: 600 objects $u_x = 1.5$ $u_y = 3.0$ $\sigma = 0.6$
Cluster5: 300 objects $u_x = 4.2$ $u_y = 3.5$ $\sigma = 0.2$
Cluster6: 300 objects $u_x = 5.5$ $u_y = 3.5$ $\sigma = 0.2$
Cluster7: 300 objects $u_x = 3.5$ $u_y = 2.0$ $\sigma = 0.8$
Cluster8: 300 objects $u_x = 3.0$ $u_y = -1.0$ $\sigma = 0.7$
Cluster9: 300 objects $u_x = 3.0$ $u_y = 5.0$ $\sigma = 0.7$

4.2 Subcellcycle Data

The subcellcycle is a subset of the yeast cell cycle data set provided by [1]. The yeast cell cycle data set contains time-course expression profiles for more than 6220 genes, with 17 time points for each gene taken at 10-min intervals covering nearly two yeast cell cycles (160min). This data set is very attractive because a large number of genes contained in it are biologically characterized and have been assigned to different phases of the cell cycle. The subcellcycle data used in this work consists of 384 genes whose expression levels peak at different time points corresponding to the five phases

(early G1, late G1, S, G2 and M) of cell cycle. We expect clustering results to approximate this five-class partitioning.

4.3 Serum Data

This data set is described and used in [8] and corresponds to the selection of 517 genes whose expression vary in response to serum concentration in human fibroblasts. Here, we expect eight clusters from it.

Both gene data sets are normalized so that every gene has an average expression value of zero and a standard deviation equal to one. Normally, to measure the dissimilarity between two genes one tends to choose correlation coefficients which capture the similarity of the "shapes" of two expression profiles, and ignores differences between their magnitudes. However, Euclidean distance metric is used for all results reported here since it has been shown that the correlation coefficient and Euclidean distance are equivalent on a standardized gene expression data set [17].

5 Parameters Configuration

Before running NGKA, there are some parameters that need to be set including the crossover probability, mutation rate, population size, stopping criterion and *Comparing Factor Group* (CFG) size.

The crossover probability (p_c) is set to 100%. Normally one tends to choose a lower value for the crossover probability in order to allow individuals to survive from one generation to the other. However, in NGKA we have found that a crossover rate of 95-100% offers best results, since only better offspring can replace its paired parent so the individuals with highest fitness will survive multiple generations. The mutation probability (p_m) is set at 0.01. Mutation is useful to allow individuals to explore other areas that haven't been explored by the algorithm.

For population size setting, we find NGKA performs well on a generally small population size. However, a too small population would heavily retard the search ability of NGKA. Based on trial runs, we find a population size of 50 for simulated data and subcellcycle data (with 10s to 100s local optima), 75 for serum data (with 100s to 1000s local optima) to be acceptable lower limit, and better performance can be obtained by using a large population size.

Our stopping criterion for NGKA is that the fitness value of the best population member has not changed for n generations, with n=10 being a reasonable choice for NGKA on all the three experimental data sets.

A final variable that needs to be set is the *CFG* size. The idea of using *CFG* is to get a group that will likely contain an individual from the same niche of the first parent. However, an appropriate value should be set to allow both thorough exploration of the search space and competition between different niches. When the group size value equals one, it is basically random selection. As the group size increases there is more possibility of selecting a mate from the same niche as the first parent. For our experimental data sets, the *CFG* size for NGKA has been determined empirically. We use *CFG*=20%**P*, where *P* is the population size.

6 Experiments

In order to be of real use, the solutions supplied by a clustering algorithm must be of high quality (i.e. best known optima) and discovered within a reasonable time. More importantly, to have confidence in the clusters supplied by a clustering algorithm, the algorithm must consistently deliver such high quality solutions. In this respect, we compare the performance of NGKA with the recently proposed GGA and GKA in terms of consistency and efficiency of identifying high quality solutions both on simulated and real data. As well as fitness value, we report the *MSE* value for all experiments. This allows us to compare performance of the different algorithms with similar but different fitness functions. All results reported in this section were obtained on a PC with AMD Athlon 1800 running Windows2000 operation system.

We are primarily interested in clustering problems that involve large data sets with many local optima. In order to determine the exact number of different optima of clustering the above three data sets, one would in principle be required to start a local search algorithm such as *k*-means from every possible initial configuration of the cluster centers. However even if we restrict the experiment to consider only the configurations where each cluster center is initially placed in one of the data points, there would still be too many initial configurations to be tested. Therefore we run 5000 trials of *k*-means using random initial configuration on each of the three data sets. There are 395, 71 and 965 different optima found on simulated data, subcellcycle data and serum data respectively. And more trials lead to more different optima, for example 1324 optima found out of 10,000 trials on serum data, *figure 1* plots all 1324 different optima associated with the final partitionings. The figure also illustrates that *k*-means is very far from being a consistent technique to identify high quality solutions. Since it has been reported in [4] and [9] that GGA and GKA outperform *k*-means, we will not consider *k*-means for further comparison.

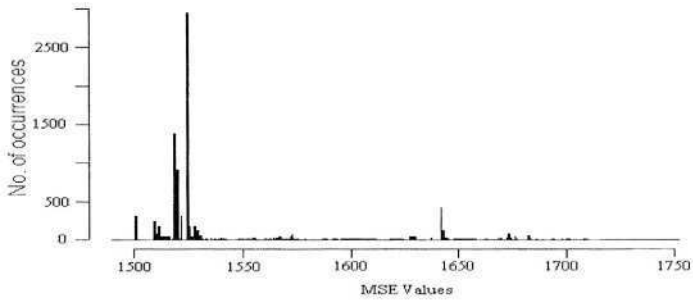


Fig. 1. All of the 1324 different optima with *MSE* value found by applying *k*-means 10,000 trials on serum data

Before comparing the performance of the three algorithms, we give a brief description of the GA used in GGA and GKA. In GGA, a GA based on tournament selection, two-point crossover and flip bit mutation is applied with binary gray coding for clustering. The GA used in GKA is based on roulette wheel selection and a biased mutation with string-of-group-number coding which is one kind of integer represen-

tation discussed in section 3.2. Moreover GKA uses one step of *k*-means instead of the traditional crossover operators, which we think may restrict the GA's search capability. A further investigation is out of the scope of this paper.

To make the comparison between the three algorithms more meaningful, the same stopping criterion (i.e. the fitness value of the best population member has not changed for n generations) is used for all experiments. And a reasonable n value is set to be n=10 for GKA and n=80 for GGA on all the three experimental data sets. The remaining parameters of GGA and GKA are set in *table 1*. The population size of GGA and GKA on the three data sets is identical to that in NGKA. Other parameters (crossover rate, mutation rate and order of tournament) of GGA and GKA are specified according to the original papers for best performance.

Table 1. Population size, crossover rate, mutation rate and order of tournament setting of the three algorithms (GGA, GKA and NGKA) for tests on three experimental data sets (*data1*: simulated data, *data2*: subcellcycle data, *data3*: serum, P is the population size and *bits* is the number of bits in an individual [4])

Parameter	Algorithm								
	GGA			GKA			NGKA		
Population size	Data1	Data2	Data3	Data1	Data2	Data3	Data1	Data2	Data3
	50	50	75	50	50	75	50	50	75
Crossover rate (P_c)	0.9			N/A			1.0		
Mutation rate (P_m)	$1.75/(P\times\sqrt{(bits)})$			0.05			0.01		
Order of tournament	2			N/A			N/A		

To compare performance of the three algorithms in terms of consistency and efficiency of identifying high quality solutions, we recorded the average *MSE* value, standard deviation, distribution of the optima and average run time to identify the high quality solution found from tests on the three data sets. The average *MSE* values and corresponding standard deviations are obtained by generating 20 random starting populations, and running each experiment once with each of the 20 random starting populations. While the average run time to identify the high quality solution is obtained by running each experiment iteratively with each of the 20 random starting populations until a high quality solution is identified. All the results were then averaged over the 20 trials. All sets of experiments use the same 20 random starting populations. *Table 2* lists the average values of *MSE* and standard deviation found from the tests. *Figures 2, 3* and *4* show the distribution of different optima found out of 20 trials on the three data sets respectively. The results of average running time to identify the high quality solution are shown in *table 3*.

The reason for using simulated data set in this work is that the structure is known a priori, thus enabling a better validation of the algorithm proposed here. Furthermore, the bivariate simulated data allows us to graphically display the clustering results. So, here we will examine the experimental results on simulated data first. In terms of consistency, *figure 2* shows that NGKA consistently identifies the best known optimum with *MSE*= 1516.41 in each of 20 trials. However GGA and GKA can only identify

Table 2. Comparing average *MSE* values and standard deviation found by the three algorithms(GGA, GKA and NGKA) on simulated data, subcellcycle data and serum data. The results are averaged over the 20 trials

Algorithm	Simulated data		Subcellcycle data		Serum data	
	MSE	St.dv.	*MSE*	St.dv.	*MSE*	St.dv.
GGA	1573.23	45.89	2361.12	12.15	1511.79	8.28
GKA	1548.32	29.89	2350.22	5.52	1507.65	6.74
NGKA	1516.41	0.0	2345.56	0.0	1500.84	0.11

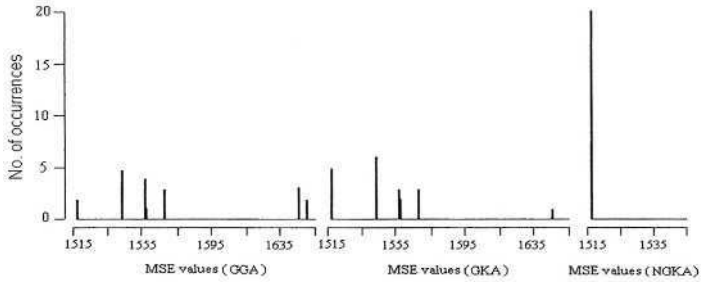


Fig. 2. Distribution of the optima with *MSE* value found for 20 trials of GGA, GKA and NGKA on simulated data

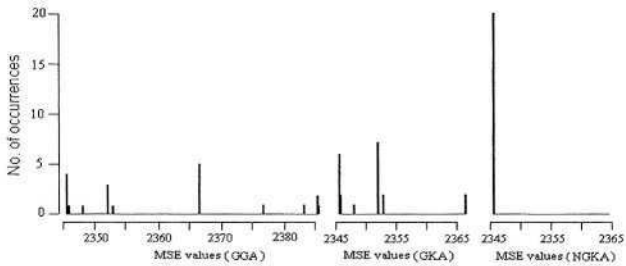


Fig. 3. Distribution of the optima with *MSE* value found for 20 trials of GGA, GKA and NGKA on subcellcycle data

the best known optimum 2 times and 5 times respectively. Two typical local optima that were identified by GGA and GKA are *MSE*= 1543.08 and *MSE*=1556.13. *Figure 5(a)* and *5(b)* show the simulated data and its best known optimum clustering result (different clusters represented by different symbols). The two local optima and their corresponding clustering results are shown in *figure 5(c)* and *5(d)* respectively. We can see that the two local optima cannot succeed in finding the correct clusters, as two clusters at the lower-left corner are joined together (represented by 'Δ' in *figure 5(c)* and '+' in *figure 5(d)*) while one other cluster is split up. In terms of efficiency, *table 3* shows that to identify the best known optimum, by average, GGA needs 6945.2 seconds, which is around 118 times longer than that of NGKA. GKA is quite efficient and takes only 169.2 seconds, however it is still more than 3 times longer than that of NGKA.

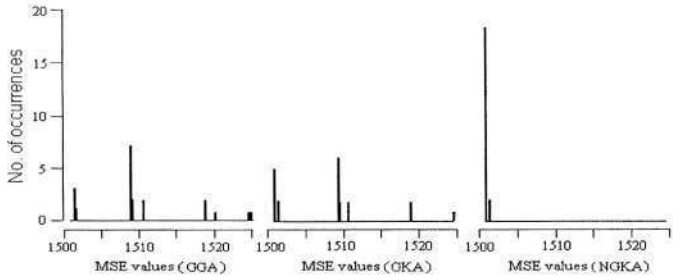


Fig. 4. Distribution of the optima with *MSE* value found for 20 trials of GGA, GKA and NGKA on serum data

Table 3. Comparing average run time to identify the high quality solution found by the three algorithms (GGA, GKA and NGKA) on simulated data, subcellcycle data and serum data. The results are averaged over the 20 trials

Algorithm	Average run time to identify the high quality solution (seconds)		
	Simulated data	Subcellcycle data	Serum data
GGA	6945.2	3805.3	12587.4
GKA	192.7	72.1	281.6
NGKA	58.6	29.7	92.1

Testing on subcellcycle gene expression data (Table 2 and Fig 3), GGA also takes a long time and result in *ave.MSE*=2361.12 with *std.dev*=12.15. Out of 20 trials, it identifies the best known optimum 4 times with *MSE*=2345.56. When applying GKA, the computational efficiency is significantly improved. On average it takes 72.1 seconds to identify the best known optimum and achieves *ave.MSE*=2350.22 with *std.dev*=5.52. However, out of 20 trials, it only identifies the best known optimum 6 times. Experiments using NGKA show that the computational efficiency of identifying the best known optimum is even more improved with only 29.7 seconds. More importantly, the average *MSE* value and standard deviation are improved to *ave.MSE*=2345.56 with *std.dev*=0.0, which means the best known optimum is consistently identified in each of 20 trials. Testing on serum gene expression data (Table 2 and Fig 4), similar results are obtained. Out of 20 trials, NGKA successfully identifies the best known optimum 18 times with *MSE*= 1500.82. However, GGA and GKA can only identify it 3 times and 5 times respectively. Clearly, based on the experiments both on simulated and real data NGKA is the best alternative as it consistently delivers high quality clustering solutions faster.

7 Conclusions and Future Work

In this work, we considered the clustering problem of identifying high quality solutions (best known optima) of given data sets, which involve in large data sets with many local optima. We propose NGKA to solve the problem consistently and efficiently. For clustering of small data sets with few local optima, all GGA, GKA and

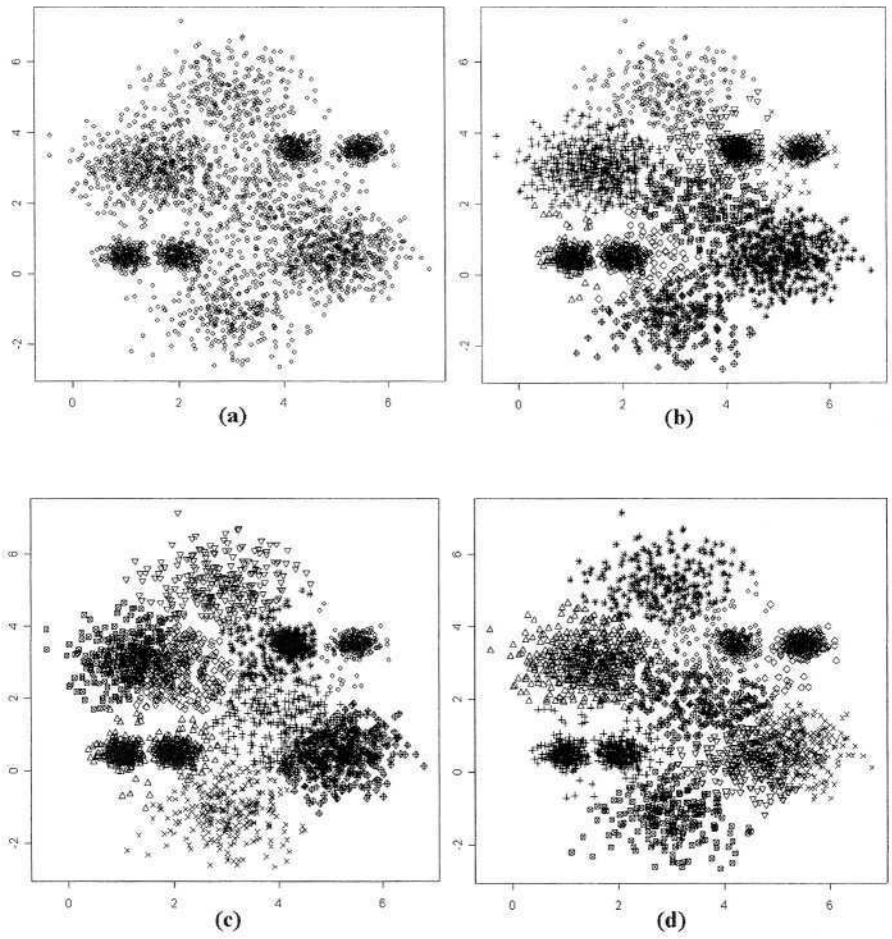


Fig. 5. *(a)* Random generated simulated data. (*b*) Clustering results corresponding to the best known optimum with *MSE*=1516.41. (*c*) Clustering results corresponding to the local optimum with *MSE*=1543.08. (*d*) Clustering results corresponding to the local optimum with *MSE*=1556.13

NGKA perform well. However, as the size of data sets and corresponding number of local optima increase, the picture changes drastically: GGA becomes very expensive and both GGA and GKA have difficulties in consistently identifying high quality solutions. However, our proposed NGKA can identify high quality solutions *faster* and more importantly, it can *consistently* deliver such high quality solutions.

The work presented here represents the first attempt to consistently and efficiently cluster gene expression data sets. Our promising results lead us to believe that NGKA can be extended to a class of real world large clustering problems where other methods have not been appropriate.

Future work will involve the analysis of biological significance of the clustering results found by NGKA based on biological knowledge. A dynamic NGKA clustering which does not require prior specification of the number of clusters will also be investigated, involving perhaps a fitness function which maximizes both the homogeneity within each cluster and the heterogeneity among clusters.

References

1. Cho, R.J. et al.: A Genome-Wide Transcriptional Analysis of the Mitotic Cell Cycle. Molecular Cell 2(1) (1998) 65-73
2. Cucchiara, R.: Genetic Algorithms for Clustering in Machine Vision. Machine Vision and Applications, Vol. 11, No.l (1998) 1-6
3. Goldberg, D.E. and Richardson, J.: Genetic algorithms with sharing for multimodal function optimization. Proceeding of the 2nd Int. Conference on Genetic Algorithms. Hillsdale, New Jersey (1987) 41-49
4. Hall, L.O., Ozyurt, B. and Bezdek, J.C.: Clustering with a Genetically Optimized Approach. IEEE Transactions on Evolutionary Computation, Vol. 3, No. 2 (1999) 103-112
5. Hartigan, J.A. and Wong, M.A.: A *k*-means clustering algorithm. Applied Statistics, 28 (1979)100-110
6. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor (1975)
7. Eduardo, R. Hruschka and Nelson, FF Ebecken: A genetic algorithm for cluster analysis. Intelligent Data Analysis 7 (2003) 15-25
8. Iyer V.R. et al.: The Transcriptional Program in the Response of Human Fibroblasts to Serum. Science, 283 (1999) 83-87
9. Krishna, K. and Murty, M. Narasimha: Genetic *K*-means Algorithm. IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics, Vol. 29, No. 3 (1999)
10. Mahfoud, S.W.: Niching methods for genetic algorithms. Ph.D. dissertation, Univ. of Illinois, Urbana-Champaign (1995)
11. Michalewicz, Z.: Genetic algorithms + Data structure = Evolution programs. 3rd edn. Springer-Verlag, Berlin Heidelberg New York (1996)
12. Murthy, C.A. and Chowdhury, N.: In search of optimal clusters using genetic algorithms. Pattern Recognition Letters, 17 (1996) 825-832
13. Petrowski, A.: A clearing procedure as a niching method for genetic algorithms. Proceeding of IEEE Int. Conf. Evolutionary Computation (1996) 798-803
14. Sarkar, M., Yegnanarayana, B. and Khemani, D.: A Clustering Algorithm Using an Evolutionary Programming-based Approach. Pattern Recognition Lett. 18 (1997) 975-986
15. Scheunders, P.: A genetic *c*-means clustering algorithm applied to color image quantization. Pattern Recognition, Vol. 30, No. 6 (1997) 859-866
16. Syswerda, G.: A study of reproduction in generational and steady-state genetic algorithms. In Foundations of Genetic Algorithms, Morgan Kauffmann Publishers (1991) 94-101
17. Yeung, K.Y.: Clustering analysis of gene expression data. PhD Thesis, University of Washington (2001)

A Comparison of Genetic Programming and Genetic Algorithms in the Design of a Robust, Saturated Control System

Andrea Soltoggio

Department of Computer and Information Science
Norwegian University of Science and Technology
N-7491, Trondheim, Norway
soltoggi@stud.ntnu.no

Abstract. The design of a robust control system for a specified second order plant is considered using three different approaches. Initially, a control system evolved by a genetic programming algorithm is reproduced and analysed in order to identify its advantages and drawbacks. The automatic design technique is compared to a traditional one through the analysis of the constraints and performance indices obtained by simulation. A set of unspecified control constraints explored by the GP search process is found to be the cause of a better performance. Hence, giving a better constraints specification, a genetic algorithm is used to evolve an alternative controller. A PID structure is used by the GA to produce and tune the controller. Simulations show a significant gain in performance thanks to a more aggressive and complete exploration of the search space within the constraints. The effectiveness of the two methods compared to the traditional approach is discussed with regard to performance, complexity of design and computational viability.

1 Introduction

In recent years, evolutionary computation has been applied to several control engineering problems. While weaknesses and strengths of traditional approaches of control system design are well known to experts in the field, evolutionary computation offers a designing and tuning tool that is not well investigated with regard to reliability, effectiveness and usability.

The new evolution based methods proposed by several scientists [4] are still not fully considered by the traditional control field. The proposed methods often lack mathematical proofs of stability, guarantees of reliability and applicability of the results. A better knowledge of the characteristics of evolutionary algorithms in control engineering could help the synthesis of more usable and feasible control systems and allow evolutionary computation to gain the status of an effective tool in control system design.

There are several weaknesses and difficulties in the design of a suitable evolutionary algorithm for control synthesis. The determination of a unique fitness

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 174–185, 2004.

value is typically a multi-objective optimization problem [13] and requires particular attention during the setting of initial parameters: a wrong choice can result in poor outcomes or failure of the search. The fidelity in the simulation of the plant is also a key factor, often affected by unknown parameters, unknown plant dynamics or noise. The difference between the real plant and the simulated plant is reflected in the evolved controller that loses reliability and performance once implemented. Therefore, the identification of constraints and characteristics of the desired controller is a decisive factor for a good design.

In this paper, the characteristics of different control systems for a robust[1], linear SISO[2] control problem are discussed. The control problem is presented in a textbook of control engineering [3, pages 697-700] and it is used as a bench mark for the analysed controllers. Initially, a GP-evolved control system presented in [11,12] is reproduced, analysed and simulated in order to highlight its advantages and flaws. A comparison is carried out with respect to the traditional design proposed in [3]. The unspecified limit of the derivative of the control variable and an unlimited bandwidth from the feedback signal to the output result in very high load disturbance suppression. The GP controller gives better performance also using a more intense control action that results in saturated control. Thus, the comparison is not relevant. An alternative controller for the same problem was evolved using a genetic algorithm. The controller structure utilizes a prefilter, a PID core and filters on the feedback and on the derivative. The high gain in performance shown by the GA controller is justified by the use of saturated, bang-bang control. The GA computation explored the search space more effectively and produced a controller that brings to the limit all the constraints. The GA approach is also found less computationally expensive and able to cover different control problems.

2 Methods

2.1 Representation of Controllers and Plant

The results presented in this paper are obtained by the simulation of the controlled systems implemented using Matlab, the Matlab Control System Toolbox and Simulink. The choice of the MathWorks Inc. software is due to the completeness of the available tools for control system engineering. The block diagram of a general controlled system is shown in figure 1.

A controller can be described as the compound of linear and nonlinear components. Linear components can be expressed by transfer functions [3,21]. The design of linear control benefits of well known mathematical theories and methodologies [3,20,21,24,25] . Nonlinear components such as saturation or rate limiter have to be expressed by special blocks or functions. They increase the complexity of the design and justify the use of simulation and evolutionary algorithms.

[1] The parameters of the plant are supposed to be time-varying between certain ranges to guarantee stability and steady performances.

[2] Single Input, Single Output.

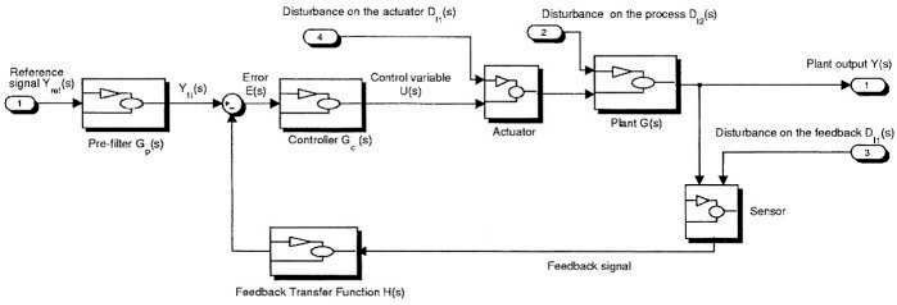


Fig. 1. General model of a controlled system

The plant to be controlled is expressed by the transfer function

$$G(s) = \frac{K}{(\tau s + 1)^2} \quad , \tag{1}$$

where K and τ are considered varying between the values $1 \leq K \leq 2$ and $0.5 \leq \tau \leq 1$ to obtain robust control. The simulation of the controlled systems is carried out for the four states corresponding to the four combinations of the values $K = 1{,}2$ and $\tau = 0.5, 1$. The measurements were obtained applying a step reference signal from 0 to 1 Volts.

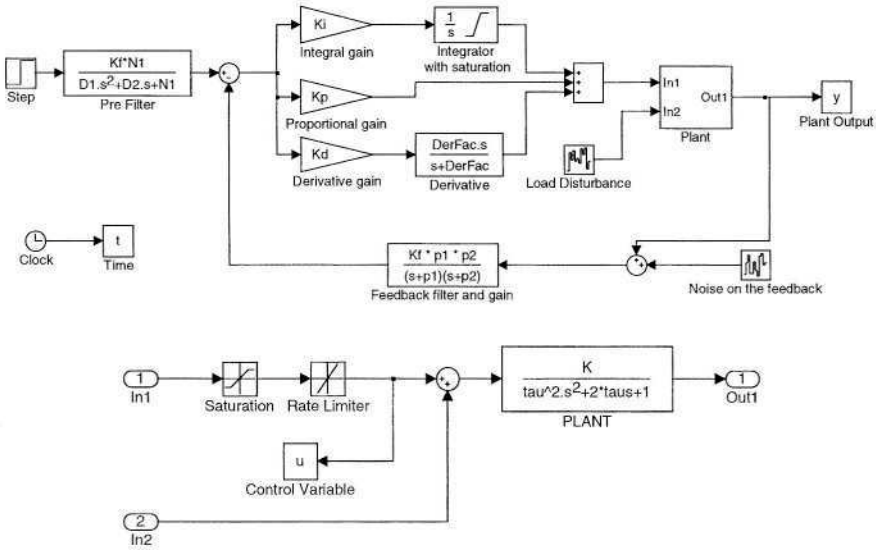


Fig. 2. Simulink model used by the genetic algorithm

2.2 Constraints and Performance Indices for a Control System

The distinction between constraints and performance indices is often blurred. Generally, a constraint is a characteristic of the controlled system that should be kept within specified boundaries. A performance index is a characteristic of the controlled system that should be minimized or maximized. The indices and constraints used in this paper are listed here with a brief description.

- Overshoot: is the amount the system output response proceeds beyond the desired value when applying a step to the reference signal.
- Rise time: is the time taken by the output to rise from 10% to 90% of the input amplitude.
- Settling time: is the time required for the system output to settle within a certain percentage of the input amplitude.
- Maximum $u(t)$: is the maximum value reachable by the control variable.
- Maximum $\dot{u}(t)$: is the maximum derivative reachable by the control variable.
- Bandwidth: is a frequency domain index that describes the reactivity of the system in following the reference signal and the sensitivity to feedback high frequency noise.
- ITAE: is the Integral of Time-weighted Absolute Error between the plant-output and the reference signal. It is the main performance index used in this experiment.
- Load disturbance deviation: is the maximum deviation of the plant-output from the reference signal when a disturbance is applied to the plant-input.

For the control problem examined, an overshoot less than 2% was considered. In [11,12] a saturation limit was imposed to 40 Volts. The derivative of the control variable was unlimited for the GP controller and limited to 10.000 Volts/sec for the GA controller.

2.3 Design Methods

The controllers presented and compared in this paper are designed using three different approaches.

The genetic programming approach has been used in [11,12] to design from scratch a controller for the the plant of equation (1). Several constraints and performance indices were used to build a fitness function. The genotype was a tree-coded controller mapped into a SPICE code for the simulation of the electric circuit. The controller proposed in [11,12] has been reproduced and simulated in this experiment.

The traditional design method is proposed in [3] and makes use of a PID controller with pre-filter to minimize the ITAE index. The design uses a parameter ω to set the intensity of the control variable and achieve optimum non-saturated control with respect to the ITAE index.

The genetic algorithm approach, implemented as part of the experiment presented in this paper, uses a PID controller with 11 parameters for tuning a

pre-filter, the PID parameters, a filter on the derivative and a filter on the feedback signal. Figure 2 shows the Simulink model used by the genetic algorithm to optimize the parameters. The figure shows also the position of the 11 parameters with the exception of *IntLim* which is embedded in the integrator block.

In spite of the proposed fixed structure, the number of parameters and the different values that they can assume allow the evolutionary computation to accentuate or disable parts of the controller. Thanks to this characteristic, the method goes beyond the tuning of the three PID parameters.

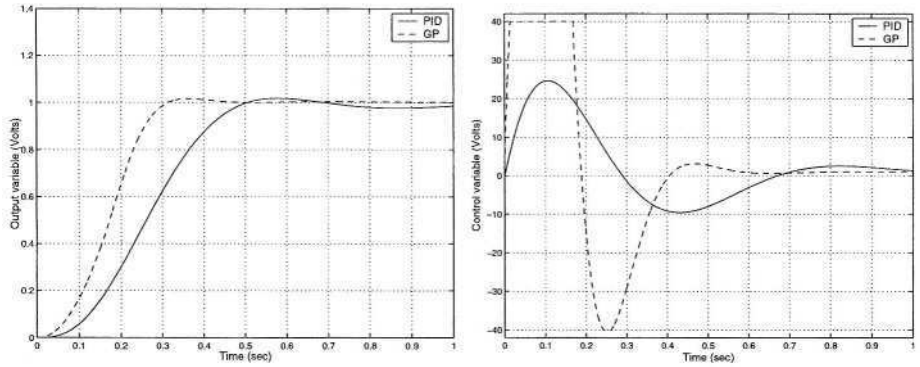


Fig. 3. Plant-outputs (left) and control variables (right) for the PID and GP controllers, plant parameters: $K = 1, \tau = 1$

3 Results

3.1 Simulation Results of the GP and PID Controllers

The simulation of the GP controller, compared to the standard PID, shows that the GP controller uses the control variable in a more intensive way than the PID. It makes use of saturated control and higher varying rate of the control variable. Figure 3 shows the output variable and the control variable for both the controllers. It is evident that use of nonlinear saturated control helps the GP controller to achieve better performance. Besides, the GP controller uses a second derivative that produces an infinite bandwidth from the feedback signal to the output. This fact, with the unlimited derivative of the control variable, gives a potential infinite load disturbance suppression but is not applicable to real systems. For the simulation, the derivative was implemented with an embedded low-pass filter, the same shown in figure 2, in order to obtain a finite bandwidth. From the analysis, it is evident that the controller presented in [11, 12] has substantially different characteristics from the PID and is therefore not comparable.

In a second simulation, the standard PID controller was tuned for a stronger control action, setting a tuning parameter ω to 16 instead of 8 as described in

[3, pages 697-700]; additionally, a limit on the integral was imposed to 8 Volts and a gain of 3 was added to the feedback signal. The gain on the feedback was added to increase the bandwidth of the system and obtain a better load disturbance suppression for a unitary step at the plant input. The roughly tuned controller, compared to the GP controller, obtained better performance under all the considered indices. Table 1 shows the simulation results for $K = 1$ and $\tau = 0.5$.

Table 1. Simulation results for the PID, GP and new PID controllers for $K = 1, \tau = 0.5$

	(PID)	(GP)	(new PID) Characteristic
Overshoot	0.3%	0.4%	1% limited
Rise time (ms)	391	239	*210* to minimize
Settling time (ms)	629	417	*326* to minimize
ITAE $(mVolts \cdot sec^2)$	49.0	19.8	*13.5* to minimize
Load disturbance deviation $(mVolts)$	6.0	0.64	*0.42* to minimize
Maximum $u(t)$ $(Volts)$	8.6	19.4	36.0 limited
Maximum $\dot{u}(t)$ $(Volts/sec)$	460	1927	8761 unspecified/free
Bandwidth Y/Y_{fil} (rad/sec)	57.6	3070	435 unspecified/free

It was observed from the simulations that both the GP controller and the new PID bring the control variable to saturation when $K = 1$ and $\tau = 1$. That is when the plant has the lowest gain *(K)* and the longest time constant (τ) and needs the strongest control action. For the other three combinations of the parameters, the system response does not change significantly and the control variable remains under the saturation limit: the controllers use saturated control only in one fourth of cases. Figure 4 shows the plant-output and the control variable of the GP controlled system for the states ($K = 1$, $\tau = 0.5$), ($K = 2$, $\tau = 1$) and ($K = 2$, $\tau = 0.5$) .

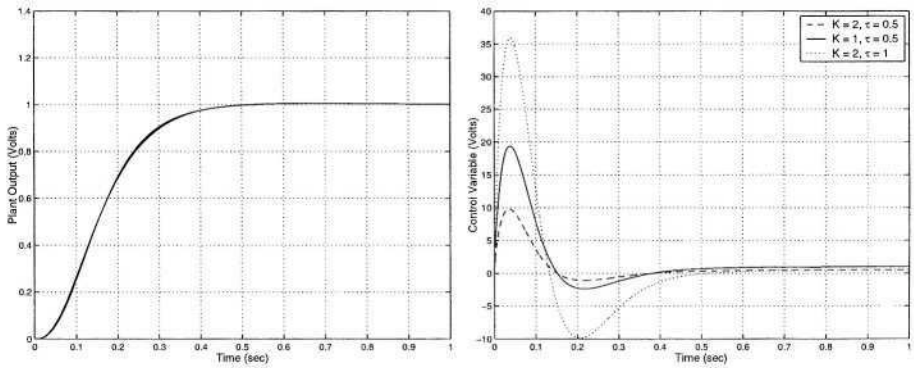


Fig. 4. Plant-outputs (left) and control variables (right) for the GP controller

Table 2. Best individual of the GA controller for noise-free system

Param.	Value	Function	Param.	Value	Function
N1	195.96	Pre-Filter numerator	Kf	3.2033	Feedback gain
D1	0.1744	Pre-Filter denominator 1	IntLim	4.7595	Limit on the integral
D2	7.7851	Pre-Filter denominator 2	DerFac	1613.8	Pole for the derivative filter
Kp	273.78	PID proportional action	p1	978.97	Pole 1 for the feedback filter
Ki	999.21	PID integral action	p2	1543.9	Pole 2 for the feedback filter
Kd	16.988	PID derivative action			

3.2 Simulation Results of the GA Controller

The GA controller was produced in several similar versions by different runs of the algorithm. Preliminary runs during the development of the application were used to identify good settings of the GA parameters. Given the stochastic nature of GAs, the final experiment was carried out by running the program 50 times. To evolve the controller, the GA computation took an average of 34 generations. The first generation took approximately 50 seconds to be evaluated.

The process ran on a laptop with processor AMD Athlon 2400+, 512Mb RAM and Windows XP as operating system.

In a first run the system was simulated without load and feedback disturbances. The population was randomly initialized and seeded with the parameters of the PID controller.

Preliminary experiments proved that the algorithm was able to reach, in a longer time, equivalent solutions without seeding the initial population. However, the quality of the seeds and their initial distribution strongly affect the number of generations required to reach the solution.

The fitness function was composed by a sum of the ITAEs of the four system responses, a penalty for overshoots greater than 2% and a penalty for a spiked or oscillating control variable to reduce the influence of feedback noise and instability. The additional constraint of 10.000 Volts/sec for the derivative of the control variable was added to make the controller applicable to real control problems. The population was composed of 300 individuals. Selection was based on a tournament within groups of 10 individuals. Two degrees of uniform distributed mutation were applied to both accelerate the initial search and finely tune the parameters. Mutation was applied to 30% of the population. A vector of likely fitness improvement was applied to one third of mutated individuals: the vector was calculated taking the difference of two individuals' genotypes where the first one had better fitness than the second one, repeating the operation all over the population and computing the average. This vector provided an indication of a likely fitness improvement and increased the speed of the search by 2 to 5 times[3]. Crossover was applied to 60% of individuals, combining random parameters of

[3] The increase in speed is strongly dependent on the initial conditions given by the quality of the seeds and their distribution in the search space. This data obtained by preliminary runs could be the central issue of a further study regarding the effectiveness of the method.

two individuals chosen from two different groups. Elitism was applied by copying the best individual of each group into the next generation. The best individual of the run is characterized by the values in table 2.

Table 3. Best individual of the GA controller for a system with noise on the feedback

Param.	Value	Function	Param.	Value	Function
N1	119.8	Pre-Filter numerator	Kf	0.95	Feedback gain
D1	0.076	Pre-Filter denominator 1	IntLim	1.29	Limit on the integral
D2	4.59	Pre-Filter denominator 2	DerFac	90.02	Pole for the derivative filter
Kp	112.6	PID proportional action	p1	786.3	Pole 1 for the feedback filter
Ki	11.75	PID integral action	p2	295.3	Pole 2 for the feedback filter
Kd	13.55	PID derivative action			

During the computation, some initial parameters were adapted to adjust and direct the multi-objective search. In particular, the weights of the ITAE values, initially set to 1, appeared to be unbalanced as soon as the computation reached saturated control, giving better performance for the plants with higher gain and lower time constant. The nonlinearity in the controlled system was being used by the genetic algorithm to increase the performance using the maximum control action allowed by rate limit and saturation. Hence, the system response gets faster as the system gets more reactive. The results are shown in figure 5. The control variable shows that the computation reached a complete bang-bang control, where the upper and lower saturation limits are reached using the maximum varying rate in order to obtain the fastest plant response. Bang-bang control provided minimum ITAE, settling time and rise time and was chosen as manual termination criterion.

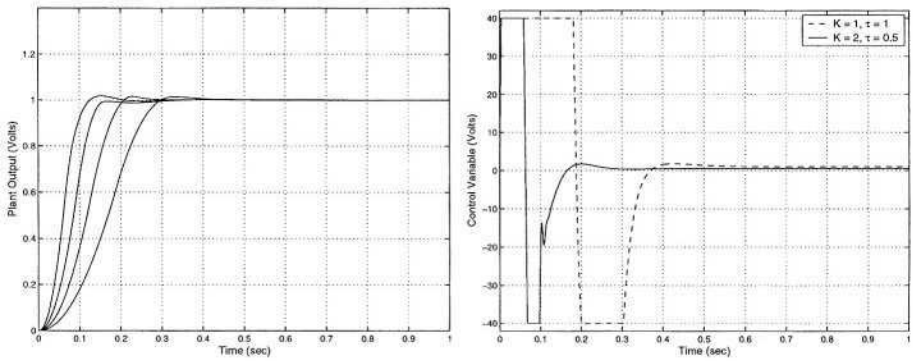


Fig. 5. Plant-outputs (left) and control variables (right) for the GA controller. In the left graph, the fastest response corresponds to $K = 2, \tau = 0.5$, the slowest to $K = 1, \tau = 1$

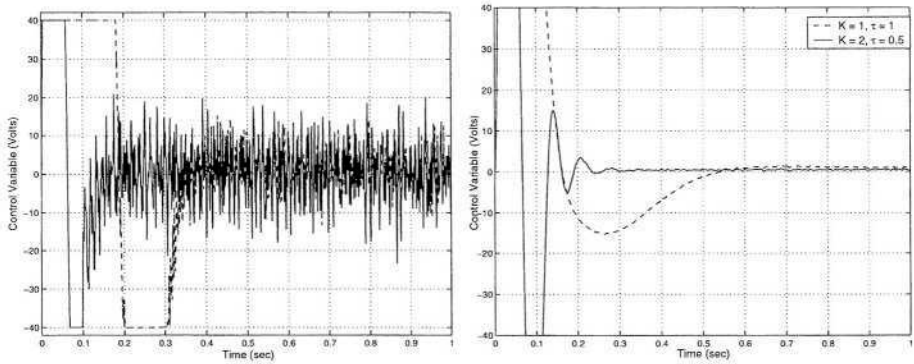


Fig. 6. Control variables at the first (left) and last (right) generation for the run with noise

In a second run, a disturbance to the feedback signal was applied. The disturbance was generated by the white band-limited noise block of the Simulink library with power $0.1nW$ and sampling time $1ms$. Randomized seeds of the best individual of the first run and the original PID controller were chosen to initialize the population. The control variable at the first generation undergoes extreme variations due to the sensitivity to the noise added to the feedback signal. This behaviour of the control variable might damage the plant or cause fast *wear and tear* of the mechanical part of the actuator. Hence, the first target is to make the control variable as smooth as possible.

To follow the optima in the dynamic fitness landscape that changes when introducing noise, the heuristic given by the vector of likely fitness improvement provided a high percentage of best individuals.

A final solution found in the second run is reported in table 3. Figure 6 shows the control variable at the first and last generation.

The GP controller was designed only for noise-free signals. The GA controller has approximately the same performance as the GP controller for $K = 1, \tau = 1$. For $K = 2, \tau = 0.5$, however, the GA controller showed considerable improvements. Figure 7 shows a comparison between the responses of two controllers for $K = 2$, $\tau = 0.5$. The rise time of 243 *ms* using the GP controller decreased to 75 *ms* using the GA controller; the settling time decreased from 419 *ms* to 128 *ms.* Finally, the ITAE recorded by the GA controller is 2.8 $mVolts \bullet sec^2$ versus 19.9 $mVolts \bullet sec^2$ of the GP controller. Table 4 reports the performance of the GA controller.

4 Discussion

The analysis and simulation of the GP and PID controllers outlined several different characteristics of the two controllers. The use of saturated control, not specified as a constraint in [11,12], allowed the evolutionary computation to

Table 4. Simulation results of the GA controller for the system without noise

	$(K = 2, \tau = 0.5)$	$(K = 1, \tau = 1)$	Characteristic
Overshoot	2%	1.9%	limited
Rise time (ms)	75	181	to minimize
Settling time (ms)	128	298	to minimize
ITAE $(mVolts \cdot sec^2)$	2.8	17.2	to minimize
Load disturbance deviation $(mVolts)$	2.8	2.6	to minimize
Maximum $u(t)$ $(Volts)$	40	40	limited
Maximum $\dot{u}(t)$ $(Volts/sec)$	10000	10000	limited
Bandwidth Y/Y_{fil} (rad/sec)	430	77.8	unspecified/free

improve the performance of the standard PID. However, saturated control can be used only in particular control problems. The nonlinearity and the heavy use of the actuator make the controller unsuitable for most industrial applications.

The tuning of a new PID and the synthesis of the GA controller were done considering a reduction of the set of control problems to the one where saturated control is applicable.

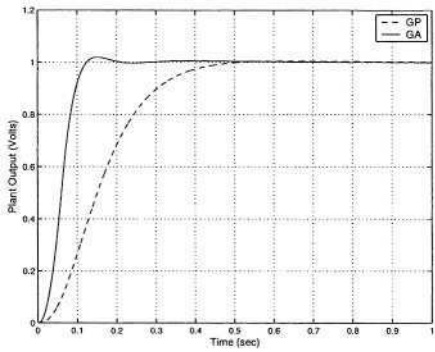


Fig. 7. Comparison of plant-outputs for the GP and GA controllers

The design cost is a decisive factor. The PID tuning requires few manual calculations as explained in [3, pages 697-700]. The GP controller was synthesized by a parallel computer architecture of sixty six 533MHz elements that took 44.5 hours [11,12]. Besides, the setting of suitable initial parameters of the evolutionary computation and the design of an effective fitness calculation require additional time and skill. The choice of wrong parameters can lead to poor results or failure [4]. On the other hand, the automatic synthesis for the GP controller, as stated in [12], does not require knowledge in control theory and it is free to evolve a different structure for each kind of control problem. However, lack of knowledge in control theory favors the production of inapplicable controllers because of simulation flaws like unspecified constraints, easily identified

by an experienced control engineer but greedily explored by the evolutionary computation.

The synthesis of a GA controller requires knowledge in control theory in order to set the proper controller architecture. However the GA computation does not only tune parameters but has some freedom to utilize particular elements in the architecture only when they are needed. This fact suggests the possibility of using a more complex structure, without limiting to PID control, and allowing the GA to shape the optimal sub-structure that better fits the present control problem. In the first run, when the system was simulated without disturbances, the low-pass filters were automatically disabled by placing poles at high frequencies. Conversely, when feedback disturbance was applied, the filters were tuned to play an important role in filtering high frequency noise. The feedback gain was also automatically lowered.

The automatically driven search forward area of likely good fitness reflects the human attitude to move in the direction that gives a likely improvement. It can easily follow the movements of a dynamic fitness landscape and here it is proven to be a decisive factor in speed and effectiveness.

From the simulation results, the dramatic improvements recorded by the GA approach qualify the method to optimise solutions for control problems with high performance requirements.

The computational aspect makes the GA approach feasible. The time of the magnitude of one or a few hours on a single machine, depending on the complexity of the fitness calculation and availability of good seeds, makes the method attractive for several applications. With an increment of computational power and a plant with a long time constant, it would be possible to apply the method to online optimisation or synthesis of adaptive control systems.

References

1. W. Banzhaf, P. Nordin, R. E. Keller, F. D. Francone: *Genetic Programming - An Introduction,* Morgan Kaufmann, San Francisco, CA and dpunkt, Heidelberg (1998)
2. R. C. Dorf, R. H. Bishop: *Modern Control System Analysis & Design Using Matlab & Simulink,* Seventh Edition, Addison Wesley Longman, Inc. Menlo Park California (1997)
3. R. C. Dorf, R. H. Bishop: *Modern Control Systems,* Ninth Edition, Upper Saddle River. N.J.: Prentice Hall (2001)
4. P. J. Fleming, R. C. Purshouse: *Evolutionary algorithms in control system engineering: a survey.* Control Engineering Practice, Vol 10. 2002, (2002) Pages 1223-1241
5. G. J. Gray, D. J. Murray-Smith, Y. Li, K. C. Sharman, T. Weinbrenner: *Nonlinear model structure identification using genetic programming.* Control Engineering Practice, Vol 6. 1998. (1998), Pages 1341-1352
6. M. Jamshidi, L. dos Santos Coelho, R. A. Krohling, P. J. Fleminig: *Robust Control System with Genetic Algorithms,* CRC Press (2002)
7. J. R. Koza: *Genetic Programming: On the Programming of Computers by Means of Natural Selection,* MIT Press: Cambrigde, MA (1992)

8. J. R. Koza: Introduction to genetic programming. In Kinner, Kenneth E. Jr (editor). *Advances in Genetic Programming,* Cambridge, MA, The MIT Press. Pages 21-45. Chapter 2 (1994)
9. J. R. Koza: *Survey of Genetic Algorithms and Genetic Programming* Procedings of 1995 WESCON Conference. Piscataway, NJ: IEEE. (1995) 589 - 594
10. J. R. Koza, Forrest H. Bennet, David Andre: *Method and Apparatus for Automated Design of Complex Structures Using Genetic Programming* U.S. Patent 5,867,397. Issued Feb 2. 1999
11. J. R. Koza, Martin A. Keane, Jessen Yu, Forrest H. Bennett III, William Mydlowec: *Automatic Creation of Human-Competitive Programs and Controllers by Means of Genetic Programming.* Genetic Programming and Evolvable Machines, 1, (2000) Pages 121-164.
12. J. R. Koza, Martin A. Keane, Jessen Yu, Forrest H. Bennet, William Mydlowec: *U.S. Patent 6,564,194. Method and Apparatus for Automatic Synthesis Controllers.* Issued May 13. 2003
13. G. P. Liu, J. B. Yang, J. F. Whidborne: *Multiobjective Optimisation and Control* Research Studies Press LTD. Baldock, Heartfordshire, England. (2003)
14. G. P. Liu, S. Daley: *Optimal-tuning PID control for industrial systems.* Control Engineering Practice, Vol 9, 2001. Pages 1185-1194
15. The Math Works Inc.: *Using the Control System Toolbox* Version 5, Online only, Revised for Version 5.2 (Release 13). July 2002
16. The MathWorks Inc.: *Using Simulink* Version 5, Fifth Printing, Revised for Simulink 5 (Release 13). July 2002
17. The MathWorks Inc.: *Simulink Reference* Version 5, Fifth Printing, Revised for Simulink 5 (Release 13). July 2002
18. Z. Michalewics: *Genetic Algorithms + Data Structures = Evolution Programs* Third, Revised and Extended Edition, Springer. (1996)
19. T. Morimoto, Y. Hashimoto: *AI approaches to identification and control of total plant production systems.* Control Engineering Practice, Vol 8. 2000. Pages 555-567
20. Chester L. Nachtigal: *Instrumentation and Control, Fundamentals and applications* Wiley Interscience Publication, John Wiley & Sons, Inc (1990)
21. Katsuhito Ogata: *Modern Control Engineering,* Third Edition, Upper Saddle River. N.J.:Prentice Hall (1997)
22. C. Vlachos, D. Williams, J.B. Gomm: *Solution to the Shell standard control problem using genetically tuned PID controllers.* Control Engineering Practice, Vol 10. 2002, Pages 151-163
23. P. Wang and D. P. Kwok *Optimal design of PID process controllers based on genetic algorithms* Control Engineering Practice, Volume 2, Issue 4, August 1994, Pages 641-648
24. K. J. Åström, T. Hägglund: *The future of PID control.* Control Engineering Practice, Vol 9, 2001. Pages 1163-1175
25. K. J. Åström, T. Hägglund: *PID Controllers: Theory, Design, and Tuning* Second Edition. Research Triangle Park, N.C (1995)

Upper Bounds on the Time and Space Complexity of Optimizing Additively Separable Functions

Matthew J. Streeter

Computer Science Department and
Center for the Neural Basis of Cognition
Carnegie Mellon University
Pittsburgh, PA 15213
matts@cs.cmu.edu

Abstract. We present upper bounds on the time and space complexity of finding the global optimum of additively separable functions, a class of functions that has been studied extensively in the evolutionary computation literature. The algorithm presented uses efficient linkage discovery in conjunction with local search. Using our algorithm, the global optimum of an order-k additively separable function defined on strings of length ℓ can be found using $O(\ell \ln(\ell) 2^k)$ function evaluations, a bound which is lower than all those that have previously been reported.

1 Introduction

A large amount of evolutionary computation research deals with formally defined classes of problems intended to capture certain aspects of real-world problem difficulty. One prominent example of such a class of problems is N-K landscapes [5]. Finding the global optimum of such functions has been shown to be an NP-complete problem for $K \geq 2$ [10, 13]. Another widely studied problem class is additively separable functions, defined by

$$f(s) = \sum_{i=1}^{m} f_i(s)$$

where each subfunction $f_i(s)$ depends on at most k string positions, and the number of subfunctions that depend on a particular position j is at most one. This class of problems plays a central role in the study of *competent* genetic algorithms [1], and is the focus of this paper. We will assume we are seeking the maximum of an additively separable function of order k defined on strings of length ℓ.

Optimizing such functions is primarily a problem of *linkage discovery:* determining which pairs of positions belong to the same subfunction. There is much research in the evolutionary computation literature on linkage discovery, including work on messy codings and operators [2], the linkage-learning genetic algorithm [3], and probabilistic model-building genetic algorithms [6, 8, 9]. Once the linkage structure of the function is known, the global optimum can be found in $O(2^k \ell / k)$ evaluations by greedy search.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 186–197, 2004.

Previous work has shown that additively separable problems of order k defined on strings of length ℓ can be solved using $O(2^k\ell^2)$ function evaluations [4,7][1,2]. Empirical evidence as well as facetwise decompositional models suggest that sub-quadratic ($O(2^k\ell^\beta)$ for $\beta < 2$) performance is possible. For example, the Bayesian Optimization Algorithm (BOA) is estimated to require $O(2^k\ell^{1.65})$ function evaluations [8]. One of the contributions of this paper is to rigorously prove that this and much better performance is in fact possible.

This paper addresses the problem of optimizing additively separable functions from a traditional algorithms perspective. We present an algorithm, prove its correctness, and prove upper bounds on the number of function evaluations it requires. For now our sole concern is finding the global optimum of order-k additively separable functions; we are not worried about whether the resulting algorithm also performs well on real optimization problems such as MAXSAT. The algorithm presented in this paper will be shown to find a global optimum of an order-k additively separable function using $O(2^k\ell\ln(\ell))$ function evaluations, a value which is $O(2^k\ell^{1+\varepsilon})$ for any $\varepsilon > 0$. This performance claim is confirmed experimentally. We obtain more than an order of magnitude improvement over BOA on a set of problems previously studied by Pelikan [9].

The following subsection introduces definitions and notation that are used throughout the paper. Section 2 gives an overview of how our optimization and linkage discovery algorithms work. Section 3 presents the linkage discovery algorithm in full detail and analyzes its time complexity. Section 4 presents our optimization algorithm in full detail, and section 5 presents experiments illustrating it in action. Section 6 discusses the implications and limitations of this work, and section 7 concludes the paper.

Due to space limitations, some of the proofs of the lemmas and theorems given in this paper have been omitted from the text. These proofs are available online at [11], where a Java implementation of our algorithm is also available.

1.1 Definitions and Notation

In this paper we are interested in maximizing a function f defined over binary strings of some fixed length ℓ. We adopt the following notation:

$s.i \equiv$ the value of the i^{th} character of string s
$s[i \rightarrow x] \equiv$ a copy of string s with the i^{th} character set to x

Definition 1. $\Delta f_i(s) \equiv f(s[i \rightarrow (1\text{-}s.i)]) - f(s)$. ■

In other words, $\Delta f_i(s)$ represents the change in fitness that results from flipping the i^{th} bit of s. This notation was introduced by Munemoto [7].

---

[1] The bound was derived for linkage discovery, but clearly holds for optimization as well.

[2] This bound assumes that the probability of discovering all links within a particular subfunction, and not all links within all subfunctions, is to remain constant as ℓ increases.

Definition 2. Two positions i and j are *linked* w.r.t. a function f (written $\mathcal{L}_f(i, j)$) if there is some string s such that $\Delta f_i(s[j\rightarrow 0]) \neq \Delta f_i(s[j\rightarrow 1])$. ■

In other words, i and j are linked if, in the context of some string s, i's affect on fitness can depend on the value of j. Note that $\mathcal{L}$ is symmetric but not reflexive or transitive.

Definition 3. Two positions i and j are *grouped* w.r.t. a function f (written $\mathcal{G}_f(i, j)$) if $i=j$, if $\mathcal{L}_f(i, j)$, or if there is some sequence of positions $i_1, i_2, ..., i_n$ such that $\mathcal{L}_f(i, i_1)$, $\mathcal{L}_f(i_1, i_2)$, ..., $\mathcal{L}_f(i_{n-1}, i_n)$, and $\mathcal{L}_f(i_n, j)$. ■

$\mathcal{G}$ is thus the reflexive, transitive closure of $\mathcal{L}$. We shall omit the subscripts on $\mathcal{L}$ and $\mathcal{G}$ when the choice of f is obvious.

Definition 4. A *linkage group* of a function f is a set γ of one or more integer positions such that if $i \in \gamma$ then $j \in \gamma$ iff. $\mathcal{G}(i, j)$. ■

Definition 5. Let f be a function with m linkage groups $\gamma_1, \gamma_2, ... \gamma_m$. Then $\Gamma=\{\gamma_1, \gamma_2, ... \gamma_m\}$ is the *linkage group partition* of f. ■

It is easy to see that each string position i belongs to exactly one linkage group, so that a linkage group partition as just defined is in fact a partition of the set of positions $\{1,2,..., \ell\}$. We adopt the notation:

$$\Gamma[j] \equiv \text{the linkage group } \gamma \in \Gamma \text{ such that } j \in \gamma.$$

The following example will serve to illustrate these definitions. Let

$$f(s) = (s.1)(s.2) + (s.2)(s.3) + (s.4)(s.5) .$$

Then f is an additively separable function of order 3 with the two subfunctions $f_1(s) = (s.1)(s.2) + (s.2)(s.3)$ and $f_2(s) = (s.4)(s.5)$. The linkage relationships that exist w.r.t. f are:

$$\mathcal{L}_f(1,2), \mathcal{L}_f(2,3), \mathcal{L}_f(4,5),$$
$$\mathcal{L}_f(2,1), \mathcal{L}_f(3,2), \mathcal{L}_f(5,4)$$

the linkage groups are $\gamma_1=\{1,2,3\}$ and $\gamma_2=\{4,5\}$, and the linkage group partition Γ is $\{\gamma_1, \gamma_2\}$.

Intuitively, it should be clear that there is a one-to-one relationship between subfunctions and linkage groups. Namely, if f has m linkage groups $\gamma_1, \gamma_2, ... \gamma_m$ then we can write f as a sum of subfunctions $f_1, f_2, ..., f_m$ where each f_i depends only on the characters at the positions in γ_i. We prove this formally in theorem 2.

2 Overview of the Algorithm

This section gives a high-level overview of how our algorithm works.

The algorithm uses a randomized test that is guaranteed to succeed with probability at least 2^{-k} to detect linkage (interaction) between a given position i and some other position in the string. When the test succeeds it reveals two strings s and s' such that $\Delta f_i(s) \neq \Delta f_i(s')$, and the algorithm then uses binary search to isolate a specific position j that is linked to i. Each newly discovered link is used to update a linkage group partition maintained by the algorithm. The algorithm then performs a local search centered around the best-so-far known string that requires at most 2^k steps, and once the linkage group partition is fully discovered this local search is guaranteed to find a globally optimum string. The algorithm never discovers the same link twice, and so long as it is run long enough to discover the complete linkage group partition it is guaranteed to return a globally optimum string. Figure 1 presents high-level pseudocode for this algorithm.

Two previous algorithms for linkage discovery used the same randomized test for linkage as used in this paper [4, 7], but instead of using binary search restricted each instance of the test to detect linkage between a specific pair of positions i and j, resulting in a requirement[2] of $O(2^k \ell^2)$ rather than $O(2^k \ell \ln(\ell))$ function evaluations.

```
Procedure ASFOPTIMIZE(f, t):
1. Initialize Ω to a random string.
2. Initialize Γ to {{1},{2}, ..., {ℓ}}.
3. Use local search to make Ω optimal with respect to one-bit perturbations.
4. Do t times:
     For i from 1 to ℓ:
          Perform randomized test for linkage between i and some other position.
          If test succeeds then:
               Use binary search to find j such that L(i, j).
               Update linkage group partition Γ.
               Use local search to make Ω optimal with respect to newly discovered
               linkage group Γ[i] ∪ Γ[j].
5. Return Ω.
```

Fig. 1. High-level overview of the optimization algorithm presented in this paper.

3 Detecting Linkage

This section formally defines the procedures used by our optimization algorithm to discover the linkage group partition for an arbitrary function. Figure 2 and lemmas 1-2 concern our binary search procedure. Figure 3 and lemmas 3-4 concern our randomized test for linkage. Figures 4-5, lemmas 5-7, and theorem 1 concern our algorithm for finding the linkage group partition of an arbitrary function.

Procedure FINDLINKEDPOSITION(f, s, s', i):
1. assert($\Delta f_i(s) \neq \Delta f_i(s')$)
2. Let $j_1, j_2, ..., j_\delta$ be the δ distinct positions for which the values of s and s' differ.
3. If δ=1, return j_1.
4. Let $s_2 = s'[j_1 \rightarrow s.j_1][j_2 \rightarrow s.j_2]...[j_{\lfloor \delta/2 \rfloor} \rightarrow s.j_{\lfloor \delta/2 \rfloor}]$.
5. If $\Delta f_i(s) \neq \Delta f_i(s_2)$ return FINDLINKEDPOSITION(s, s_2, i) else return FINDLINKEDPOSITION(s', s_2, i).

Fig. 2. Procedure FINDLINKEDPOSITION. Given strings s and s' such that $\Delta f_i(s) \neq \Delta f_i(s')$, the procedure performs a binary search to return a position j such that $\mathcal{L}(i, j)$.

Lemma 1. If $\Delta f_i(s) \neq \Delta f_i(s')$, FINDLINKEDPOSITION($f$, s, s', i) returns a value j such that $\mathcal{L}(i, j)$ is true.

Proof: By induction on the hamming distance, δ, between s and s'.

Case δ=1: If s and s' differ only in one position there must be some j such that $s' = s[j \rightarrow s'.j]$. Assume without loss of generality that $s'.j = 0$. Then $s.j = 1$, so $s = s[j \rightarrow 1]$ while $s' = s[j \rightarrow 0]$. Thus by our assumption that $\Delta f_i(s) \neq \Delta f_i(s')$ we have $\Delta f_i(s[j \rightarrow 1]) \neq \Delta f_i(s[j \rightarrow 0])$, so by definition $\mathcal{L}(i, j)$ is true.

Case δ>1: In step 4, FINDLINKEDPOSITION creates a string s_2 that differs from s' in $\lfloor \delta/2 \rfloor$ positions and from s in $\lceil \delta/2 \rceil$ positions. Suppose $\Delta f_i(s) \neq \Delta f_i(s_2)$. In this case, the value j returned in step 5 is FINDLINKEDPOSITION(s, s_2, i), so $\mathcal{L}(i, j)$ is true by the induction hypothesis. Otherwise if $\Delta f_i(s) = \Delta f_i(s_2)$, then because $\Delta f_i(s) \neq \Delta f_i(s')$ (by assumption) it must be that $\Delta f_i(s') \neq \Delta f_i(s_2)$. In this case the value j returned in step 4 is FINDLINKEDPOSITION(s', s_2, i), so by the induction hypothesis we have $\mathcal{L}(i, j)$. ∎

Lemma 1 establishes that FINDLINKEDPOSITION performs a binary search to isolate a position j such that $\mathcal{L}(i, j)$ is true. Lemma 2 then follows from the fact that binary search on a set of δ positions requires at most $\lceil \lg(2\delta) \rceil$ iterations.

Lemma 2. If $\Delta f_i(s) \neq \Delta f_i(s')$, FINDLINKEDPOSITION($f$ s, s', i) performs no more than $4*(\lceil \lg(2\ell) \rceil - 1)$ function evaluations. ∎

Procedure TESTFORLINK(f, Γ, i):
1. s := RANDOMSTRING().
2. s_{wrk} := RANDOMSTRING().
3. Let $j_1, j_2, ..., j_n$ be the n positions in $\Gamma[i]$.
4. s' := $s_{wrk}[j_1 \rightarrow s.j_1][j_2 \rightarrow s.j_2]...[j_n \rightarrow s.j_n]$
5. If $\Delta f_i(s) \neq \Delta f_i(s')$, return FINDLINKEDPOSITION(f, s, s', i).
6. Otherwise return -1.

Fig. 3. Procedure TESTFORLINK. The procedure performs a randomized test for linkage and returns either a position j such that $\mathcal{L}(i, j)$ is true or returns -1 if the test is inconclusive.

Lemma 3. If TESTFORLINK(f, Γ, i) returns a value j then either (a) $j = -1$ or (b) $\mathcal{L}(i, j)$ and $j \notin \Gamma[i]$.

Proof: If TESTFORLINK returns at line 6 the lemma is trivially satisfied. Otherwise the procedure must return at line 5, and by lemma 1 it must return a value j such that $\mathcal{L}(i, j)$ is true. To see that $j \notin \Gamma[i]$, note that line 4 guarantees that s and s' do not differ in any of the positions in $\Gamma[i]$, so it is sufficient to show that if FINDLINKEDPOSITION(f, s, s', i) returns a value $j \neq -1$ then $s.j \neq s'.j$. This can easily be shown by induction on the number of times n that FINDLINKEDPOSITION recurses. For $n=0$, we see by inspection of line 3 of FINDLINKEDPOSITION that $s.j \neq s'.j$. For $n>0$, the induction hypothesis gives either $s.j \neq s_2.j$ or $s'.j \neq s_2.j$, and it follows from inspection of lines 2 and 4 that $s.j \neq s'.j$. ∎

In other words, TESTFORLINK only finds links that allow us to make a meaningful update to Γ.

Lemma 4. If there exists a j such that $\mathcal{L}(i, j)$ and $\Gamma[i] \neq \Gamma[j]$, then TESTFORLINK(f, Γ, i) returns a value other than -1 with probability at least $2^{-|\gamma|}$, where γ is the linkage group to which i belongs.

Proof: Suppose there exists a j such that $\mathcal{L}(i, j)$ and $\Gamma[i] \neq \Gamma[j]$. By definition of $\mathcal{L}$, there must be some string $s_{\mathcal{L}}$ such that $\Delta f_i(s_{\mathcal{L}}[j \rightarrow 0]) \neq \Delta f_i(s_{\mathcal{L}}[j \rightarrow 1])$. Let S denote the set of strings that agree with $s_{\mathcal{L}}$ on all positions in $\Gamma[i]$. Letting s and s' denote the strings created on lines 1 and 4, respectively, of TESTFORLINK, we note three things. First, if $s \in S$ then $s' \in S$ by inspection of line 4. Second, if $s \in S$ then there must be some $r \in S$ such that $\Delta f_i(s) \neq \Delta f_i(r)$. This is true because $s_{\mathcal{L}}[j \rightarrow 0]$ and $s_{\mathcal{L}}[j \rightarrow 1]$ give different Δf_i and both belong to S. Third, because Δf_i depends only on the positions in γ (see lemma 8), for any string r' that agrees with r on these $|\gamma|$ positions we have $\Delta f_i(s) \neq \Delta f_i(r')$. The probability that $s \in S$ is $2^{-|\Gamma[i]|}$. Given that $s \in S$, the probability that s' agrees with r on the remaining $|\gamma|-|\Gamma[i]|$ positions in γ is $2^{-(|\gamma|-|\Gamma[i]|)}$. So the overall probability that $\Delta f_i(s) \neq \Delta f_i(s')$ is at least $2^{-|\Gamma[i]|}2^{-(|\gamma|-|\Gamma[i]|)} = 2^{-|\gamma|}$. ∎

```
Procedure FINDLINKAGEGROUPS(f, t)
1. Γ := {{1},{2}, ..., {ℓ}}.
2. Do t times:
     For i from 1 to ℓ:
       j := TESTFORLINK(f, Γ, i)
       If j ≠ -1 then Γ := (Γ - {Γ[i]} - {Γ[j]}) ∪ {Γ[i] ∪ Γ[j]}.
3. Return Γ.
```

Fig. 4. Procedure FINDLINKAGEGROUPS. If run for a sufficiently long time, the procedure returns the linkage group partition for an arbitrary function f.

Lemma 5 follows from lemma 3 and the manner in which Γ is updated in FINDLINKAGEGROUPS.

Lemma 5. For any t, FINDLINKAGEGROUPS(f, t) returns a partition Γ such that for any i and j, if $j \in \Gamma[i]$ then $\mathcal{G}(i, j)$. ∎

We do not analyze the behavior of FINDLINKAGEGROUPS directly. Instead we analyze a simpler algorithm called MARKPOSITIONSA, and prove that its performance provides a lower bound on that of another algorithm called MARKPOSITIONSB, which we prove has exactly the same performance as FINDLINKAGEGROUPS.

```
Procedure MARKPOSITIONSA(π, t)
1. Π:={}.
2. Do t times:
   2.1. For i from 1 to ℓ:
        2.1.1. With probability π^-1,
        do:
             2.1.1.1. Π := Π ∪ {i}.
3. Return Π.
```

```
Procedure MARKPOSITIONSB(f, t, Γ_f)
1. Π := {}; Γ := {{1},{2}, ..., {ℓ}}.
2. Do t times:
   2.1. For i from 1 to ℓ:
        2.1.1. j := TESTFORLINK(f, Γ, i).
        2.1.2. If j ≠ -1 then:
             2.1.2.1. Π := Π ∪ {i}.
             2.1.2.2. Γ := (Γ - {Γ[i]} - {Γ[j]})
             ∪ {Γ[i] ∪ Γ[j]}.
             2.1.2.3. If Γ[j] = Γ_f[j], then Π :=
             Π ∪ Γ_f[j].
4. Return Π.
```

Fig. 5. Procedures MARKPOSITIONSA and MARKPOSITIONSB, which are used in the analysis of FINDLINKAGEGROUPS.

Lemma 6. The probability that MARKPOSITIONSA(π, t) returns a set containing all ℓ positions is $(1-(1-\pi^{-1})^t)^\ell$. To ensure that this probability is at least p, we must set t to at least $\ln(1-p^{1/\ell})/\ln(1-\pi^{-1})$, and this expression is $O(\pi \ln(\ell))$ for constant p.

Proof: By inspection, the probability that $j \in \Pi$ for any particular j is $1-(1-\pi^{-1})^t$. Because each position is independent, the probability that Π contains all ℓ positions is $(1-(1-\pi^{-1})^t)^\ell$. For this quantity to exceed p, t must satisfy $(1-(1-\pi^{-1})^t)^\ell \geq p$, and solving this inequality for t yields:

$$t \geq \frac{\ln\left(1-p^{\left(\frac{1}{\ell}\right)}\right)}{\ln\left(1-\pi^{-1}\right)}.$$

To see that this expression is $O(\pi \ln(\ell))$, observe that $-\ln(1-\pi^{-1})$ is $\Theta(\pi^{-1})$ because:

$$\lim_{\pi\to\infty} \frac{-\ln\left(1-\pi^{-1}\right)}{\pi^{-1}} = \lim_{x\to 0} \frac{-\ln(1-x)}{x} = \lim_{x\to 0} \frac{1}{1-x} = 1 .$$

where in the first step we have made the change of variable $x = \pi^{-1}$ and in the second step we have used l'Hôpital's rule. Thus $-1/\ln(1-\pi^{-1})$ is $O(\pi)$.

A similar argument shows that $-\ln(1-p^{1/\ell})$ is $O(\ln(\ell))$ [12], from which the lemma follows. ∎

Lemma 7. Let f be an order-k additively separable function with linkage group partition Γ_f. Then for any t, the probability that MARKPOSITIONSB(f, t, Γ_f) returns a set

containing all ℓ positions (i) is at least as high as the probability that MARKPOSITIONSA(2^k, t) returns such a set and *(ii)* is equal to the probability that FINDLINKAGEGROUPS(f; t) returns Γ_f.

Proof: (ii) The code for MARKPOSITIONSB is identical to that for FINDLINKAGEGROUPS except for the additional bookkeeping needed to maintain Π. So to prove *(ii)* it suffices to show just before MARKPOSITIONSB returns, $\Gamma=\Gamma_f$ iff. $\Pi=\{1, 2, ..., \ell\}$. To prove this it is sufficient to show that for all i' $(1 \le i' \le \ell)$, $\Gamma[i']=\Gamma_f[i']$iff. $\Gamma_f[i'] \subseteq \Pi$. Suppose $\Gamma_f[i'] \subseteq \Pi$. If any $j' \in \Gamma_f[i']$ was added to Π by line 2.1.2.3 then by inspection $\Gamma[i']=\Gamma_f[i']$. The other possibility is that all $j' \in \Gamma_f[i']$ were added to Π by line 2.1.2.1. In this case let $J'(\Gamma) = |\{\gamma: \gamma \in \Gamma \text{ and } \gamma \subseteq \Gamma_f[i']\}|$. Initially $J'(\Gamma)$ is $|\Gamma_f[i']|$. Each time line 2.1.2.1 adds a member of $\Gamma_f[i']$ to Π, the following line decreases $J'(\Gamma)$ by 1. But because $J'(\Gamma)$ must be at least 1 this can happen at most $|\Gamma_f[i']|$-1 times, so it must be that at least one $i' \in \Gamma_f[i]$ was added to Π by line 2.1.2.3. The converse (that $\Gamma[i']=\Gamma_f[i']$ implies $\Gamma_f[i'] \subseteq \Pi$) follows from inspection of lines 2.1.2.2 and 2.1.2.3.

(i) The difference between MARKPOSITIONSA and MARKPOSITIONSB is in the actions they perform within the inner loop that begins on line 2.1. When this inner loop is executed for some position i, each of the two algorithms will add i to Π with some probability. If $i \in \Pi$ already, then executing $\Pi := \Pi \cup \{i\}$ has no affect, so we may concern ourselves only with the case $i \notin \Pi$. Whether or not $i \notin \Pi$, MARKPOSITIONSA adds i to Π with probability exactly 2^{-k}. When $i \notin \Pi$, MARKPOSITIONSB adds i to Π with a probability that depends on Γ. Thus to prove part (i), it suffices to show that when $i \notin \Pi$ this probability is always at least 2^{-k}, independent of Γ. But if $i \notin \Pi$, then by the previous paragraph it must be that $\Gamma[i] \neq \Gamma_f[i]$ and lemma 4 guarantees that TESTFORLINK(f, Γ, i) returns a value other than -1 with probability at least $2^{-|\gamma|}$ where $\gamma=\Gamma_f[i]$. That $|\Gamma_f[i]| \le k$ follows from the definition of $\mathcal{G}$.■

Theorem 1 follows immediately from lemmas 6 and 7.

Theorem 1. Let f be an order-k additively separable function with linkage group partition Γ_f. The probability that FINDLINKAGEGROUPS(f, t) returns Γ_f is at least $(1-(1-2^{-k})^t)^\ell$. To find Γ_f with probability p we must invoke FINDLINKAGEGROUPS(f, $\ln(1-p^{1/\ell})/\ln(1-2^{-k})$), which will require $O(2^k \ell \ln(\ell))$ evaluations of f. ■

4 Optimizing Additively Separable Functions

Our optimization algorithm uses TESTFORLINK to discover the linkage group partition Γ in exactly the same manner as FINDLINKAGEGROUPS. Additionally, whenever a new linkage group γ is discovered, the optimization algorithm performs a local search that guarantees discovery of a string Ω that is optimal w.r.t. the linkage group γ. The result is that whenever the optimization algorithm discovers the linkage group partition Γ, it also discovers a globally optimum string.

Figures 6 and 7 present the code for our algorithm. Lemma 8 and theorem 2 are presented without proof, and formally establish the connection between the subfunctions that make up an additively separable function and the function's linkage group partition. Full proofs are available online [11]. Lemma 9 shows that our optimization algorithm finds a globally optimum string provided that it discovers the linkage group partition, and theorem 3 gives its time complexity.

```
Procedure OPTIMIZEWRTGROUP(f, Ω, γ):
1. Let i_1, i_2, ..., i_|γ| be the positions in γ.
2. For each of the 2^|γ| tuples <x_1, x_2, ..., x_|γ|> where x_a ∈ {0,1} for 1≤ a ≤ |γ|:
     Ω' := Ω[i_1→x_1][i_2→x_2]...[i_|γ|→x_|γ|].
     If f(Ω') > f(Ω) then Ω := Ω'.
3. Return Ω.
```

Fig. 6. Procedure OPTMIZEWRTLINKAGEGROUP. Performs a local search to return a string Ω that is guaranteed to be optimal with regard to a linkage group γ.

```
Procedure ASFOPTIMIZE(f, ℓ):
1. Ω := RANDOMSTRING().
2. Γ := {{1},{2}, ..., {ℓ}}.
3. For i from 1 to ℓ:
     Ω := OPTIMIZEWRTGROUP(f, Ω, Γ[i]).
4. Do t times:
     4.1. For i from 1 to ℓ:
          j := TESTFORLINK(Γ, i)
          If j ≠ -1 then
               γ := Γ[i] ∪ Γ[j].
               Ω := OPTIMIZEWRTGROUP(f, Ω, γ).
               Γ := (Γ - {Γ[i]} - {Γ[j]}) ∪ {γ}.
5. Return Ω.
```

Fig. 7. Procedure ASFOPTIMIZE. With arbitrarily high probability, returns a global optimum of an order-k additively separable function f using $O(2^k \ell \ln(\ell))$ function evaluations.

Lemma 8. Let γ be a linkage group of f, and let s_1 and s_2 be two strings such that for all $i \in \gamma$, $s_1.i = s_2.i$. Then $\Delta f_i(s_1) = \Delta f_i(s_2)$. ■

Theorem 2. Any function whose largest linkage group is of size k is order-k additively separable. Specifically, let $\Gamma_f = \{\gamma_1, \gamma_2, ..., \gamma_m\}$ be the linkage group partition for f, and let $\gamma_i = \{\alpha_{i,1}, \alpha_{i,2}, ..., \alpha_{i,k(i)}\}$, where $k(i) = |\gamma_i|$. Let S_0 be a string consisting entirely of zeroes. Then for all s, $f(s) = f_\Sigma(s)$, where:

$$f_\Sigma(s) \equiv \sum_{i=1}^{m} f_i(s.\alpha_{i,1}, s.\alpha_{i,2}, ..., s.\alpha_{i,k(i)}) \text{ , and}$$

$$f_i(\beta_1, \beta_2, ..., \beta_{k(i)}) \equiv f(S_0[\alpha_{i,1}\rightarrow\beta_1][\alpha_{i,2}\rightarrow\beta_2]...[\alpha_{i,k(i)}\rightarrow\beta_{k(i)}]) + f(S_0)(1/m\text{-}1).$$ ■

We are now prepared to make the following definition.

Definition 6. A string s is optimal w.r.t. a linkage group $\gamma_i = \{\alpha_{i,1}, \alpha_{i,2}, ..., \alpha_{i,k(i)}\}$ if $f_i(s.\alpha_{i,1}, s.\alpha_{i,2}, ..., s.\alpha_{i,k(i)})$ is maximized. ■

Corollary 1. A string s is globally optimal w.r.t. a function f iff. s is optimal w.r.t. each linkage group of f. ■

Corollary 2. If γ is a linkage group for f, then OPTIMIZEWRTGROUP(f, s, γ) returns a string s' that is optimal w.r.t. γ. ■

Lemma 9. If, when ASFOPTIMIZE(f, n) returns, Γ is the linkage group partition (Γ_f) for f, then the string Ω returned by ASFOPTIMIZE(f, n) is globally optimal w.r.t. f.

Proof: If $\Gamma=\Gamma_f$ then ASFOPTIMIZE(n) must have executed the statement Ω := OPTIMIZEWRTGROUP(Ω, γ) for every linkage group $\gamma \in \Gamma_f$. Consider the effect of executing this statement for some particular γ. Immediately after the statement is executed, Ω must be optimal w.r.t. γ by corollary 2. Because of the way in which Γ is updated, and noting γ cannot be merged with any other linkage group if we are to have $\Gamma=\Gamma_f$, all subsequent calls to OPTIMIZEWRTGROUP(Ω, γ') must be such that γ and γ' are disjoint. Such calls do not alter the positions in γ, so once Ω := OPTIMIZEWRTGROUP(Ω, γ) has been executed Ω will remain optimal w.r.t. γ until the function returns. Thus the Ω returned by ASFOPTIMIZE will be optimal w.r.t. all linkage groups $\gamma \in \Gamma_f$, so by corollary 1 Ω will be globally optimal w.r.t. f. ■

Theorem 3. Let f be an order-k additively separable function. ASFOPTIMIZE(f, t) returns a globally optimum string with probability at least $(1-(1-2^{-k})^t)^\ell$. To make this probability at least p we must invoke ASFOPTIMIZE(f, $\ln(1-p^{1/\ell})/\ln(1-2^{-k})$). This requires $O(2^k \ell \ln(\ell))$ evaluations of f and storage of O(1) strings.

Proof: The theorem would follow directly from lemma 9 and theorem 2 if we ignored the function evaluations required by calls to OPTIMIZEWRTGROUP. Thus to prove the theorem it is sufficient to show that these calls cannot account for more than $O(2^k \ell \ln(\ell))$ function evaluations. Each time OPTIMIZEWRTGROUP is called within the loop that begins on line 4, $|\Gamma|$ has just been reduced by 1 by the previous line. Because $|\Gamma|$ is initially ℓ and can never fall below 1, OPTIMIZEWRTGROUP can be called at most $|\Gamma|-1$ times within this loop. Prior to this loop OPTIMIZEWRTGROUP is called exactly ℓ times, so the total number of calls is at most $2\ell-1$. Because each call requires at most 2^k function evaluations, the total number of function evaluations is $2^k(2\ell-1)$, which is $O(2^k \ell \ln(\ell))$. The fact that ASFOPTIMIZE requires storage of O(1) strings is clear by inspection. ■

5 Experimental Results

To make the performance claims concerning our algorithm more concrete, we now present a set of experiments using the ASFOPTIMIZE procedure on additively separable functions. In particular, we examine the performance of ASFOPTIMIZE on order-5 folded trap functions [1]. Using ASFOPTIMIZE we have solved up to 10,000 bit problems; results are presented here for $5 \leq \ell \leq 1{,}000$. Performance data for BOA on this same problem for $30 \leq \ell \leq 180$ was presented by Pelikan [9].

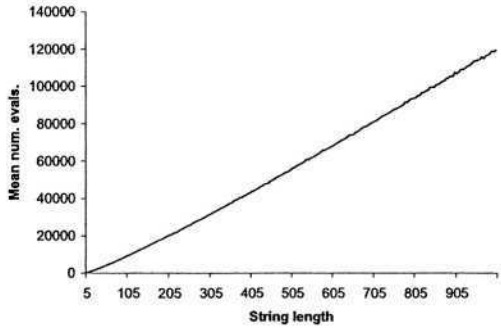


Fig. 8. Average number of function evaluations required by ASFOPTIMIZE to find the global optimum of an order-5 folded trap function, for problem sizes between 5 and 1,000. Each data point is an average of 1,000 runs. Error bars are not shown.

Figure 8 shows the average number of function evaluations required by ASFOPTIMIZE to find the global optimum as a function of problem size. As expected, the curve is near-linear. An average of 2,200 evaluations are required for ℓ=30, while 17,700 are required for ℓ=180. By way of comparison, BOA requires an average of approximately 220,000 evaluations for ℓ=180 [9]. Thus, it addition to its guaranteed $O(2^k \ell \ln(\ell))$ performance as ℓ tends toward infinity, ASFOPTIMIZE appears to be quite efficient for problems of modest size.

6 Discussion

Although the algorithm presented in this paper satisfies the operational definition of competence [1], it is not what one would consider to be a competent problem-solving algorithm. For any problem of practical interest, we expect that there is *some* degree of interaction between every pair (i, j) of positions (i.e., $\Delta f_i(s[j \rightarrow 0]) \neq \Delta f_i(s[j \rightarrow 1])$ for some s and s'), so that by our definitions the linkage group partition will be $\Gamma = \{\{1, 2, ..., \ell\}\}$. For problems with this linkage group partition, our optimization procedure will attempt to search through all 2^ℓ strings for one that is optimal w.r.t. the single linkage group in Γ, and thus will degenerate into an exhaustive search.

Though the algorithm is clearly brittle as currently defined, we do not believe that this brittleness is inherent. As discussed by Munemoto [7], the randomized linkage test employed by our algorithm can employ repeated sampling in order to handle an additively separable function corrupted by (external) additive noise. Handling noise in this manner is a first step toward handling the more systematic departures from a linear model that are characteristic of real optimization problems.

7 Conclusion

We have presented upper bounds on the time and space complexity of optimizing additively separable functions. We exhibited a simple algorithm that optimizes such functions and provided upper bounds on the number of function evaluations it requires to find a globally optimum string with a specified probability. The upper bounds provided in this paper are sharper than all those that have previously been reported. While acknowledging that the algorithm as described is not practical, we have suggested that the algorithm could be modified so as to overcome its existing limitations, and we are optimistic that the ideas presented here can be used to construct more powerful competent genetic algorithms.

References

1. D. E. Goldberg. *The Design of Innovation: Lessons from and for competent genetic algorithms.* Boston, MA: Kluwer Academic Publishers; 2002.
2. D. E. Goldberg, K. Deb, H. Kargupta, and G. Harik. Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. In *Proc. Fifth Int'l. Conf. on Genetic Algorithms.,* 1993, p 56-64.
3. G. R. Harik and D. E. Goldberg. Learning linkage through probabilistic expression. *Computer Methods in Applied Mechanics and Engineering,* 186(2-4):295-310, 2000.
4. R. B. Heckendorn and A. H. Wright. Efficient linkage discovery by limited probing. In *Proc. 2003 Genetic and Evolutionary Computation Conf.,* 2003, p 1003-1014.
5. S. A. Kauffman. *The Origins of Order.* New York: Oxford University Press; 1993.
6. P. Larrañaga and J. A. Lozano. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation.* Boston, MA: Kluwer Academic Publishers. 2001.
7. M. Munemoto and D. E. Goldberg. Linkage identification by non-monotonicity detection for overlapping functions. *Evolutionary Computation,* 7(4):377-398, 1999.
8. M. Pelikan. *Bayesian Optimization Algorithm: From Single Level to Hierarchy.* Ph.D thesis, University of Illinois Urbana-Champaign. 2002.
9. M. Pelikan, D. E. Goldberg, and E. Cantú-Paz. Linkage problem, distribution estimation, and Bayesian networks. *Evolutionary Computation,* 8(3):311-340, 2000.
10. E. D. Weinberger. NP completeness of Kauffman's NK model, a tunable rugged fitness landscape. Santa Fe Institute T.R. 96-02-003. 1996.
11. M. J. Streeter. http://www.cs.cmu.edu/~matts/gecco_2004/index.html.
12. A. H. Wright and R. B. Heckendorn. Personal communication. Jan. 2004.
13. A. H. Wright, R. K. Thompson, and J. Zhang. The computational complexity of N-K fitness functions. *IEEE Transactions on Evolutionary Computation,* 4(4):373-379, 2000.

Winnowing Wheat from Chaff: The Chunking GA

Hal Stringer and Annie S. Wu

School of Computer Science
University of Central Florida, Orlando, FL 32816
{stringer, aswu}@cs.ucf.edu

Abstract. In this work, we investigate the ability of a Chunking GA (ChGA) to reduce the size of variable length chromosomes and control bloat. The ChGA consists of a standard genetic algorithm augmented by a communal building block memory system and associated memory chromosomes and operators. A new *m*x*n* MaxSum fitness function used for this work is also described. Results show that a ChGA equipped with memory capacity equal to or greater than the minimal size of an optimal solution naturally eliminates unexpressed genes.

1 Introduction

Variable length evolutionary algorithms often evolve overly long chromosomes in which useful information is interspersed with unused information. This "bloat" or non-coding DNA can increase the computational resources required to evolve a solution. For problems where resources are limited, CPU processing time is valuable. Smaller solutions are quicker to process and are more likely to contain generalized rules, but may not be able to handle all necessary situations. A variety of methods have been employed to reduce bloat in GA and GP systems, including the addition of parsimony pressure in the fitness function and active testing for and removal of non-functional regions. Parsimony pressure if too high can favoring short but non-optimal solutions. The implementation of both parsimony pressure and active removal methods require additional computational resources during a GA or GP run.

In [1], we investigate the use of a new form of genetic algorithm coined the "Chunking GA" or ChGA to develop rule sets for autonomous micro air vehicles (MAVs). Inspired by the chunking operator in SOAR [2], the ChGA identifies and preserves good individual building blocks in the form of genes (useful rules) from solutions that a GA has already considered. Identified rules are then made available to other individuals in the population for evolving potential future solutions.

The ChGA exhibits a number of unique properties in experiments performed using a MAV simulator. One of these is the ChGA's ability to reduce the size of individual base chromosomes (rule sets) over time. This reduction occurs without explicit parsimony pressure built into either the ChGA or the fitness function.

To use an agricultural analogy, the ChGA appears able to separate good rules from extraneous rules like a farmer who separates wheat from the chaff. We refer to this property as the "winnowing effect" – the ability to reduce variable length chromosomes to a minimal sized, optimal solution. In this work, we investigate the ChGA's winnowing ability and show that the ability to reduce solution size is directly related to the size of a ChGA's memory capacity.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 198–209, 2004.

2 Related Research

Within evolutionary computation, variable length representations are most prominent in genetic programming (GP), a variant of the GA which directly evolves programs that vary in both size and content. Interestingly, with no bound on size, GP tends to evolve programs that are much larger than necessary, containing sections of code that are never used [3] [4]. Though many early efforts focused on "editing out" these excess regions, later studies indicate that the size of evolved programs may affect their fitness and evolvability [5][6][7][8].

Although most GA's focus on fixed length chromosomes, there are several examples in which variable length genomes have been used successfully, including the messyGA [9], the SAGA system [10], the SAMUEL learning system [11], and the Virtual Virus (VIV) project [12][13]. Various studies have found that parsimony pressure is an important factor in keeping individuals manageable in size as well as in maintaining reasonable fitness. There appear to be links between the evolved length and fitness of individuals and various GA parameter settings.

In a ChGA, we use a simple variable length representation. In addition, each individual in the population incorporates a short fixed length chromosome to serve as a reference to slots within a shared communal memoryThe size of a complete solution therefore varies in two ways. First, the base chromosome can vary in length as it evolves. Second, prior to fitness evaluation, the base chromosome is augmented with referenced slots in memory, potentially increasing the size of a full solution.

The idea of memory has been explored in a range of studies in the GA literature. Memory is thought to be particularly useful in dynamically changing environments, particularly those with repeated cyclical changes, e.g. seasonal changes. In such environments, genes that define a good individual or solution may become dormant when the environment changes, but can become useful again when the cycle returns to the original environment. The ability to retain previously useful information in a memory would benefit a GA working in such environments.

A common method of creating memory potential in a GA is to use redundant representations. Redundant representations are representations in which only a subset of the encoded information on an individual is expressed. Memory is possible because non-expressed information is not subject to GA selection pressure making it possible to retain information that may not be currently optimal. Examples of redundant representations include diploidy and dominance mapping [14][15][16][17], the messy GA [9], the structured GA [18], and the floating representation [19]. Additional studies on the dynamics of redundant representations include [20] [21] [22] [23].

Research in the area of explicit memory systems for the GA is more rare. In [24] an external memory structure is constructed to select and retain entire chromosomes. This "online" approach is used to construct individuals in the next generation. [25] presents a more "offline" approach. A GA's past results are used as the starting point for solving a new but similar problem – a form of case-based reasoning.

Most of these memory-related works focus on saving entire chromosomes or individuals from the population for later use. The ChGA is a departure from this approach, saving single genes or building blocks that have been extracted from the population at large. The ChGA's focus on building blocks or genes combined with a shared communal memory appears to be new to GA research.

3 Overview of the Chunking GA

The ChGA starts with a genetic algorithm using genes (bit substrings) of fixed equal length within a variable length chromosome. As an example, each gene might represent a single condition/action rule for a MAV. The set of rules (all genes contained in a chromosome) would serve as input to a simulator that returned a fitness value for the individual. Using a variable length chromosome allows the GA to evolve individuals with rule sets of different sizes.

The following three elements are added to this variable length genetic algorithm in order to implement the memory system of our Chunking GA:

1. *Shared Communal Memory.* A shared memory structure with some l number of slots is added. Each slot is capable of holding a single gene, building block or rule. All slots are of equal size (fixed number of bits). The number of available slots (l) is set by parameter prior to each ChGA run and remains constant during that run.
2. *Memory Chromosomes.* Individuals within the population have two chromosomes: a *base* chromosome and a *memory* chromosome. The base chromosome is similar to a variable length haploid chromosome used with a simple GA. The memory chromosome serves as a reference to slots in the shared memory structure. Prior to fitness evaluation, genes located in slots indicated by the memory chromosome are added to the genes in the base chromosome to create an *augmented* chromosome. The augmented chromosome is then sent to the fitness function for evaluation.
3. *Memory operators.* Operators are added to the GA to manage the shared memory structure. These operators identify good genes from the pool of base chromosomes and add them to memory, in some cases replacing good genes with better genes. Other memory operators construct augmented chromosomes by combining an individual's base chromosome with genes from referenced memory slots.

Access to rules in memory is controlled through each individual's memory chromosome. The memory chromosome consists of l bits set to either 1 or 0. Each bit is directly associated with a given slot (bit position one corresponds to slot 1, bit two corresponds to slot 2, etc.) A "1" in a bit position indicates that the associated slot is referenced by the individual. A "0" in a particular bit position indicates that the contents of that memory slot should be ignored.

The following brief example is provided for illustration purposes. Assume that a 3-slot memory structure exists. Slots contain 4-bit genes as indicated below:

	Slot 1	Slot 2	Slot 3
Contents:	1001	0110	0111

Individual X within the population has the following 3-bit memory chromosome in addition to its base chromosome.

Memory Chromosome	Base Chromosome
101	1101 1011 0001 0101

Prior to fitness evaluation, the genes contained in slots referenced by the memory chromosome are randomly inserted into the base chromosome to create an augmented chromosome representing a complete solution. This augmented chromosome is then

sent to the fitness function for evaluation. For our individual X above, the augmented chromosome might appear as follows:

0111 1101 1011 0001 ***1001*** 0101

The two rules in bold italicized typeface were obtained from memory slots referenced by "1" bits in the first and third position of X's memory chromosome. The rule "0110" from slot 2 was not included since X's memory chromosome does not reference this slot (bit position two contains a "0"). The other four rules (non-italics) of X's augmented chromosome are from its own base chromosome.

The execution of a ChGA occurs as shown in the following pseudo code:

```
procedure Chunking_GA

  // Initialize Memory
     for (i=1 to number of slots){
        set slot to random bit string;}

  // Initialize Population
     for (j=1 to population size){
        set memory chromosome to random bit string;
        set base chromosome to random bit string;}

  // Begin GA Run
     while (stopping condition not satisfied){
        for (j=1 to population size){  // Evaluate Fitness
           augmented chromosome = base + referenced slots
           evaluate fitness of augmented chromosome;}
        for (j=1 to population size / 2){ // Reproduction
           select two parents for crossover;
           crossover base chromosomes;
           crossover memory chromosomes;
           update children with resulting chromosomes;}
        for (j=1 to population size){  // Mutation
           perform mutation on child base chromosomes;
           perform mutation on child memory chromosomes;}
        copy children to next generation;
        if (update interval reached)  // Update Memory
           determine frequent genes from population
           update memory
     }
end procedure Chunking_GA
```

During execution, the ChGA must periodically identify good genes and determine if they should be added to memory. We do this by counting the number of occurrences (frequency) of each gene within the base chromosomes of the entire population. Genes are then sorted by their frequency. The basic assumption is that genes which are more useful or valuable will appear more frequently within the population. There may be other more accurate methods for identifying good genes but in this work we use frequency as a measure of "goodness" due to its simplicity.

Once frequent genes are identified and sorted, we compare them with the existing contents of memory. Memory is updated if a gene appears more often in the

population's base chromosomes than a current slot is referenced. The contents of the less referenced slot is then replaced with the newly found frequent gene.

4 Experiment Design

Experiments conducted for this work have a twofold purpose. First, we want to investigate the ChGA's ability to eliminate unnecessary genes from variable length chromosomes. Second, we want to illustrate the ChGA using a fitness function somewhat simpler than the MAV simulator used in [1]. A simpler fitness function would consume less computational resources and allow for direct analysis of results due to *a priori* knowledge of optimal solutions to the function.

4.1 A Simpler Fitness Function

It was first noted in [1] that the majority of rule sets evolved by the ChGA became smaller over time with no loss in overall fitness. This anti-bloat property occurred despite the absence of any explicit parsimony pressure in the ChGA or in the MAV simulator fitness function. Why then did this winnowing occur? Were reductions in base chromosome size a fluke? Were they a bi-product of the simulator? Or were they the result of some random occurrence?

In order to answer these questions efficiently, we require a new fitness function that meets a number of important criteria. Chief among these criteria is our ability to know the optimal solution to the function *a priori* and the exact number and composition of genes which represent the optimal solution. Other important criteria include: (1) a fitness function that is computationally simple allowing for numerous, fast evaluations (the MAV simulator require up to six seconds per evaluation); (2) a fitness function that is easily replicable by other researchers; and (3) solutions must be represented with variable length bit strings yet allow for the existence of unexpressed genes that do not contribute to the fitness of an individual.

We have named the fitness function which meets these criteria the "*m* x *n* MaxSum" function. During evaluation, each gene within a chromosome is viewed as a 2-tuple <*m, n*> where the first *m*-bits represent a positive integer index and the remaining *n*-bits represent a positive integer value which contributes to an individual's fitness. For this work we have utilized three versions of this function:

- 2x8 MaxSum: Indices range from 0-3, Values range from 0-255
- 3x8 MaxSum: Indices range from 0-8, Values range from 0-255
- 4x8 MaxSum: Indices range from 0-16, Values range from 0-255

The function is the sum of the largest value associated with each of the 2^m indices. In the case of the 2x8 MaxSum, the optimal fitness of a chromosome is 1020 based on the presence of the following 2-tuples appearing somewhere in the chromosome: <0, 255>, <1,255>, <2, 255> and <3, 255>.

Chromosomes can be of any length with this fitness function since only the largest value for each index is added to the sum. As a result, many of the genes in a chromosome can go unexpressed and serve no specific purpose.

Using the 2x8 example, individuals within a population can have a fitness less than the optimal (1020) for one of two reasons: 1) no gene exists for one or more of the permitted indices (underspecification) or 2) the maximum value represented by a gene's *n*-bits represents a number less than 255. No effort is made to make up for underspecification. A chromosome must contain 2^m tuples, one for each index in order to reach the optimal solution.

Our winnowing hypothesis states that the ChGA's ability to reduce chromosome size is tied to the size of the optimal solution. In previous work, a memory size of 10 slots was sufficient to cause anti-bloating to take place. Experiments for this work take advantage of the fact that we know the optimal size of the solution to any *m*x*n* MaxSum function – that size is 2^m. Knowing this, we are able to option a ChGA with memories of different sizes relative to the known size of the optimal solution.

There are a few caveats that should be made about our choice of fitness function. First, we recognize that the *m*x*n* MaxSum is a simple, linear function and does not contain any epistatic relationships between genes. Each gene can be optimized independently. Further, the function can be solved in a number of other ways including a simple fixed length GA. But our goal is not to put the ChGA to the test in terms of problem complexity. Rather, we are investigating only a single property of this GA and chose a fitness function that makes this both possible and efficient.

4.2 Chunking GA Details and Parameters

A series of experiments were performed to test our winnowing hypothesis. Each set of runs used a different size memory. The number of memory slots were chosen to be less than, equal to, and greater than the minimal number of genes in an optimal *m*x*n* MaxSum solution (2^m). Specifically, the following memory sizes are tested:

- For 2x8 MaxSum: No Memory (0 slots), 2, 4, 6, 8, 12, 16, & 24 slots.
- For 3x8 MaxSum: No Memory (0 slots), 4, 6, 8, 10, 12,16, & 24 slots.
- For 4x8 MaxSum: No Memory (0 slots), 10, 12, 16, 24, 32,48 & 56 slots.

If our hypothesis holds, we should see reduction in base chromosome sizes for 2x8, 3x8 and 4x8 MaxSum functions when memory contains a number of slots greater than or equal to 4, 8, and 16, respectively.

Features and parameters incorporated into the ChGA for the majority of our experiments are listed below. Unless otherwise indicated, results from all experiments were averaged over 50 runs.

- Generations per Run = 300
- Population Size =100 Individuals
- Genes per Chromosome = 20 Initial, 100 Maximal
- Number of Bits per Gene = 10, 11 or 12 (varies based on fitness function)
- Selection Method = Rank Proportional
- Crossover Type = 1-Point (both base and memory chromosomes)
- Crossover Rate = 70% (both base and memory chromosomes)
- Mutation Type = Bit Flip (both base and memory chromosomes)
- Mutation Rate = 1% (both base and memory chromosomes)
- Number of Memory Slots = varies with each experiment
- Memory Update Interval = every 10 generations

The majority of these parameters are reasonably self-evident but one in particular should be explained. At the start of each run, an initial population is generated using random bit strings. Each base chromosome starts out with 20 genes. As the ChGA executes, base chromosomes are free to increase in size up to 100 genes. Any base chromosomes larger than 100 genes that result from the crossover of two large individuals is right truncated back to 100.

5 Results from Experiments

The results of our experiments confirm our hypothesis. The winnowing effect of the ChGA is directly related to the minimal number of genes required for an optimal solution. Figures 1 through 3 show the results from experiments with different *mxn* MaxSum problems. In all cases, the chunking GA reduces base gene size when the number of memory slots is large enough to hold the minimal size solution (2^m).

When the number of available slots is less than the minimal size, the base chromosome grows and remains at an average length of 80 genes. Variable length base chromosomes are unable to go beyond this average size due to the 100-gene maximum cap specified in the ChGA's parameters.

Figures 1, 2 and 3 also illustrate an apparent correlation between speed of reduction and number of available slots. The greater the ChGA's memory capacity relative to the minimal solution size, the faster the reduction in base chromosome size.

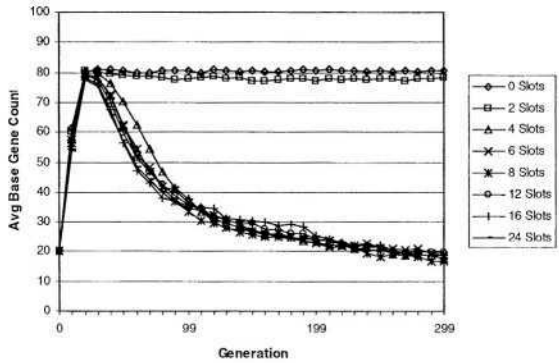


Fig. 1. Average Base Gene Counts for 2x8 MaxSum Problems. Base gene count decreases when memory holds 4 or more slots

But why do base chromosomes reduce in size to begin with? An analysis of memory contents indicates that the ChGA copies good genes into the available slots. Individuals in the population then evolve memory chromosomes to take advantage of these good genes. As an example, Table 1 gives the memory contents at the end of three different randomly selected runs.

In the Table 1, 8-slot example, memory contains the optimal genes for solving a 3x8 MaxSum problem. Individuals in the population that reference all of these genes will have the optimal solution regardless of the contents of their base chromosome.

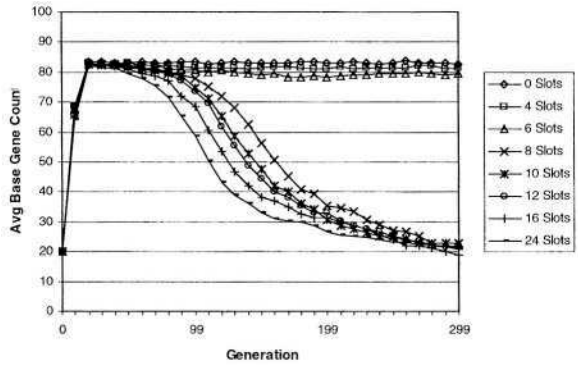


Fig. 2. Average Base Gene Counts for 3x8 MaxSum problems. Base gene count decreases when memory holds 8 or more slots. Shows speed of reduction related to number of slots.

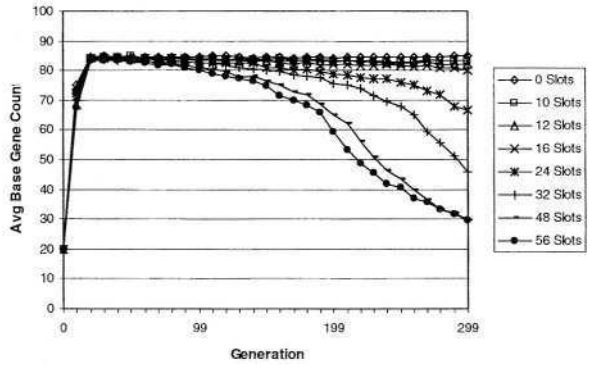


Fig. 3. Average Base Gene Counts for 4x8 MaxSum problems. Base gene count decreases when memory holds more than 16 slots. Shows speed of reduction related to number of slots.

And as seen in Figure 2, the ChGA can then begin reducing the size of the base chromosome. In one of the more surprising results of this work, the base chromosome on many individuals actually reduces to zero length. The entire solution is contained in memory as referenced by the individual's memory chromosome.

Table 1 's 4-slot example provides a contrast to the winnowing effect. The example has very good genes for indexes 0, 3, 4 and 5 but not enough room to hold an entire solution. As a result, the population must use its base chromosomes to find the four missing genes. At this point, the ChGA acts as a standard GA with variable length chromosomes and thus is subject to bloat.

The 12-slot example illustrates another interesting point. Here memory contains more slots than that required for the minimal optimal solution. For each index, a near maximal value is included. Redundant slots (those with duplicate indices) allow the individual to select from more than one possibility. Note that the frequency is lower for lower value slots.

Table 1. Samples of memory slot contents and frequency for 3x8 MaxSum runs. For each Content entry *a/b*, *a* represents the index while *b* represents the value. Frequency entries show the number of indivuals out of 100 which reference that slot in thier memory chromosome.

	4 Slots		8 Slots		12 Slots	
Slot #	Content	Freq.	Content	Freq	Content	Freq
1	5/255	95	1/255	99	6/255	100
2	3/255	92	7/255	100	5/78	37
3	4/254	98	5/255	99	1/114	25
4	0/255	99	0/255	100	5/255	97
5	n/a	n/a	3/255	99	1/254	93
6	n/a	n/a	6/255	98	3/255	99
7	n/a	n/a	4/255	100	1/255	98
8	n/a	n/a	2/255	99	2/255	100
9	n/a	n/a	n/a	n/a	7/255	99
10	n/a	n/a	n/a	n/a	4/255	99
11	n/a	n/a	n/a	n/a	1/240	62
12	n/a	n/a	n/a	n/a	0/255	97

But exactly when does the reduction in base chromosome size take place? The following figures give us some idea. Figures 4, 5 and 6 show average (online) fitness, average best (offline) fitness and the average base chromosome size (gene count).

We include Figure 4 for completeness. This graph illustrates how the ChGA behaves without any memory (0 slots). The size of the base chromosome increases quickly to its maximum capacity. Only the ChGA's 100-gene truncation parameter keeps the size of the base chromosome from increasing indefinitely.

Figure 5 shows base chromosome size increasing at the start of a run. This appears to continue until a best fit individual arises that contains the optimal solution. Base chromosome size begins to decrease slowly at first until the average fitness of the population is also very high or close to optimal. Winnowing then picks up speed dramatically. A quick inspection of memory shows that memory slots contain most, if not all of the optimal gene set when this reduction speed up occurs.

Figure 6 shows results similar to that of Figure 5 and further illustrates the fact that reduction in base chromosome size occurs earlier given larger memory capacity. We suspect that this happens since the larger memory allows critical genes to be "loaded" into slots earlier – there is more opportunity to be added to memory earlier.

Due to space limitations we are not able to present graphically, the results for all of the different experiments run. We did find similar results for all of the various combinations of memory size and *mxn* MaxSum functions listed earlier in this work.

6 Conclusions

In the introduction, we put forth the idea that the Chunking GA is able to reduce solution or chromosome size for variable length GAs. We hypothesize that this winnowing effect is influenced by the size of a ChGA's memory capacity.

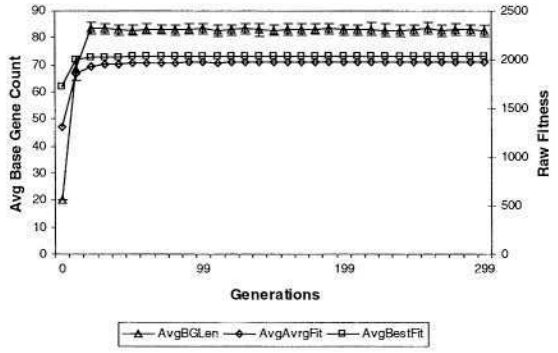


Fig. 4. Fitness vs. Base Gene Count (+/–1 std. deviation) for 3x8 MaxSum without memory

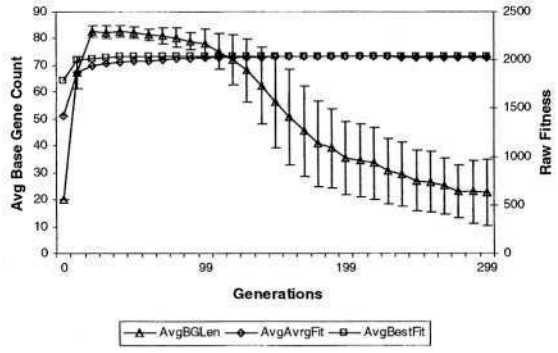


Fig. 5. Fitness vs. Base Gene Count (+/–1 std. deviation) for 3x8 MaxSum w/ 8-slot memory

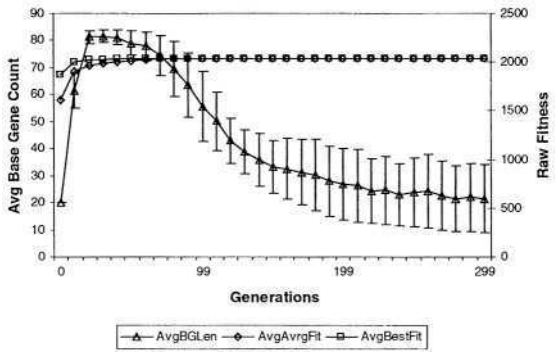


Fig. 6. Fitness vs. Base Gene Count (+/–1 std. deviation) for 3x8 MaxSum w/ 32-slot memory

Experimental results indicate that this is the case. Specifically, winnowing occurs when the memory capacity is sufficient to hold the minimal optimal solution.

Providing even greater memory capacity than that required for the minimal solution further speeds up the reduction in chromosome length. This speedup, however, is not without cost. Memory chromosomes may point to slots containing optimal genes as well as slots containing redundant counterparts. Deciding which slots contain the best genes, even from a reduced set, may be difficult. The best approach is to size memory as small as possible to capture the minimal size solution.

Our studies so far do not indicate that the ChGA necessarily yields "better" solutions unless we take into account shorter size as a criteria. If we do, then the ChGA provides optimal solutions both in fitness and in size. These shorter solutions are possible without incorporating explicit parsimony pressure in the fitness function.

The underlying cause to the ChGA's winnowing effect can be attributed to the natural desire of a variable length GA to evolve shorter chromosomes. In [26] we show that absent any selection pressure, no more than 1/3 of all child chromosomes produced by a variable length GA will be longer than their longest parent. Over time, a variable length GA will instead evolve shorter and shorter chromosomes until little or no genetic material exists in the population.

A ChGA with the proper size memory is able to select and store genes forming the best solution in its memory structure. The population then converges on a solution with a memory chromosome referencing all of the best slots in memory. Once all individuals use the same slots, they will all evaluate to the same fitness. Selection among equally fit individuals becomes random selection. At this point, the ChGA begins producing children with shorter and shorter base chromosomes until all that is left are the slots referenced by memory chromosomes.

Variable length GAs have uses in a variety of problem domains. Their use can be further enhanced by the ability to minimize the size of a solution. Without a reduction in size, the GA practitioner may not be able to determine which genes are the ones which really create the true solution. The Chunking GA illustrates an approach for finding and isolating the best genes, building blocks or rules which make up a complete solution.

References

1. A.S. Wu & H. Stringer: "Learning using chunking in evolutionary algorithms", *Proc. 11th Conf. on Computer-Generated Forces and Behavior Representations,* pp. 243-254, 2002.
2. J.E. Laird, A. Newell & P.S. Rosenbloom: "Soar: An architecture for general intelligence", *Artificial Intelligence,* Vol. 33, pp. 1-64, 1987.
3. T. Blickle & L. Thiele: "Genetic programming and redundancy", *Genetic Algorithms Within the Framework of Evolutionary Computation* (Workshop at KI-94), pp.33-38, 1994.
4. J. Koza: *Genetic Programming,* MIT Press, Cambridge, MA, 1992.
5. W.B. Langdon & R. Poli: "Fitness causes bloat", *2nd On-line World Conference on Soft Computing in Engineering Design and Manufacturing,* 1997.
6. P. Nordin & W. Banzhaf: "Complexity compression evaluation", *Proc. 6th Int'l Conference on Genetic Algorithms,* pp. 310-317, 1995.
7. J. Rosca: "Generality versus size in genetic programming", *Genetic Programming,* 1996.
8. T. Soule, J.A. Foster & J. Dickinson: "Code growth in genetic programming", *Genetic Programming,* pp. 215-233, 1996.

9. D.E. Goldberg, G. Korb & K. Deb: "Messy genetic algorithms: Motivation, analysis , and first results", *Complex Systems,* pp. 3:493-530, 1989.
10. I. Harvey: "Species adaptation genetic algorithms: A basis for a continuing SAGA", *Proc. 1st European Conference on Artificial Life,* pp. 346-354, 1992.
11. J.J. Grefenstette, C.L. Ramsey, and A.C. Schultz: "Learning sequential decision rules using simulation models and competition", Machine Learning, 5:4, pp. 355-381, 1990.
12. D.S. Burke, K.A. De Jong, J.J. Grefenstette, C.L. Ramsey & A.S. Wu: "Putting more genetics in genetic algorithms", *Evolutionary Computation,* pp. 6(4):387-410, 1998.
13. C.L. Ramsey, K.A. De Jong, J.J. Grefenstette, A.S. Wu & D.S. Burke: "Genome length as an evolutionary self-adaptation", *PPSN 5,* pp. 345-353, 1998.
14. R. E. Smith and D. E. Goldberg: "Diploidy and dominance in artificial genetic search", *Complex Systems,* Vol. 6, Number 3, pp. 251-285, 1992.
15. K. Yoshida and N. Adachi: "A diploid genetic algorithm for preserving population diversity", *Parallel Problem Solving from Nature 3,* 1994.
16. K. P. Ng and K. C. Wong: "A new diploid scheme and dominance change mechanism for non-stationary function optimization", *Proc. 6th Int'l Conf. on Genetic Algorithms,* 1995.
17. J. Lewis: "A comparative study of diploid and haploid binary genetic algorithms", Master's Thesis, University of Edinburgh, 1997.
18. D. Dasgupta and D. R. MacGregor: "Nonstationary function optimization using the structured genetic algorithm", *Parallel Problem Solving from Nature 2,* pp. 145-154, 1992.
19. A. S. Wu and R. K. Lindsay: "A comparison of the fixed and floating building block representation in the genetic algorithm", *Evolutionary Computation,* 4:2, pp. 169-193, 1996.
20. W. Banzhaf: "Genotype-phenotype mapping and neutral variation - A case study in genetic programming", *Parallel Problem Solving from Nature 3,* pp. 322-332, 1994.
21. H. Kargupta: "The gene expression messy genetic algorithm", *"Proc. IEEE Int'l Conference on Evolutionary Computation",* pp. 814-815, 1996.
22. G. R. Harik: "Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms", Ph.D. Thesis, University of Michigan, 1997.
23. M. Shackleton, R. Shipman, and M. Ebner: "An investigation of redundant genotype-phenotype mappings and their role in evolutionary search", *Proc. Congress on Evolutionary Computation,* pp. 493-500, 2000.
24. J.S. Byassee & K.E. Mathias: "Expediting Genetic Search with dynamic memory". *Proc of the Genetic and Evolutionary Computation Conference 2002,* 2002.
25. C.L. Ramsey & J.J. Grefenstette: "Case-based initialization of genetic algorithms", *In Proceedings of the 5th International Conference on Genetic Algorithms,* 1993.
26. H. Stringer & A.S. Wu: "Bloat is unnatural: An analysis of changes in variable chromosome length absent selection pressure", Technical Report CS-TR-04-01, University of Central Florida, 2004.

An Effective Chromosome Representation for Evolving Flexible Job Shop Schedules

Joc Cing Tay and Djoko Wibowo

Intelligent Systems Lab
Nanyang Technological University
asjctay@ntu.edu.sg

Abstract. As the Flexible Job Shop Scheduling Problem (or FJSP) is strongly NP-hard, using an evolutionary approach to find near-optimal solutions requires effective chromosome representations as well as carefully designed parameters for crossover and mutation to achieve efficient search. This paper proposes a new chromosome representation and a design of related parameters to solve the FJSP efficiently. The results of applying the new chromosome representation for solving the 10 jobs x 10 machines FJSP are compared with three other chromosome representations. Empirical experiments show that the proposed chromosome representation obtains better results than the others in both quality and processing time required.

Keywords. Flexible Job Shop Scheduling Problem, Genetic Algorithm, Chromosome Representation

1 Introduction

In today's engineering domain, effective planning and scheduling has become a critical issue [1]. For instance, in manufacturing production lines, efficient and effective resource allocations are required to maintain sufficient inventories while maximizing production volume. Even on smaller problems, the number of possible plans and ways to allocate resources are prohibitively large to preclude any form of enumerative search.

A commonly used abstraction of scheduling tasks is known as the *Flexible* Job Shop Problem (or FJSP). The FJSP is an extension of the classical job shop scheduling problem (or JSP) which allows an operation to be processed by any machine from a given set of resources. The task is to assign each operation to a machine and to order the operations on the machines, such that the maximal completion time (or *makespan*) of all operations is minimized.

This problem has wide applications in manufacturing and transportation systems [2] [3]. Compared to the classical JSP, the FJSP is strongly NP-hard due to 1) assignment decisions of operations to a subset of machines and 2) sequencing decisions of operations on each machine. Therefore, heuristics based on randomized search have typically been the path taken to find good approximate solutions [4].

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 210–221, 2004.

Recently, some researchers have focused on applying Genetic Algorithms (or GAs) to solve the FJSP and have obtained promising results [5] [6] [7]. In their works, chromosome representation is the first important task for a successful application of GA to solve the FJSP. Chen *et al.* [5] have used *A-string* (operations) and *B-string* (machines) representations, Mesghouni *et al.* [6] have used a parallel machine representation and a parallel job representation, while Kacem *et al.* [7] used *an assignment table* representation. In this paper, we propose a new chromosome representation that can be used by GAs to solve the FJSP efficiently.

This paper is organized as follows: Section 2 introduces the problem formulation. Section 3 describes three different chromosome representations described in literature and a new chromosome representation proposed by ourselves. Section 4 compares the results of the chromosome representation to three other chromosome representations described in Section 3 in solving the 10 jobs × 10 machines FJSP. Then, Section 5 and Section 6 represent crossover operators and mutation operators with suitable rates for the new chromosome representation. Finally, Section 7 summarizes and analyses the strengths and weaknesses of our representation.

2 Problem Formulation

The FJSP [5] can be defined as follows:

- There are n jobs, indexed by i, and these jobs are independent of each other.
- Each job i has l_i operations, and a set of precedence constraints P_i. The i-th job is denoted by J_i.
- Each job i is a set of operations O_{ij} for $j = 1, ..., l_i$.
- There are m machines, indexed by k.
- For each operation O_{ij}, there is a set of machines capable of performing it. The set is denoted by M_{ij}, $M_{ij} \subseteq \{1, ..., m\}$.
- The processing time of an operation O_{ij} on machine k is predefined and denoted by t_{ijk}.
- Each operation cannot be interrupted during its performance (non-preemptive condition).
- Each machine can perform at most one operation at any time (resource constraint).
- The precedence constraints of the operations in a job can be defined for any pair of operations.
- The objective is to find a schedule with shortest makespan, where the makespan of a schedule is the time required for all jobs to be processed in the job shop according to the schedule.

For simplicity, a matrix is used to denote both M_{ij} and t_{ijk}.

3 Chromosome Representations

Solution Representation

Each chromosome in the population represents a solution of the problem. The solution is represented by an activity graph. The activity graph is a weighted directed acyclic graph $G = (V, E, w)$. The node $v \in V$ indicates the operation and the machine where the operation will be performed. E represents a set of edges in G. The weight w of the edge $v_i \rightarrow v_j$ indicates the duration of the operation represented by the node v_i. G can be transcribed to a Gantt chart to visualize its corresponding schedule.

This section will briefly describe three chromosome representations commonly used for encoding GA-based solutions to the FJSP, afterwhich, a new chromosome representation will be presented.

Chromosome A: Machine Order with Integers

This chromosome by Chen *et al* [5] consists of two integer strings (denoted as A_1 and A_2). The length of each string is equal to the total number of operations. String A_1 assigns a machine index to each operation. The value of the j-th position of string A_1 indicates the machine performing the j-th operation. String A_2 encodes the order of operations on each machine. Both strings of Chromosome A are as follows:

- String A_1:

O_{11}	O_{12}	...	O_{ij}	...	O_{nl}
$M_{O_{11}}$	$M_{O_{12}}$	...	$M_{O_{ij}}$	...	$M_{O_{nl}}$

where $M_{O_{ij}}$ is the machine to perform operation O_{ij}, $M_{O_{ij}} \in M_{ij}$.

- String A_2:

M_1	M_2	...	M_m
O_{M_1}	O_{M_2}	...	O_{M_m}

where O_{M_k} is an ordered set of operations on machine M_k.

Chromosome B: Machine Order with Bits

The chromosome by Paredis [8] also consists of two strings (denoted as B_1 and B_2). String B_1 is identical to A_1. String B_1 is a bit string that gives the order of any pair of operations. A bit value of 0 indicates that the first operation in the paired-combination must be performed *before* the second operation. Both strings of Chromosome B are as follows:

- String B_1 (identical to A_1):

O_{11}	O_{12}	...	O_{ij}	...	O_{nI}
$M_{O_{11}}$	$M_{O_{12}}$	...	$M_{O_{ij}}$	...	$M_{O_{nI}}$

where $M_{O_{ij}}$ is the machine to perform operation O_{ij}, $M_{O_{ij}} \in M_{ij}$.

- String B_2:

$\{O_{11}\ O_{12}\}$	$\{O_{11}\ O_{13}\}$	...	$\{O_{n(I-1)}\ O_{nI}\}$
b_{1112}	b_{1113}	...	$b_{n(I-1)nI}$

where b_{ijik} is a bit specifying the precedence constraint between O_{ij} and O_{ik}.

Chromosome C: Simple Operation Order

Chromosome C by Ho and Tay [9] also consists of two strings (denoted as C_1 and C_2). This chromosome represents an instance of the FJSP, where the operations in each job have sequential precedence constraints. String C_1 encodes the order of the operations with respect to other jobs. It does not specify the order of operations within the same job as this is already implied by its index value. String C_2 represents the machine assignment to operations (as in A_1 and B_1) but with a twist. To ensure solution feasibility, the machine index is manipulated so that the string will always be valid. String C_2 identifies those machines according to availability and viability. Therefore, if the machine is not available for an operation, it won't have an index number in the set of machines and therefore this machine won't be selected. Machine selection is indicated simple boolean values. Both strings of Chromosome C are given as follows:

- String C_1:

$f_{job}(O_1)$	$f_{job}(O_2)$	...	$f_{job}(O_h)$

where O_h denotes the h^{th} operation to be performed, and f_{job} (O_h) indicates the job number of operation O_h.

- String C_2:

O_{11}	O_{12}	...	O_{ij}	...	O_{nI}
$f_{idx}(M_{O_{11}})$	$f_{idx}(M_{O_{12}})$	...	$f_{idx}(M_{O_{ij}})$	...	$f_{idx}(M_{O_{nI}})$

where $M_{O_{ij}}$ is the machine to perform operation O_{ij}, $M_{O_{ij}} \in M_{ij}$, and $f_{idx}(M_{O_{ij}})$ gives the set of index numbers of available machines for O_{ij}.

Fig. 1 gives an example of the FJSP for 2 jobs; each having 3 operations, running on 2 machines. One feasible solution for this problem is shown as an activity graph and a Gantt chart in Fig. 2. Note that the weight of an edge in the activity graph indicates the duration of the preceding operation node. The Chromosome C representation for this particular solution is shown in Fig. 3. Note that string C_2 does not indicate the machine number but the index number of available machine. Therefore, machine 2 is the first available machine for O21.

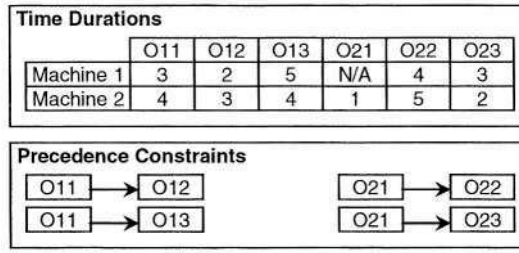


Fig. 1. Example of a 2×2 FJSP

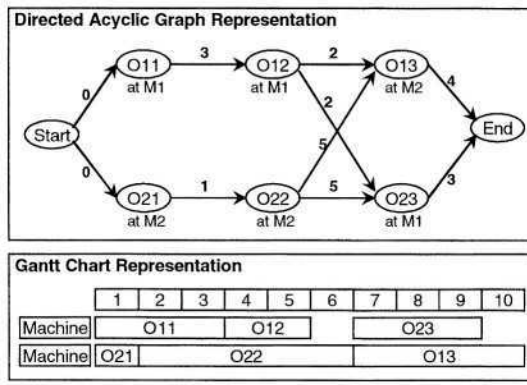


Fig. 2. Activity Graph and Gantt Chart for a Feasible Solution to the 2x2 FJSP

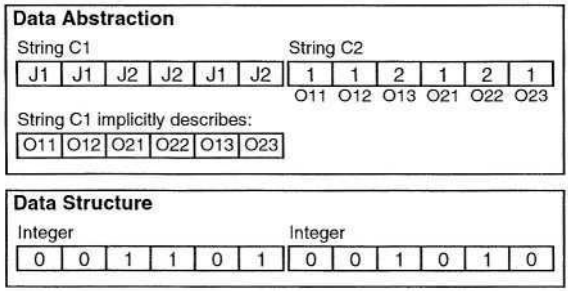


Fig. 3. Chromosome C Representation of the Feasible Solution to the 2x2 FJSP

Chromosome D: Operation Order with Bits

This chromosome is composed from chromosome representations B and C. The chromosome consists of three strings (denoted as D_1, D_2 and D_3). D_1 and D_2 are equivalent to C_1 and C_2 respectively while D_3 is similar to B_2. String D_3 is described as follows:

$\{O_{11}\ O_{12}\}$	$\{O_{11}\ O_{13}\}$	...	$\{O_{n(l-1)}\ O_{nl}\}$
b_{1112}	b_{1113}	...	$b_{n(l-1)nl}$

In string D_3, b_{ijik} is a bit specifying the precedence constraint between O_{ij} and O_{ik}. The constraints created by D_3 will only contain pairs of operations that are not included in the precedence constraints of the problem. Therefore, the possibility of invalid orders occurring due to precedence constraints is reduced. Also, a particular location in the chromosome only controls a specific property, therefore the difference between two solutions will be approximately proportional to the hamming distance between the chromosome representations.

Table 1 shows the space complexity and the time complexity in converting each chromosome into a FJSP solution during each generation. T denotes the total number of job operations in the FJSP, c denotes the number of precedence constraints, and d denotes the length of the string D_3.

Table 1. Theoretical Complexity of the Chromosome Representations

Chromosome Representation	Chromosome Length	Conversion Complexity
Chromosome A	$2T$	$O(T + c)$
Chromosome B	$T + 0.5T(T - 1)$	$O(T^2 + c)$
Chromosome C	$2T$	$O(T + c)$
Chromosome D	$2T + d$	$O(T + c + d)$

Although the asymptotic conversion complexity of Chromosome A is of the same order as Chromosome C, the former often has a larger hidden constant multiple. This is because, for each generation, Chromosome A has to be converted several times to process its partial string as follows: Though the validity of the machine can be checked in $O(T)$ time, cycle detection in $O(T + c)$ and cycle removal in $O(c)$ time, other cycles often exists after the previous cycle is removed. Therefore, for one generation, the cycle detection and removal process maybe performed more than once.

4 Empirical Results

The purpose of this experiment is to empirically compare the four different chromosome representations. Our algorithm was coded in Java on a 2 GHz Pentium IV. The experiment was conducted for 100 runs of each chromosome representation to solve a randomized FJSP of 10 jobs × 10 machines using a population of 100 individuals.

Each run consists of at least 200 generations. Chromosome A, B and D may produce invalid solutions, therefore repairing the invalid chromosome can be optionally incorporated. Chromosome C [9] can only be used to solve the FJSP where the order of operations within a job is predetermined, therefore the problem definition for the chromosome C does not include randomized precedence constraints. The results of the experiment are shown in Table 2.

Table 2. Experimental results

Chromosome Representation	Forced Repair	Average Best Makespan	Standard Deviation	Average Time (seconds)
Chromosome A	No	26.22	2.13	96.72
	Yes	25.81	2.33	97.58
Chromosome B	No	44.45	24.49	21.81
	Yes	24.88	2.67	39.21
Chromosome C	N/A	22.14	1.02	3.61
Chromosome D	No	19.73	0.97	5.17
	Yes	19.42	0.82	5.54

In terms of computation time, Chromosome A was the slowest. As strings $\mathbf{A}_1$ and $\mathbf{A}_2$ of Chromosome A must be processed separately, the algorithm is not strictly a canonical GA. Each time $\mathbf{A}_1$ was modified, $\mathbf{A}_2$ had to be rebuilt [5]. As this process was performed for every generation, the total time required increased significantly. In contrast, the strings in $\mathbf{B}_1$, $\mathbf{C}_1$, and $\mathbf{D}_1$ can be modified independently of $\mathbf{B}_2$, $\mathbf{C}_2$, and $\mathbf{D}_2$. The empirical results validate that these chromosomes are empirically faster.

The fastest result was produced by chromosome C. Unfortunately, this representation was only able to solve the FJSP with ordered precedence constraints. Although the chromosome D was not the fastest, it could solve the general FJSP with acceptable computational time.

As Chromosome A, B, and D may produce invalid representation, the forced mechanism to repair the invalid chromosome must be incorporated. For these chromosome representations, the presence of a repair mechanism always improves the makespan because it takes a larger number of generations to get a valid chromosome from the invalid chromosome by the using the standard crossover and mutation operator without any repair mechanism.

From the result in Table 2, it can also be observed that the average best makespan produced by Chromosome B without a repair mechanism is very high. This indicates that Chromosome B would produce many invalid chromosomes. Without repair, the solution can only be found in 62 cases out of 100. The repair mechanism improves the solution of Chromosome B significantly. It could also be observed that the additional computation time for the repair mechanism of Chromosome B was also more significant than the other representations.

In terms of the *average best makespan,* Chromosome D gives the best result compared to the other three representations. This is due to the simplicity of Chromosome

D's representation as compared to the others. Chromosome D also has a smaller standard deviation. Therefore, chromosome D is better in term of robustness and stability compared to the other chromosome representations used on this experiment.

5 Crossover Rates

Crossover Operators

String D_1 of Chromosome D describes the order of jobs. Therefore, offsprings may not be genetically reproduced by using standard 1-point or 2-point crossover, as the result may become infeasible. We use order-preserving 1-point crossover operator [5] for D_1. The following is an example of order-preserving 1-point crossover given two strings $D_1(1)$ and $D_1(2)$.

Before crossover:	a	crossover	b
String $D_1(1)$:	0 0	|	1 0 1 1
String $D_1(2)$:	1 0	|	1 0 0 1
After crossover:			
String $D_1(3)$:	0 0	|	1 1 0 1
String $D_1(4)$:	1 0	|	0 0 1 1

String $D_1(3)$ consists of partial strings denoted as $D_1(3)a$ and $D_1(3)b$. String $D_1(3)a$, which contains the elements before the crossover point, is constructed from $D_1(1)a$. Therefore, $D_1(3)a$ is "0 0" in this example. The string $D_1(3)b$, which contains elements after the crossover point, is constructed by removing the first occurrences of elements that are now in string $D_1(3)a$ from the string $D_1(2)$. In this case, the first two 0's of string $D_1(2)$ are removed, and therefore the string $D_1(3)b$ is "1 1 0 1". This method is also similarly applied to obtain $D_1(4)$. It can be seen that the number of 0's and 1's are preserved by this method, and therefore the string D_1 of the offspring will always be valid. For string D_2 and D_3, we use the standard 1-point or 2-point crossover operator. For string D_2, the invalid machine index assignment is handled separately. If there exist invalid indices of available machines in string D_2 after crossover is applied, then the machine is reassigned randomly to a valid machine index number.

Suitable Rates for Crossover Operators

The canonical GA relies mainly on its operators to control the diversity of the population from one generation to the next. An experiment was conducted to study the effects of the crossover rate on the optimality of the makespan when using Chromosome D. In order to show the significance of the observed parameter and reducing the effects of the other factors, all other parameters are kept constant. Order-preserving 1-point crossover operator is used for the string D_1, and standard 1-point crossover operator is used for the string D_2 and D_3, and the experiment is conducted without mutation. Therefore, the population will only be affected by the crossover operator.

We use a modified JSP standard problem instance `ft10` from Fisher and Thompson [10] with 10 machines, 10 jobs, each with 10 operations and vary the crossover rate from 0 to 1 with an interval step of 0.05. The experiment was repeated 100 times, and the averages of the best makespan were observed. The graphs of the average best makespans and its standard deviations are shown in Fig. 4. As the crossover rate increases, the average of the best makespans is observed to decrease gradually. The graph shows a significant improvement initially when the crossover rate is small. With a larger crossover rate, there is little improvement on the result and the trend flattens out. We conjecture that a crossover rate of greater than 0.85 may produce the best result. However, without mutation, it is unlikely to be the global optimal.

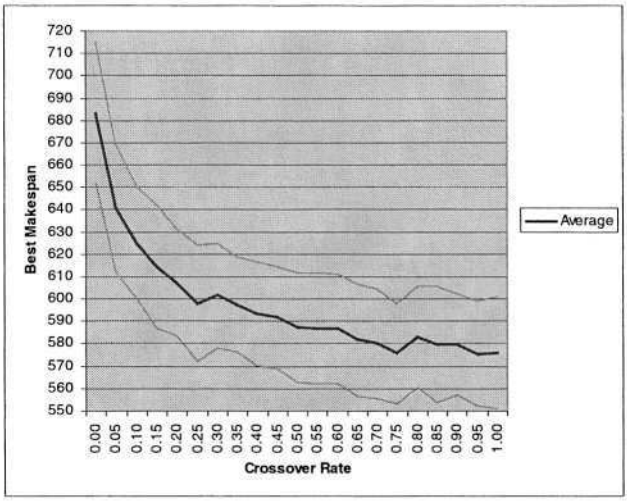


Fig. 4. Effects of crossover rate on the best makespan

6 Mutation Rates

Mutation Operators

Mutation can be applied to String D_1 of Chromosome D by swapping two elements at two randomly selected locations. Another mutation operator that can be applied is the two-opt mutation operator [11]. This operator inverts the order of operations between two randomly selected locations. In this experiment, the string D_1 is mutated by swapping two elements at two randomly selected locations, because this is commonly used for resource scheduling [12].

String D_2 can be mutated by randomly changing the machine indices. The invalid machine assignment may be handled separately, or incorporated into the operator. By incorporating it into the operator, the mutation operator can be considered as a constraint-preserving operator that will always ensure feasibility of the mutated instance.

Finally, string D_3 is a sequence of bits, which can be mutated by simply inverting a random number of them.

Suitable Rates for Mutation Operators

This experiment uses the same configuration, as well as the same randomized problem as in the previous experiment on the crossover operator. In this experiment, the crossover operator was set at 0.95. The mutation rate was varied from 0 to 1. The experiment was repeated 100 times, and the averages of the best makespan were observed. The graphs of the average best makespans and its standard deviations are shown in Fig. 5 and Fig. 6.

Fig. 5 shows that the best result is achieved when the mutation rate is around 0.025. There is a significant improvement on the average of makespan from mutation rate of 0 to 0.025. As the mutation rate increases beyond 0.025, the average of the best makespan also increases gradually.

To investigate the significant improvement on the best makespan from mutation rate of 0 to 0.025, another experiment was conducted by varying the mutation rate from 0 to 0.03 with an interval of 0.001. The graph of the result is shown on Figure 6.

In the interval of 0 to 0.004, the best makespan drops significantly. This shows the importance of the mutation operators in reducing the best makespan. As the mutation rate increases, the best makespan is quite stable and slowly increases beyond 0.018. The trend of increasing makespan after 0.03 is further supported by the trend of the best makespan after 0.025 in Fig. 6. A large mutation rate may destroy the good schemata and reduce the GA's ability to find better makespans. Therefore, we conjecture that the mutation rate around 0.006 to 0.017 would produce the best result.

The experiment was conducted using Chromosome D, with high crossover at null mutation until equilibrium is reached, then followed by regeneration and application of mutation at the previous equilibrium crossover point (or greater). We believe this is a good approach to solve the FJSP.

7 Conclusions

In this paper, a new chromosome representation for solving the FJSP was proposed. The three partial strings of Chromosome D are independent; therefore they can be modified separately by appropriate genetic operators. A particular location in the chromosome only controls a specific property, therefore the difference between two solutions will be proportional to the hamming distance between the chromosome representations. The order specified by String D_1 is always valid. The transitive closure of the precedence constraints has to be constructed to create String D_3. This may incur an overhead at the initial stage, but it will reduce the possibility of invalid orders due to precedence constraints as the explicit and implied precedence constraints of the problem are not included in the chromosome.

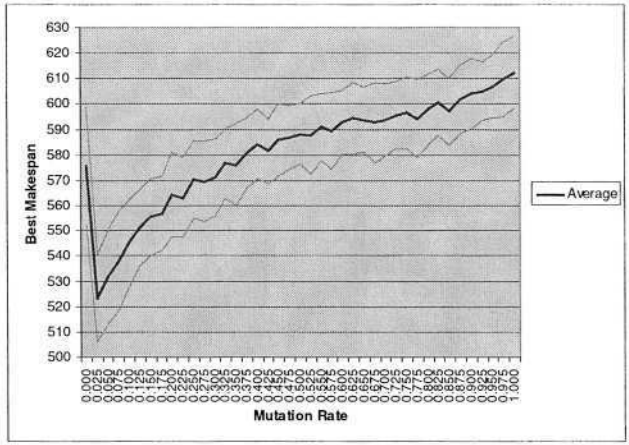


Fig. 5. Effects of mutation rate on the best makespan

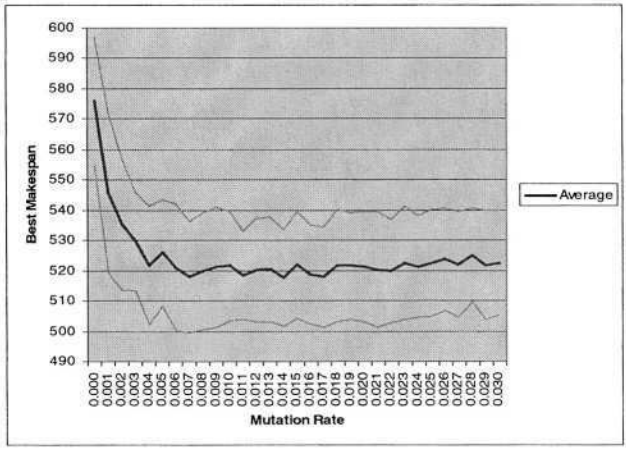


Fig. 6. Effects of mutation rate on the best makespan

The crossover operator and mutation operator for the new chromosome representation and their suitable rates were also presented. From the experiments conducted, it has been empirically shown that the new chromosome representation produces a schedule with shorter makespan. By using Chromosome D, with high crossover (> 0.85) at null mutation until equilibrium is reached, then followed by regeneration and application of mutation (around 0.006 to 0.017) at the previous equilibrium crossover point (or greater) would produce a lower average makespan.

References

1. Jain A.S. and Meeran S., "Deterministic Job-Shop Scheduling: Past, Present and Future", In *European Journal of Operation Research,* **113** (2), 390-434, 1998.
2. Pinedo, M., Chao, X., *Operations scheduling with applications in manufacturing and services,* McGraw-Hill, Chapter 1, 2–11, 1999.
3. Gambardella, L. M., Mastrolilli, M., Rizzoli, A. E., Zaffalon, M., "An optimization methodology for intermodal terminal management", In *Journal of Intelligent Manufacturing,* **12**, 521-534, 2001.
4. Jansen K., Mastrolilli M., Solis-Oba R., "Approximation Algorithms for Flexible Job Shop Problems", In *Proceedings of Latin American Theoretical Informatics* (LATIN'2000), LNCS 1776, 68-77, 1999.
5. Chen, H., Ihlow, J., and Lehmann, C., "A genetic algorithm for flexible job-shop scheduling", In *Proceedings of IEEE International Conference on Robotics and Automation,* **2,** 1120–1125, 1999.
6. Mesghouni K., Hammadi S., Borne P., "Evolution programs for job-shop scheduling", In *Proc. IEEE International Conference on Computational Cybernetics and Simulation,* **1,** 720-725., 1997.
7. Kacem I., Hammadi S. and Borne P., "Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems", In *IEEE Transactions on Systems, Man and Cybernetics,* **32**(1), 1-13, 2002.
8. Paredis, J., *Exploiting constraints as background knowledge for genetic algorithms: A case-studyfor scheduling,* Ever Science Publishers, The Netherlands, 1992.
9. Ho N. B. and Tay J. C., "GENACE: An Efficient Cultural Algorithm for Solving the Flexible Job-Shop Problem", accepted for publication in *IEEE Congress of Evolutionary Computation* 2004.
10. Fisher, H., Thompson, G.L., "Probabilistic learning combinations of local job-shop scheduling rules", Industrial Scheduling, Prentice Hall, Englewood Cliffs, New Jersey, 225-251, 1963.
11. Lin, S., and Kernighan, B. W., "An effective heuristic for traveling salesman problem", In *Operations Research,* **21**, 498-516, 1973.
12. Syswerda, G., *Schedule optimization using genetic algorithms,* Ed. L Davis (New York: Van Nostrand Reinhold) 332-349, 1991.

Linkage Identification by Nonlinearity Check for Real-Coded Genetic Algorithms

Masaru Tezuka[1,2], Masaharu Munetomo[1], and Kiyoshi Akama[1]

[1] Hokkaido University,
Kita 11, Nishi 5, Kita-ku, Sapporo, 060-0811, JAPAN
tezuka@uva.cims.hokudai.ac.jp
{munetomo, akama}@cims.hokudai.ac.jp
[2] Hitachi East Japan Solutions, Ltd.
2-16-10, Honcho, Aoba-ku, Sendai, 980-0014, JAPAN
tezuka@hitachi-to.co.jp

Abstract. Linkage identification is a technique to recognize decomposable or quasi-decomposable sub-problems. Accurate linkage identification improves GA's search capability. We introduce a new linkage identification method for Real-Coded GAs called LINC-R (Linkage Identification by Nonlinearity Check for Real-Coded GAs). It tests nonlinearity by random perturbations on each locus in a real value domain. For the problem on which the proportion of nonlinear region in the domain is smaller, more perturbations are required to ensure LINC-R to detect nonlinearity successfully. If the proportion is known, the population size which ensures a certain success rate of LINC-R can be calculated. Computational experiments on benchmark problems showed that the GA with LINC-R outperforms conventional Real-Coded GAs and those with linkage identification by a correlation model.

1 Introduction

The optimization by means of Genetic Algorithms (GAs) is promoted by the exchange of building blocks (BBs). BBs are considered sub-solutions and important subcomponents. GA's search capability is improved by identifying BBs accurately and preventing crossover operators from destructing BBs. Thus, linkage identification, the procedure to recognizing BBs, plays an important role in GA's optimization. Many research efforts have been concentrated on the linkage identification of binary-coded GAs. LLGA (Linkage Learning GA) [1] applies a two-point-like crossover operator to ring-shaped chromosomes and construct linkages dynamically. BOA (Bayesian Optimization Algorithm) [2] constructs a Bayesian network based on the distribution of individuals in a population and identifies linkages indirectly. LINC (Linkage Identification by Nonlinearity Check) [3], LIMD (Linkage Identification by non-Monotonicity Detection) [4], and LIEM (Linkage Identification with Epistasis Measure) [5] identifies linkage groups based on the nonlinearity, non-monotonicity, and epistasis measures between loci respectively.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 222–233, 2004.

Real-Coded GAs operate on real-valued vectors as their chromosomes [6] and have been applied to various real value optimization problems[7,8,9]. When an objective function in the domain of the reals is decomposable or quasi-decomposable to a number of lower order sub-functions, the solution of the original problem is yielded by combining independently obtained sub-solutions. Solving the sub-problems is considered as obtaining BB candidates in the domain of the reals. Thus, accurate linkage identification methods for Real-Coded GAs are desired.

While many researches on the linkage identification for binary-coded GAs have been conducted, only a few are known for Real-Coded GAs. Piecewise Interval Correlation by Iteration (PICI) [10,11] is one of them. PICI calculates correlation coefficients among loci, and recognizes the set of loci with high correlations as a linkage group. However, high correlation means that there is *linear* relation among the loci. The loci with linear relation are still additively decomposable into much smaller sub-problems. Each linkage group as a set of unseparable portions should have *nonlinear* relation. Thus, we introduced a new linkage identification method for Real-Coded GAs called LINC-R (Linkage Identification by Nonlinearity Check for Real-Coded GAs), based on the nonlinearity among loci.

In section 2, we briefly review PICI. Then we propose a new method, LINC-R in section 3. LINC-R is compared to PICI and a conventional Real-Coded GA. The results of the comparison are reported in section 4, followed by conclusion in section 5.

2 Piecewise Interval Correlation by Iteration

The probability density function of individual $\mathbf{x}$ at generation t under proportionate selection is written as

$$p(\mathbf{x}, t) = \frac{f^t(\mathbf{x}) p_0(\mathbf{x})}{\int f^t(\mathbf{x}) p_0(\mathbf{x}) d\mathbf{x}} \tag{1}$$

where f is a non-negative fitness function and p_0 is an initial distribution of individuals. The distribution of individuals in a population reflects the landscape of f and that gets amplified as t increases. If there are linkages among the arguments of f, there also may be some degree of correlation among them. PICI divides a domain into several sub-domains and calculate correlation coefficients of the sub-domains. Then the weighted average of the coefficients is taken as the piecewise interval correlation. PICI recognizes the set of loci with high correlations as a linkage group.

PICI has two variations, Linkage Identification with Single-Stage evolution (LISS) and with Multi-Stage evolution (LIMS). LISS learns linkages as optimization progresses. LIMS consists of three stages, initial, learning and searching stage. Firstly, in the initial stage, population forms a linkage structure. Then in the learning stage, LIMS learns linkages. Finally in the searching stage, LIMS uses the linkage information in optimization. It is reported that LIMS has higher performance than LISS.

3 GA with Linkage Identification by Nonlinearity Check for Real-Coded GAs

3.1 Nonlinearity Check for Real-Coded GAs

In this section, we propose Linkage Identification by Nonlinearity Check for Real-Coded GAs (LINC-R). LINC-R is based on an idea that if a function is linearly decomposable to several sub-functions and two loci belong to different sub-functions, the partial difference of the function with respect to one of the loci is independent from the value on the other locus. LINC-R is also an extension of LINC which is proposed for binary GAs since both LINC and LINC-R identify linkages based on nonlinearity detection.

LINC checks nonlinearity in each pair of loci whether $\Delta f_i + \Delta f_j = \Delta f_{ij}$ or not, where Δf_i is the amount of change caused in fitness by a perturbation on locus i, Δf_j is that caused by a perturbation on locus j, and Δf_{ij} is that by the perturbations on both i and j at a time. In other words, LINC judges the loci have linkage when the following condition is satisfied.

$$|\Delta f_{ij} - (\Delta f_i + \Delta f_j)| > \epsilon \tag{2}$$

where ϵ is a parameter specifying allowable error for nonlinearity check.

Since the chromosomes of Real-Coded GAs are real-valued vectors, LINC-R can not perturb the value on each locus in the same way that LINC does on binary strings. Thus we introduced random perturbation. LINC-R checks nonlinearity by equation (2) with random perturbation,

$$\Delta f_{ij} = f(x_i + \Delta x_i, x_j + \Delta x_j) - f(x_i, x_j) \tag{3}$$

$$\Delta f_i = f(x_i + \Delta x_i, x_j) - f(x_i, x_j) \tag{4}$$

$$\Delta f_j = f(x_i, x_j + \Delta x_j) - f(x_i, x_j) \tag{5}$$

where Δx_i and Δx_j are chosen randomly so that the perturbed points will fall into the domain.

LINC-R evaluates the objective function values at four points, (x_i, x_j), $(x_i + \Delta x_i, x_j)$, $(x_i, x_j + \Delta x_j)$, and $(x_i + \Delta x_i, x_j + \Delta x_j)$. There is a possibility of misjudgment that in spite of a pair of loci having a linkage the randomly selected four points seem to be linear unexpectedly. In order to reduce the possibility of such misjudgment, a number of individuals are used to identify a linkage. Figure 1 shows the procedure of LINC-R. The number of objective function evaluations required for LINC-R is $\{3n(n-1)/2+1\}|P|$, where $|P|$ is the size of a population P.

The essential difference between LINC-R and LINC is their domain, the binaries or the reals. LINC-R tests nonlinearity by random perturbations in its domain while so does LINC by bit-wise perturbations. In the domain of the binaries, only '0' and '1' can be taken on the locus. However, one random perturbation covers only small region in the domain of the reals. Thus, in order to realize accurate linkage identification, we need to employ properly sized population according to the difficulty of problems.

```
1.  Randomly initialize a population P.
2.  for each x in P
3.    for i = 1 to n-1
4.      for j = i +1 to n
5.        if linkage between locus i and j has not detected yet
6.          dx_i = (Uniform random value [x_i^L, x_i^U]) - x_i.
7.          dx_j = (Uniform random value [x_j^L, x_j^U]) - x_j.
8.          Δf_ij = f(..., x_i + dx_i, ..., x_j + dx_j, ...) - f(..., x_i, ..., x_j, ...)
9.          Δf_i = f(..., x_i + dx_i, ..., x_j, ...) - f(..., x_i, ..., x_j, ...)
10.         Δf_j = f(..., x_i, ..., x_j + dx_j, ...) - f(..., x_i, ..., x_j, ...)
11.         if | Δf_ij - (Δf_i + Δf_j) | > ε then
12.             Detect linkage between locus i and j.
13.         end if
14.       end if
15.     end for
16.   end for
17. end for
```

Fig. 1. LINC-R: Linkage Identification for Real-Coded GA based on nonlinearity

3.2 Population Sizing of the LINC-R

In this section, we will discuss on the population size required for LINC-R. In order to investigate the population size, we introduced a function f_D which has partial nonlinearity in the domain $x_i \in [0.0, 1.0]$ for $i = 1, ..., n$.

$$f_D(\mathbf{x}) = \begin{cases} f_L(\mathbf{x}) + f_{NL}(\mathbf{x}), & if \sum_{i=1}^{n} x_i^2 \leq r^2 \\ f_L(\mathbf{x}), & otherwise. \end{cases} \tag{6}$$

$$f_{NL}(x_1, ..., x_n) = \left(1 - \frac{1}{r}\sqrt{\sum_{i=1}^{n} x_i^2}\right) \tag{7}$$

$$f_L(x_1, ..., x_n) = \frac{\lambda}{n}\sum_{i=1}^{n} x_i \tag{8}$$

where $0.0 \leq \lambda < 1.0$ and $0.0 \leq r \leq 1.0$ are the parameters. The optimal value of f_D is 1.0 at $\mathbf{x} = (0, ...,0)$ and a sub-optimum is λ at $\mathbf{x} = (1,...,1)$. f_D is nonlinear within an r radius from the origin and linear otherwise. Figure 2 shows an example of f_D with $n = 2$, $\lambda = 0.8$, $r = 0.2$.

Since LINC-R uses four points (x_i, x_j), $(x_i + \Delta x_i, x_j)$, $(x_i, x_j + \Delta x_j)$, and $(x_i + \Delta x_i, x_j + \Delta x_j)$ to test nonlinearity, when all the points are outside of the nonlinear region of the objective functions, LINC-R fails to detect linkage. However, it can succeed in detecting linkage as long as at least one of the points is in the nonlinear region. Here, a denotes the proportion of the nonlinear region in the domain. $a = \pi r^2/4$ for f_D. The probability that at least one of four points falls into the nonlinear region is $1-(1-a)^4$. Thus, the probability of successfully

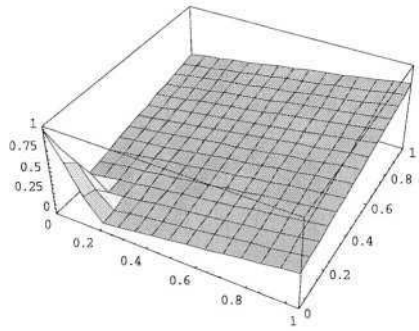


Fig. 2. Objective function f_D which has nonlinerity in a part of domain

identifying linkage in a pair of loci is,

$$Pr = \left\{1 - (1-a)^4\right\} \sum_{i=1}^{|P|} \left\{(1-a)^4\right\}^{i-1} = \left\{1 - (1-a)^4\right\}^{|P|}. \tag{9}$$

The third term is derived from the fact that the second term is the sum of the geometric sequence. From the equation, we have the population size as follows:

$$|P| = \frac{ln(1-Pr)}{4ln(1-a)} \tag{10}$$

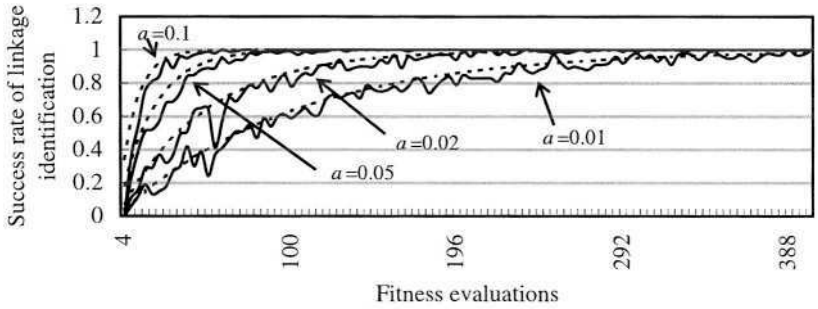


Fig. 3. Success rate of linkage identification along with the proportion of nonlinear region.

Figure 3 shows the success rate of linkage identification with $a = 0.1$, 0.05, 0.02, 0.01. X axis shows the number of objective function evaluations to identify linkage, that is $|P| \times 4$. Dotted lines show theoretical probability obtained from equation (9). Solid lines show the result of computational experiments on f_D where $n = 2$. The values are the average of 100 trial runs. A linkage may be

successfully identified within 100 evaluations if the proportion of the nonlinear region is more than five percent ($a \geq 0.05$). Note that the above discussion concerns the probability of linkage identification for one pair of loci while there are ${}_nC_2$ combinations of possible pairs for n dimensional vectors.

3.3 Optimization Procedure of the GA with Linkage Identification by Nonlinearity Check for Real-Coded GAs

The optimization procedure of real-coded GAs with LINC-R consists of two stages as shown in figure 4. Firstly, in the linkage identification stage, LINC-R identifies linkage groups. Then, in the optimization stage, GAs are performed in each linkage group separately.

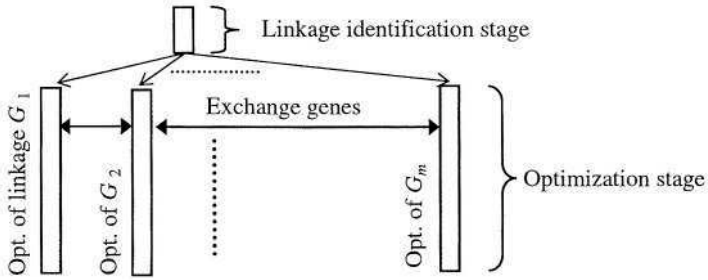


Fig. 4. Optimization by Real-Coded GA with LINC-R.

Here, m denotes the number of linkage groups identified by LINC-R. $G_i = \{x_{k_i-1}, ..., x_{k_i}\}$ is the set of loci belonging to linkage group i. In the optimization stage, m islands are created. The population created in the linkage identification stage is not used in this stage. The population of island i is initialized so that the genes on the loci belonging to G_i are initialized randomly, and for the genes on the other loci, the gene on the same locus has the same value throughout the population. In island i, genetic operators are applied to the only loci in G_i. Thus, island i optimizes linkage group G_i, that is $|G_i|$ dimensional sub-problem. In this stage, genes are exchanged among islands periodically. The genes on the loci in G_i of the best individual of the island i are copied to those of the other islands. Every time genes are exchanged, all the population in each island has to be re-evaluated and the number of function evaluations is consumed largely for the re-evaluation.

Generally, the computational complexity of the optimization problems rises exponentially as the dimension increases. Thus, in the optimization stage, more search effort has to be invested to the island which optimizes larger linkage group. In this paper, we allocate search cost as follows:

The number of generations. Allocate the number of generations proportional to the dimension of linkage group. $|G_i|$ generations are alternated in island i while $|G_j|$ generations are in island j.

Population size. Population sizes are set to be proportional to the square of the dimension. The population size of island i is $|P_i| = C_p \times |G_i|^2$ where C_p is a constant.

4 Numerical Experiments

4.1 Experimental Conditions

We employ SPX (simplex crossover) [12] and MGG (Minimum Generation Gap) model [13] both employed in the reports on PICI [10,11] in order to compare the performance of LINC-R and PICI.

We judge the optimal solution $(o_1, ..., o_n)$ is found when $\forall i,\ x_i \in [o_i - \Delta x/2, o_i + \Delta x/2]$ where Δx is a resolution of the solution. We set Δx to 0.001, same as the PICI's reports used.

We set the interval of gene exchange depending on the total population which is the sum of the population of all the islands. When the total population is smaller than 5,000, genes are exchanged every 50,000 function evaluations. When the total population is smaller than 10,000, genes are exchanged every 100,000 function evaluations, otherwise every 1,000,000 evaluations.

4.2 Functions with Nonlinearity in the Whole Search Space

Firstly, we use Type I and Type II functions employed by Tsutsui et al. in PICI's reports defined by following equations respectively.

$$F_1(\mathbf{x}) = Fr_T(x_1, ..., x_T) + Fs_L(x_{T+1}, ..., x_{T+L}) \tag{11}$$

$$F_2(\mathbf{x}) = \sum_{i=1}^{T} Fr_2(x_{2i-1}, x_{2i}) + Fs_L(x_{2T+1}, ..., x_{2T+L}) \tag{12}$$

where $-2.048 \leq x_i < 2.047$ for all i. These functions are to be minimized and have global optimum of 0.0 at x = (1,...,1)].

Fr_T is a T dimensional Rosenbrock function defined by equation (13) and has linkage between the first and the other arguments. Since nonlinearity of this linkage group exists throughout the domain, proportion of the nonlinear region is 1.0, that is, $a = 1.0$.

$$Fr_T(x_1, ..., x_T) = \sum_{i=2}^{T} \left(100\left(x_1 - x_i^2\right)^2 + (x_i - 1)^2\right) \tag{13}$$

Fs_L is an L dimensional sphere function defined by equation (14) that has no linkage.

$$Fs_L(x_1, ..., x_L) = \sum_{i=1}^{L} (x_i - 1)^2 \tag{14}$$

Both type I and II function are nonlinear among the loci in the same linkage group in the whole domain. That means $a = 1.0$ in equation (9) and the

probability of successfully identifying linkage, Pr, is always 1.0. Thus, we set the population size in the linkage identification stage to one.

We test the capability of the GA employing LINC-R with various T, the dimension of Rosenbrock function. $T = 2, ..., 8$ for Type I function and $T = 2, ..., 4$ for Type II function. L, the dimension of the sphere function, is fixed to 20. C_p is set to 10.

Ten runs are performed for each parameter setting. Each run continues until the optimal solution is found or the number of function evaluations reach to 1.0×10^6. The number includes the evaluations for linkage identification.

Table 1 and 2 show the number of successful runs which found the optimum (#Opt) and the mean (MNE) and the standard deviation (STDEV) of the number of function evaluations to find the optimum in the runs which found the optimum. The standard deviations are zero or very small in easy cases because the optimum solutions are found at the first gene exchange in all the trials in the cases.

Table 1. Result of optimization of type I function.

	without linkage identification		PICI(LIMS)		LINC-R		
T	#Opt	MNE	#Opt	MNE	#Opt	MNE	STDEV
2	10	200,033	10	172,420	10	51,074	0
3	3	452,392	10	204,837	10	52,272	0
4	1	441,632	10	222,771	10	67,053	9,703
5	0	-	10	246,794	10	97,747	12,084
6	0	-	10	266,844	10	144,197	16,413
7	0	-	10	287,307	10	183,976	11,907
8	0	-	10	326,890	10	278,533	18,738

Table 2. Result of optimization of type II function.

	without linkage identification		PICI(LIMS)		LINC-R		
T	#Opt	MNE	#Opt	MNE	#Opt	MNE	STDEV
2	0	-	10	205,808	10	51,417	0
3	0	-	10	252,105	10	51,723	15
4	0	-	10	285,725	10	61,619	11,794

For comparison, the results of GA without linkage identification and PICI are also shown. The results of PICI and the GA without linkage identification are from the work of Tsutsui et al[11]. Standard deviatoins are not shown because they are not reported in the work. Obviously, the proposed method – LINC-R – shows outstanding performance. LINC-R is superior to the conventional methods

because (1) LINC-R has an ability to identify linkage groups precisely on the functions with nonlinearity in a whole domain, and (2) Parallel GA optimizes smaller and easier sub-problems into which the original problem is divided.

4.3 Functions with Nonlinearity in a Part of Search Space

Secondly, we employ the sum of two dimensional trap functions f_D defined by equation (6) which has nonlinearity in a part of its domain.

$$F(\mathbf{x}) = \sum_{i=1}^{n/2} f_D(x_{2i-1}, x_{2i}) \tag{15}$$

This function is to be maximized and has global optimum of $n/2$ at $\mathbf{x} = (0,\ldots,0)$. λ is set to 0.8, that means $F(\mathbf{x})$ has a sub-optimum of $0.8n/2$ at x = (1, …,1). Each pair of locus $2i-1$ and locus $2i$ is tightly linked. Thus, F has $n/2$ linkage groups. We set n to 12, 16, and 24. We test LINC-R on the function with various $a(=\pi r^2/4)$ which is the proportion of the nonlinear region on two dimensional f_D. This function gets difficult as a decreases.

For $a = 0.01$, that is the hardest case in this experiment, the population size required for identifying a pair of loci with probability 0.99 is 115 which is obtained by substituting $a = 0.01$ and $Pr = 0.99$ in equation (10). For $n = 24$, the number of function evaluations consumed in the linkage identification stage is 94,964 ($\approx 1.0 \times 10^5$) on that occasion. Thus, in this experiment, linkage identification stage is executed until the number of the function evaluations reached to 1.0×10^5. Then optimization stage is performed until the optimal solution was found or the number of function evaluations reached to 1.0×10^8. 20 runs are performed for each parameter setting.

C_p is set to $10/a$. Since the problem gets difficult as a decreases, more population is required for the problem with smaller a. Thus we set C_p depending on a.

Table 3, 4 and 5 show the number of successful runs which found the optimum (#Opt) and the mean (MNE) and the standard deviation (STDEV) of the number of function evaluations to find the optimum in the runs which found the optimum. The test function F has $n/2$ linkage groups. For LINC-R, the tables also show the average rate of the linkage groups successfully identified (Success rate). For comparison, the result of GA without linkage identification is also shown.

In the case of $n = 12$, LINC-R can obtain optimal solution in all 20 trial runs for every a while so can the GA without linkage identification in a half of trials for $a = 0.02$ and only one trial for $a = 0.01$. In the case of $n = 24$, that is the hardest case, LINC-R can obtain optimum in all trials for a smaller than 0.01 and in 11 trials for $a = 0.01$ while the GA without linkage identification can obtain no optimum even for $a = 0.5$.

LINC-R is more effective in reaching the optimal solutions, however, in the cases of $a = 0.1$ and $a = 0.05$ in $n = 12$ and $n = 16$, it requires higher computational effort than the conventional GA. We thought that is because of an

Table 3. Result of optimization.($n = 12$)

	without linkage identification			LINC-R			
a	#Opt	MNE	STDEV	#Opt	MNE	STDEV	Success rate of linkage identification(%)
0.5	20	255,407	538	20	152,760	5.6	100
0.2	20	291,370	18,777	20	234,831	21,311	100
0.1	20	379,521	24,726	20	513,679	43,283	100
0.05	20	599,305	55,613	20	1,272,980	89,762	100
0.02	10	4,614,750	8,449,032	20	3,679,120	436,108	100
0.01	1	7,769,660	-	20	11,846,500	1,533,870	100

Table 4. Result of optimization.($n = 16$)

	without linkage identification			LINC-R			
a	#Opt	MNE	STDEV	#Opt	MNE	STDEV	Success rate of linkage identification(%)
0.5	20	359,487	11,490	20	187,958	21,375	100
0.2	20	428,539	32,691	20	305,411	6,907	100
0.1	20	617,478	43,013	20	1,208,440	73,266	100
0.05	20	1,095,780	364,831	20	3,441,950	348,240	100
0.02	13	3,737,390	1,210,611	20	5,362,120	440,592	100
0.01	0	-	-	20	18,904,000	2,421,623	100

Table 5. Result of optimization.($n = 24$)

	without linkage identification			LINC-R			
a	#Opt	MNE	STDEV	#Opt	MNE	STDEV	Success rate of linkage identification(%)
0.5	0	-	-	20	208,019	11,559	100
0.2	0	-	-	20	954,755	64,969	100
0.1	0	-	-	20	2,391,770	146,602	100
0.05	0	-	-	20	2,167,270	71,125	100
0.02	0	-	-	20	11,551,660	1,058,316	100
0.01	0	-	-	11	90,454,300	6,519,236	98

inadequate gene exchange schedule. In the optimization stage, optimization is performed in parallel and the population size on island i is set to $|G_i|^2 \times 10/a$ where G_i is the set of loci in linkage group i. Thus, on the problems with smaller a, total population size soars higher. For example, in the case of $n = 12$ and

$a = 0.1$, total population is 2,400. That means a computational effort for 2,400 evaluations is consumed every time genes are exchanged among islands. We believe that LINC-R will be improved more by introducing effective gene exchange schedule.

The average fitness values at the end of trial runs are shown in figure 5. The optimal fitness is 6.0, 8.0, and 12.0 for $n = 12$, $n = 16$, and $n = 24$ respectively. ○ and △ indicate the average fitness obtained by LINC-R and the GA without linkage identification respectively. As a decreases and the problem gets harder, the performance of the GA without linkage identification inclines while LINC-R hold on to the optimum or near optimum.

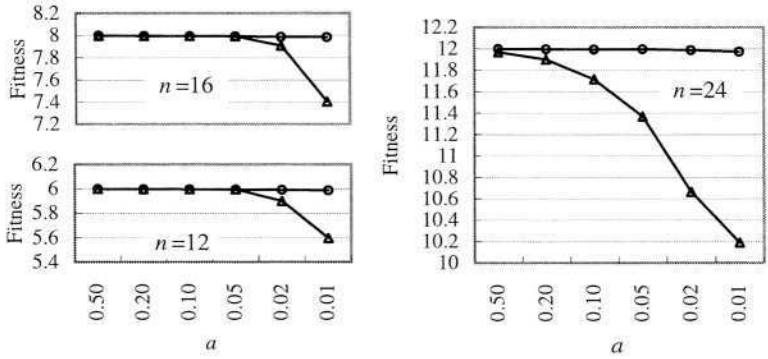


Fig. 5. ○ shows the average solution obtained by LINC-R and △ shows that by the GA without linkage identification.

5 Conclusion and Discussion

In this paper, we proposed Linkage Identification by Nonlinearity Check for Real-Coded GAs (LINC-R). When an objective function is decomposable to lower order sub-functions, the original problem can be yielded by solving the sub-problems independently and combining them. Linkage identification is a technique to recognize such sub-problems. A decomposed sub-problem is the set of unseparable loci and linear correlated loci are still decomposable. Therefore, the loci in the same linkage group should have nonlinear relation. Testing the equality of two partial differences of an objective function in the domain of the reals, LINC-R detects nonlinearity and reconginzes linkages accurately. The population size required for LINC-R at a certain probability is estimated as a function of the proportion of the nonlinear region in the domain.

It was shown that LINC-R outperformed PICI which identifies linkages according to correlation of loci on Type I and Type II benchmark functions.

Another experiment showed LINC-R with properly sized population had superior performance to the conventional GA without linkage identification on the additively decomposable problems.

If the proposed method is applied to the problems which are not additively decomposable, LINC-R does not detect any linkage groups and optimization is performed by one island. That means the method operates like the conventional GA on such problems except more computational effort is consumed for LINC-R. It is possible that the problems have the other kinds of decomposability such as non-monotonicity and epistasis. Real-world problems have some kinds of decomposablity. For example, a large supply chain can be decomposed to some small supply acitivities. In order to optimize real-world problems, LIMD and LIEM are planned to be introduced in real-valued problems in futur work.

References

1. Harik, G. R. and Goldberg, D. E.: Learning Linkage, In: Belew, R. K. and Vose, M. D. (eds.): Foundations of Genetic Algorithms IV (1996), pp. 247–262.
2. Pelikan, M. Goldberg, D. E., and Cantú-Paz, E.: BOA: The Bayesian Optimization Algorithm, Proc. the 1999 Genetic and Evolutionary Computation Conference (1999), 525-532.
3. Munetomo, M. and Goldberg, D. E.: Designing a genetic algorithm using the linkage identification by nonlinearity check, IlliGAL Technical Report (1998).
4. Munetomo, M. and Goldberg D. E.: Linkage Identification by Non-monotonicity Detection for Overlapping functions, Evolutionary Computation, vol. 7, no. 4 (1999), pp. 377–398.
5. Munetomo, M.: Linkage identification based on epistasis measures to realize efficient genetic algorithms, Proc. the 2002 Congress on Evolutionary Computation (2002).
6. Fogel, D. B.: Real-valued vectors, In: Bäck, T., Fogel, D. B. and Michalewicz, Z. (eds.): Handbook of Evolutionary Computation (1997), C1.3:1–2.
7. Wright, A. H.: Genetic Algorithms for Real Parameter Optimization, In: Rawlins, G. J. E. (ed.): Foundations of Genetic Algorithms (1991).
8. Michalewicz, Z. and Janikow, C. Z.: Handling Constraints in Genetic Algorithms, Proc. 4th Intl. Conf. on Genetic Algorithms (1991), pp. 151–157.
9. Eshelman, L. J. and Schaffer, J. D.: Real-Coded Genetic Algorithms and Interval-Schemata, In: Whitley, L. D. (ed.): Foundations of Genetic Algorithms 2 (1993).
10. Tsutsui, S., Goldberg, D. E., and Sastry, K.: Linkage Learning in Real-Coded GAs with Simplex Crossover, Proc. 5th Intl. Conf. Artificial Evolution (2001), pp. 73–84.
11. Tsutsui, S. and Goldberg, D. E.: Simplex Crossover and Linkage Identification: Single-Stage Evolution VS. Multi-Stage Evolution, Proc. the 2002 Congress on Evolutionary Computation (2002), pp. 974–979.
12. Higuchi, T., Tsutsui, S., and Yamamura, M.: Theoretical analysis of simplex crossover for real-coded Genetic Algorithms, Proc. PPSN VI (2000), pp. 365–374.
13. Sato, H., Yamamura, M., and Kobayashi, S.: Minimal generation gap model for GAs considering both exploration and exploitation, Proc. the IIZUKA'96 (1997), pp. 494–497.

Population-Based Iterated Local Search: Restricting Neighborhood Search by Crossover

Dirk Thierens

Institute of Information and Computing Sciences
Utrecht University, The Netherlands
dirk.thierens@cs.uu.nl

Abstract. Iterated local search (ILS) is a powerful meta-heuristic algorithm applied to a large variety of combinatorial optimization problems. Contrary to evolutionary algorithms (EAs) ILS focuses only on a single solution during its search. EAs have shown however that there can be a substantial gain in search quality when exploiting the information present in a population of solutions. In this paper we propose the use of a population for ILS. We define the general form of the resulting meta-heuristic, called population-based iterated local search (PILS). PILS is a minimal extension of ILS that uses previously generated solutions in the neighborhood of the current solution to restrict the neighborhood search by ILS. This neighborhood restriction is analogous to the way crossover preserves common substructures between parents when generating offspring. To keep the discussion concrete, we discuss a specific instantiation of the PILS algorithm on a binary trap function.

1 Introduction

In the field of meta-heuristics a large - and increasing - set of algorithms is studied to tackle hard combinatorial optimization problems typically found in Artificial Intelligence and Operational Research [11] [13] [16]. Most meta-heuristics try to find better solutions by exploring the neighborhood of a single solution. Examples are tabu search, simulated annealing, variable neighborhood search, greedy adaptive search procedure, and iterated local search. Other meta-heuristics bias their search by exploiting the information contained in a multi-set of current solutions. Examples of these population-based methods are evolutionary algorithms, ant colony optimization, and scatter search. All these meta-heuristics have their own view on how an efficient search process might be build. However, it is important to realize that their underlying philosophies are not necessarily incompatible. This observation creates the opportunity to design new meta-heuristics by combining the driving mechanism of two (or more) meta-heuristics. In this paper we propose a meta-heuristic that combines the power of iterated local search with the principle of extracting useful information about the search space by keeping a population of solutions. We call the resulting algorithm population-based iterated local search (PILS). As in ILS, PILS tries

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 234–245, 2004.

to improve on a single solution by running an iterated local search (ILS) algorithm. In addition to ILS, PILS also keeps a small population of neighboring solutions. These solutions are used to focus the iterated local search process to the common subspace between the current solution and one of the population members. The underlying assumption of PILS is that neighboring local optima share common substructures that could be exploited to generate new solutions. More specifically, PILS restricts the perturbation of ILS to the subspace where the current solution and a population member disagree, thus preserving their common substructure.

In the next section we formally describe the PILS meta-heuristic. This general algorithmic framework has to be instantiated for a particular problem type. In Section 3 we discuss such an instantiation for the trap function. Section 4 discusses related and future work. Finally, Section 5 concludes this preliminary work on the PILS meta-heuristic.

2 Population-Based Iterated Local Search

PILS is a population-based extension of the ILS meta-heuristic. ILS applies a local search procedure to explore the neighborhood of the current solution in search for a local optimum [8]. When a local optimum is reached, ILS perturbs this solution and continues the local search from this new solution. This perturbation should be large enough such that the local search does not return to the same local optimum in the next iteration. However the perturbation should not be too large, otherwise the search characteristics will resemble those of a multi-start local search algorithm. As long as the termination condition has not been met ILS continues its search this way. The underlying design philosophy of ILS - or in other words, its inductive bias - is to perform a stochastic neighborhood search in the space of local optima. Naturally, we do not have an operator that directly generates the neighborhood in this space, but the combination of the perturbation operator and the local search operator, both working in the original solution space, achieves this in an indirect way. The algorithmic description of ILS is as follows:

```
ITERATED LOCAL SEARCH()
1  S ← GENERATEINITIALSOLUTION
2  S ← LOCALSEARCH(S)
3  while NOTTERMINATED?(S)
4      do S' ← PERTURBATION(S, history)
5         S' ← LOCALSEARCH(S')
6         S ← ACCEPTANCECRITERION(S, S', history)
7  return S
```

The goal of the population-based iterated local search meta-heuristic is to improve upon the ILS algorithm by keeping a population of neighboring solutions, and exploit this information to focus the ILS algorithm. PILS explores the neighborhood of the current solution in two ways. With probability *Pratio*

it simply applies the regular ILS procedure. In the other case, a random member of the population is chosen and the `Perturbation`, and if meaningful for the particular problem, the `LocalSearch`, are restricted to the search subspace where the current solution and the population member disagree. The general algorithmic description of PILS is as follows:

```
POPULATION-BASED ITERATED LOCAL SEARCH()
 1  Pop ← CREATEINITIALPOP(PopSize)
 2  Pop[0] ← ACCEPTANCECRITERION(Pop)
 3  while NOTTERMINATED?(Pop)
 4     do if COINFLIP(Pratio)
 5           then S' ← PERTURBATION(Pop[0], history)
 6                S' ← LOCALSEARCH(S')
 7                Pop ← ACCEPTANCECRITERION(Pop, S', history)
 8           else i ← RANDOMINT(1, PopSize)
 9                S' ← PERTURBATION(Pop[0], Pop[i], history)
10                S' ← LOCALSEARCH(S', Pop[i])
11                Pop ← ACCEPTANCECRITERION(Pop, S', history)
12  return Pop[0]
```

The acceptance criterion could be the requirement that new solutions should have a better (or at least equal) fitness value than the current solution. Another criterion - applied in simulated annealing - accepts new solutions with a probability depending on the fitness difference between the current and new solution. The acceptance criterion might also keep a history list such that its decision depends on previously visited solutions. This approach is the main mechanism of tabu search.

When applying PILS to a particular class of problems the algorithmic framework needs to be instantiated in accordance with the characteristics of the underlying problem. To illustrate this we have applied the PILS meta-heuristic to a trap function, which is a hard and well studied optimization problem in EA research [1][3][15]. In the next section we specify one possible instantiation of the PILS meta-heuristic for the trap functions.

3 Example

3.1 Instantiating PILS for Trap Functions

In the previous section we have described the general population-based iterated local search algorithm. A specific instantiation of PILS requires the specification of the procedures `GenerateInitialSolution` (or `CreateInitialPop`), `LocalSearch`, `NotTerminated?`, `Perturbation`, and `AcceptanceCriterion`. In the remainder of this paper we will look at a specific PILS algorithm applied to a hard optimization problem in a fixed-length binary search space. First, we specify the ILS algorithm for this problem. `GenerateInitialSolution` simply generates an initial fixed-length binary string at random. The `NotTerminated?` test ensures

that we continue searching until we have reached the global maximum or we have exceeded a maximum number of trials. The `Perturbation` operator flips the bits with a fixed probability. The `LocalSearch` and the `AcceptanceCriterion` are combined to create a first-improvement hill-climbing algorithm. Each bit in the current string is flipped individually and the resulting fitness value is computed. Whenever there is a fitness improvement, or when the new value equals the previous one, the bit change is accepted. The search continues flipping bits at the current position in the string.

The PILS algorithm extends the ILS by exploiting information stored in other solutions in the population. More specifically, we adapt the `LocalSearch` and the `Perturbation` mechanisms by restricting the bits that they are allowed to change. `Perturbation` and `LocalSearch` now take two solutions as input: one is the current solution the search is focusing on, and the other is a randomly chosen solution from the population, which acts as a mask. The `Perturbation` operator flips with fixed probability only those bits that have a different value in the current solution and the mask. The combined `LocalSearch` and `AcceptanceCriterion` also restrict their first-improvement hill-climbing search to those bits that have a different value in the current solution and the mask. When a new solution is accepted it replaces the current solution which in turn replaces the worst solution in the population. The algorithmic description of the PILS meta-heuristic for the binary problem is given by:

```
PERTURBATION(BitString, MaskBitString, ProbMut)
1  for i ← 1 to LENGTH(BitString)
2     do if BitString(i) ≠ MaskBitString(i)
3           then if COINFLIP(ProbMut)
4                   then BitString ← FLIPBIT(Bitstring, i)
5  return BitString

FIRSTIMPROVEMENTHILLCLIMBING(BitString, MaskBitString)
1  for i ← 1 to StringLength
2     do if BitString(i) ≠ MaskBitString(i)
3           then NewBitString ← FLIPBIT(Bitstring, i)
4                if FITNESS(NewBitString) ≥ FITNESS(BitString)
5                   then BitString ← NewBitString
6  return BitString
```

The underlying assumption of the PILS algorithm is that neighboring local optima have some common substructures. By matching the current solution and a neighboring solution we try to identify these common substructures during the search process. The adaptive restriction of the ILS to a neighborhood with smaller dimension has two advantages. First, it ensures that the perturbation mechanism does not destroy the possibly useful common substructures. Second, the dimensionality reduction of the neighborhood to search might save considerable computational cost.

3.2 Fitness Function

To illustrate the use of the above PILS algorithm we measured its performance on a trap function consisting of misleading subfunctions of different lengths. Specifically, the fitness function $F(X)$ is constructed by adding subfunctions of length 1 (F_1), 2 (F_2), and 3 (F_3). Each subfunction has two optima: the optimal fitness value is obtained for an all-ones string, while the all-zeroes string represents a local optimum. The fitness of all other string in the subfunction is determined by the number of zeroes: the more zeroes the higher the fitness value. This causes a large basin of attraction toward the local optimum when applying the first-improvement hill-climber. The fitness values for the subfunctions are specified in Table 1 where the columns indicate the number of ones in the subfunctions F_1, F_2, and F_3. The fitness function $F(X)$ is composed of 4 subfunctions F_3, 6 subfunctions F_2, and 12 subfunctions F_1. The overall string-length of the problem is thus 36 (= 4 x 3 + 6 x 2 + 12 x 1). $F(X)$ has $2^{10} = 1024$ optima of which only one is the global optimum: the string with all ones having a fitness value of 220.

$$F(x_0 \ldots x_{35}) = \sum_{i=0}^{3} F_3(x_{3i}x_{3i+1}x_{3i+2}) + \sum_{i=0}^{5} F_2(x_{12+2i}x_{12+2i+1}) + \sum_{i=0}^{11} F_1(x_{24+i})$$

Table 1. The fitness values of the subfunctions F_i of length i; the columns represent the number of bits in the subfunction that are equal to one.

	0	1	2	3
F_3	4	2	0	10
F_2	5	0	10	
F_1	0	10		

3.3 Analysis

It is shown in [10] that the most difficult part for a local search algorithm when optimizing trap functions is the last step where a single, largest suboptimal subfunction needs to be changed in the optimal substring while preserving all the other optimal subfunctions. It is instructive to compute the probability this will take place with ILS and the PILS. Call *Pmut* the probability that a single bit is flipped by the `Perturbation` procedure. If we assume - without loss of generality - that the first-improvement hill-climber (FIHC) visits all bits from left to right then the basins of attraction of the two optima are:

111	111 011
000	000 100 010 001 110 101

The probability $Pr[\mathbf{xxx} \rightarrow \mathbf{yyy}]$ that the substring **xxx** is changed into **yyy** by the perturbation and local search (FIHC) algorithm is given by:

$Pr[111 \rightarrow 111]$	$(1-p_{mut})^2$
$Pr[111 \rightarrow 000]$	$1-(1-p_{mut})^2$
$Pr[000 \rightarrow 000]$	$1-p_{mut}^2$
$Pr[000 \rightarrow 111]$	p_{mut}^2
$Pr[11 \rightarrow 11]$	$1-p_{mut}$
$Pr[11 \rightarrow 00]$	p_{mut}
$Pr[00 \rightarrow 00]$	$1-p_{mut}$
$Pr[00 \rightarrow 11]$	p_{mut}
$Pr[1 \rightarrow 1]$	1
$Pr[1 \rightarrow 0]$	0
$Pr[0 \rightarrow 0]$	0
$Pr[0 \rightarrow 1]$	1

The probability that the ILS algorithm generates the optimal string when only one order-3 subfunction is incorrect is given by the probability that changes the suboptimal substring, while preserving the other optimal substrings:

$$\begin{aligned} Pr(success) &= Pr[000 \rightarrow 111](Pr[111 \rightarrow 111])^3(Pr[11 \rightarrow 11])^6 \\ &= p_{mut}^2(1-p_{mut})^{12}. \end{aligned}$$

The PILS algorithm can only generate the optimal string from the current solution with one suboptimal subfunction if the chosen solution from the population has the optimal substring at the corresponding position. Assuming the chosen population member has only one other suboptimal subfunction the probability that the PILS algorithm now generates the optimal solution is given by:

$$\begin{aligned} Pr(success) &= Pr[000 \rightarrow 111]Pr[111 \rightarrow 111] \\ &= p_{mut}^2(1-p_{mut})^2. \end{aligned}$$

Naturally, the solution picked from the population will quite often not have the optimal subfunction at the required position but this will be compensated by the much higher probability of generating the global optimal string in case the condition is fulfilled. Figure 1 shows the difference between the probabilities of generating the optimal string for both ILS and PILS in the above circumstances.

3.4 Experimental Results

To see how PILS performs for the above trap function and to investigate its sensitivity to certain parameter choices, we have performed some experiments and measured the number of function evaluations needed to generate the optimal string for the first time. The parameters studied are the population size N, the perturbation strength P_{mut}, the perturbation strength in the reduced subspace $P_{maskmut}$, and the ratio P_{ratio} between the ILS and the dimensionality reduced ILS. All results are reported by plotting the experimental cumulative distribution function of the probability of generating the global optimal solution as a function

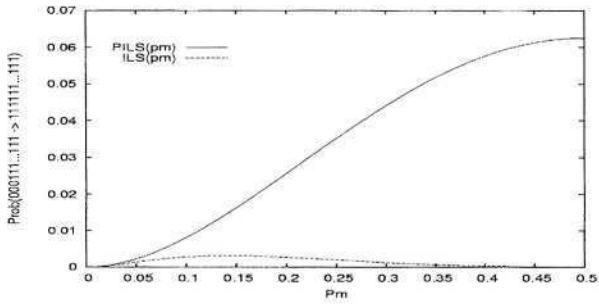


Fig. 1. Probability optima string generated from 000111...111 by ILS and PILS

of the number of function evaluations which is limited to a maximum of 100, 000. Every experimental CDF is calculated by running 1000 independent runs of the algorithm for a particular set of parameters.

1. Figure 2 shows the success probability as a function of the number of fitness function evaluation for ILS with varying perturbation strength. It can be observed that if the perturbation is too low the probability of finding a new local optimum is too low and the performance drops. If on the other hand the perturbation is too high the ILS becomes too destructive and its performance also degrades. Note that when *Pmut* = 0.5 ILS becomes a multi-start local search. From the size of the basins of attraction it is easy to calculate the success probability of generating the optimal string in this case, which is only 1 out of 2^{14} !

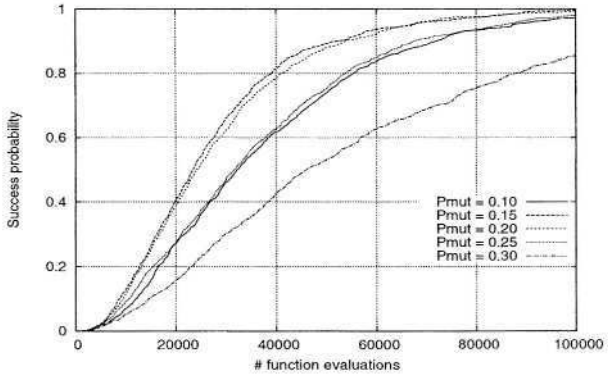


Fig. 2. ILS for varying perturbation strengths *Pmut*

2. In Figure 3 we have plotted the results of PILS for varying perturbation strengths when applying ILS to the entire - this is, the unrestricted - search space. The other parameters are fixed (N = 5, *Pratio* = 0.5, *Pmaskmut* = 0.25). Compared to the previous figure it is clear that PILS requires less function evaluations to reach a given performance level. In addition, PILS also seems to be more robust to the choice of parameter value of the perturbation strength.

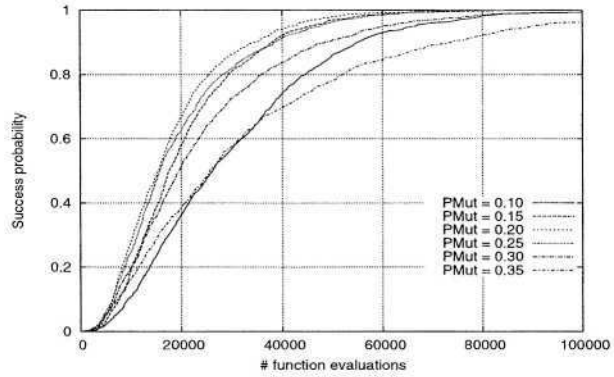


Fig. 3. PILS for varying perturbation strengths *Pmut* of the unrestricted perturbation with N = 5, *Pratio* = 0.5, *Pmaskmut* = 0.25

3. The previous experiments showed the sensitivity of ILS (either as standalone or within PILS) for the perturbation strength *Pmut*. Figure 4 shows that PILS is rather insensitive to the perturbation strength in the reduced search space when applying the dimensionality restricted ILS. Even when all bits in the restricted subspace are fully randomized (*Pmaskmut* = 0.5), the performance stays good.
4. Figure 5 shows that PILS is also rather insensitive to the size of the population of which the neighboring solutions are chosen to compute the restricted subspace to explore.
5. Finally in Figure 6 we have plotted the results for different values of the probability of applying unrestricted ILS or the restricted ILS. *Pratio* = 1.00 is actually the standard ILS algorithm: clearly PILS improves on this in each case. Note that *Pratio* = 0.00 makes no sense because parts of the search space would become unaccessible to the search algorithm.

4 Discussion

PILS tries to improve the performance of ILS by keeping a population of neighboring solutions to reduce the dimensionality of the neighborhood to be explored

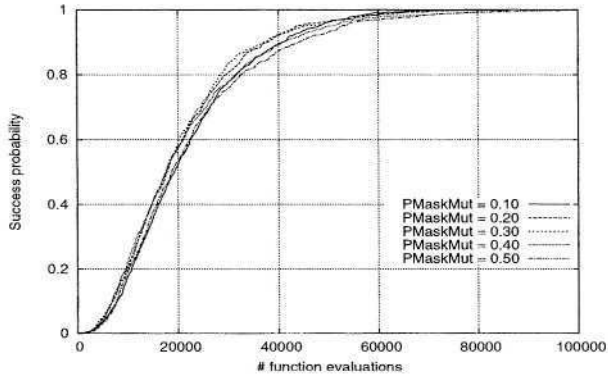


Fig. 4. PILS for varying perturbation strengths *Pmaskmut* of the restricted perturbation with $N = 5$, $Pratio = 0.5$, $Pmut = 0.15$

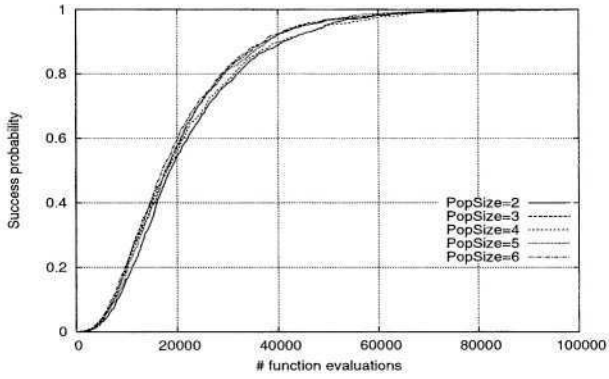


Fig. 5. PILS for varying population sizes N with $Pratio = 0.5$, $Pmut = 0.15$, $Pmaskmut = 0.25$

by ILS. By shielding off the common subspace between the current solution and a neighboring solution, PILS reduces the probability that common substructures found in good solutions are destroyed by the perturbation operator. Obviously, this mechanism is not unlike crossover in genetic algorithms. Although Holland [6] and Goldberg [4] consider crossover to be an operator that recombines or juxtaposes different substructures of two parent solutions, some crossover operators described in the literature could better be viewed as a perturbation operator in the search subspace defined by those variables where the two parent solutions have no common values. A good example in the binary search space is parameterized uniform crossover (PUX) where allele values between two strings are flipped with a fixed probability [14]. In fact because flipping equal bit values

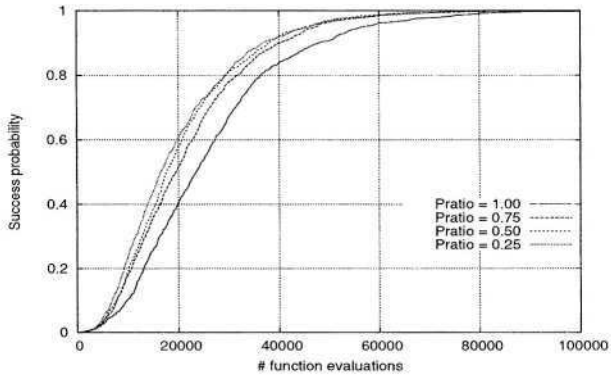


Fig. 6. PILS for varying probabilities *Pratio* between unrestricted and restricted perturbation and local search with $N = 5$, $Pmut = 0.15$, $Pmaskmut = 0.25$

does not produce a change, PUX is actually generating the same strings as the restricted perturbation operator we have applied in PILS.

The combination of local search with a population of solutions is also found in hybrid genetic algorithms, or memetic algorithms [2] [9]. Memetic algorithms are basically genetic algorithms that search in the space of local optima. Therefore, the recombination operator will in general try to juxtapose different substructures from the two parents. Also, selection is applied to let all solutions in the population compete for their survival. When using selection the population should not be too small to avoid the premature loss of diversity which would lead to unsatisfying search results. In many combinatorial optimization tasks the local search operator is a computationally expensive procedure, and memetic algorithms often need to limit their use of the local search operator due to the high computational cost. In PILS the search is only focused on a single - for instance, the current best - solution as in ILS. No selection between members of a population is taking place, other than replacing the current solution by a new neighboring solution. The population in PILS are highly fit solutions encountered during the search, and are only used to help define a reduced neighborhood around the current solution.

Recently, it has been brought to our attention that Reeves also recognized the role crossover might play as a neighborhood restricting operator [12]. Reeves focused on the description of the working of a GA in terms of a generalized neighborhood search algorithm. He also noted that even the restricted neighborhood would in general be too large to search exhaustively, and while a traditional GA can be viewed as taking a random sample from that neighborhood, Reeves advocated the use of a more systematic local search. In PILS the restricted neighborhood is searched by ILS, which is a very efficient neighborhood search algorithm.

5 Conclusions

We have proposed the population-based iterative local search (PILS) algorithm. PILS aims to extend the efficiency of the iterative local search algorithm by exploiting information contained in a population of neighboring solutions. By using a population PILS restricts the ILS algorithm to explore only lower dimensional neighborhoods. The gain in efficiency is obtained at two levels. First, the perturbation operator is restricted to explore only the subspace where the current solution and a neighboring solution from the population disagree. This way promising, partial solutions remain untouched. Second, the local search operator only needs to search a neighborhood of smaller size. We have analyzed and tested PILS on trap functions, showing how and why PILS can be more efficient than ILS. A key assumption of the PILS algorithm is that local optimal solutions possess common substructures that can be exploited to increase the efficiency of the ILS. In future work we will investigate the use of PILS on other combinatorial optimization problems.

Acknowledgments. I would like to thank Ken De Jong, Colin Reeves, and Peter Merz for the valuable discussion at the Dagstuhl-seminar 04081 'Theory of Evolutionary Algorithms'.

References

1. D.H. Ackley. *A connectionist machine for genetic hill climbing,* Kluwer Academic Publishers, 1987.
2. D. Corne, F. Glover, and M. Dorigo (eds.). *New Ideas in Optimization,* McGraw-Hill, 1999.
3. K. Deb, and D.E. Goldberg. Analysing deception in trap functions. *Proceedings of Foundations of Genetic Algorithms,* pages 93–108, Morgan Kaufmann, 1993.
4. D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989.
5. D. Frost, I. Rish, and L. Vila. Summarizing CSP hardness with continuous probability distributions. *Proceedings of the Fourteenth National Conference on Artificial Intelligence,* pages 327–334, AAAI-Press, 1997.
6. J.H. Holland. *Adaptation in Natural and Artificial Systems.* Michigan University Press, Ann Arbor, 1975.
7. H. Hoos, and T. Stützle. Towards a Characterisation of the Behaviour of Stochastic Local Search Algorithms for SAT. *Artificial Intelligence,* vol. 112, nrs. 1-2, pages 213-232, Elsevier Science Publishers, 1999.
8. H.R. Lourenço, O. Martin, and T. Stützle. A Beginner's Introduction to Iterated Local Search. *Proceedings of the 4th Metaheuristics International Conference,* 2001.
9. P. Merz, and B. Freisleben. Fitness Landscapes and Memetic Algorithm Design. *New Ideas in Optimisation,* D. Corne, M. Dorigo, and F. Glover (eds.) McGraw-Hill, 1999.
10. H. Mühlenbein. How genetic algorithms really work. I. Mutation and Hillclimbing. *Proceedings of Parallel Problem Solving from Nature,* pages 15–25, North-Holland, 1992.

11. I.H. Osman, and J.P. Kelly (eds.). *Meta-Heuristics: The Theory and Applications,* Kluwer Academic Publishers, Boston, 1996.
12. C.R. Reeves. Genetic Algorithms and Neighbourhood Search. *Proceedings of Evolutionary Computing, AISB Workshop,* pages 115–130, 1994.
13. C.C. Ribeiro, and P. Hansen (eds.). *Essays and Surveys in Metaheuristics,* Kluwer Academic Publishers, Boston, 2001.
14. W.M. Spears, and K.A. De Jong On the Virtues of Parametrized Uniform Crossover. *Proceedings of the Fourth International Conference on Genetic Algorithms,* pages 230-236, Morgan Kaufmann, 1991.
15. D. Thierens, and D. Goldberg. Mixing in genetic algorithms. *Proceedings of the Fifth International Conference on Genetic Algorithms,* pages 38-45, Morgan Kaufmann, 1993.
16. S. Voss, S. Martello, I.H. Osman, and C. Roucairol (eds.). *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization,* Kluwer Academic Publishers, Boston, 1999.

Modeling Dependencies of Loci with String Classification According to Fitness Differences

Miwako Tsuji, Masaharu Munetomo, and Kiyoshi Akama

Hokkaido University, North 11, West 5, Sapporo, 060-0811 Japan.
{m_tsuji,munetomo,akama}@cims.hokudai.ac.jp

Abstract. Genetic Algorithms perform crossovers effectively when we can identify a set of loci tightly linked to form a building block. Several methods have been proposed to detect such linkage. Linkage identification methods investigate fitness differences by perturbations of gene values and EDAs estimate the distribution of promising strings. In this paper, we propose a novel approach combining both of them, which detects dependencies of loci by estimating the distribution of strings classified according to fitness differences. The proposed algorithm called the Dependency Detection for Distribution Derived from *df* (DDDDD or D^5) can detect dependencies of a problem which is difficult for EDAs requiring lower computation cost than linkage identifications.

1 Introduction

According to the building block hypothesis [6], genetic algorithms (GAs) implicitly decompose a problem into sub-problems by processing building blocks. Therefore proper building block mixing is essential for GAs. However, the exchange operators of traditional GAs such as one- or two- point crossovers depend on the order of variables on a string. If variables of a same sub-problem are encoded loosely on a string, the crossover operators disrupt building blocks — tight linkage is necessary for effective genetic recombination. A set of loci tightly linked to form a building block is called a linkage set. If such set of loci is known, GA can perform mixing more efficiently.

In order to ensure appropriate mixing, several methods have been proposed. Most of them are categorized as:

1. Linkage Identification Methods
2. Estimation of Distribution Algorithms (EDAs)

The Algorithms classified in the first category examine fitness difference by perturbations in loci to detect dependency. For example, the Gene Expression Messy GA (GEMGA) [7] records fitness differences by perturbation of every locus for strings in population and detects relations of genes according to possibilities that the loci construct local optima. The Linkage Identification by Nonlinearity Check (LINC) [11] identifies linkage by detecting the second order nonlinearity. It assumes that nonlinearity must exist within loci to form a building block. If

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 246–257, 2004.

a fitness difference by simultaneous perturbations at a pair of loci is equal to the sum of fitness differences by perturbations at each locus in the pair, these loci can be optimized separately; otherwise, they are considered to be linked. Heckendorn et. al. [5] generalized this category through a Walsh analysis.

In these algorithms, the amount of fitness difference depends only on a sub-problem including the perturbed locus and contributions from the other sub-solutions are canceled by subtracting fitness of perturbed string from fitness of original string. Therefore their abilities are not affected by the difference of fitness contribution of each sub-solutions. In addition, they get to know problem structure before they start searching, and so their search procedures themselves are generally efficient. However, because most of them need to evaluate fitness differences by pairwise or more perturbations, they require $O(l^2)$ or more fitness evaluations where l is a string length.

For the second category, EDAs construct a probabilistic model of promising solutions and use that model to generate better solutions. Early EDAs such as the Population Based Incremental Learning Algorithm (PBIL) [14] and the Compact Genetic Algorithm (CGA) [4], evaluate distribution of gene value in every locus independently and assume no dependency of variables. Subsequent works such as the Mutual Information Maximization for Input Clustering (MIMIC) [1], the Factorized Distribution Algorithm (FDA) [8] and the Bayesian Optimization Algorithm (BOA) [13] exploit conditional probabilities to encode dependency of variables in their models.

Because EDAs need no additional fitness evaluation for their modeling processes, the computational cost for fitness evaluations on problems constructed of sub-problems having uniformly scaled fitness contribution is lower than the methods in the first category. However, for problems constructed of sub-problems having various fitness contribution, sub-problems which give small fitness contribution can not be modeled appropriately. This is because such sub-problems can be selected in promising sub-population only if they come with sub-problems with large fitness contribution.

In this paper, we propose a novel approach hinted from both EDAs and linkage identifications in order to detect dependencies. The proposed approach can detect dependencies of problems which are difficult for EDAs with less computational cost than linkage identifications. Although the proposed approach estimates sub-population like EDAs, the sub-population is selected according to fitness differences by perturbations instead of absolute fitness values. Because the fitness difference depend only on a sub-problem including the perturbed locus, distribution of the sub-population has information of the sub-problem. Just as linkage identifications using perturbations are not greatly affected by the scaling of fitness contribution of each sub-function, the proposed approach can detect sub-solutions even if they give a small contribution to the whole fitness value. The proposed approach can detect dependencies accurately by quasi-linear number of fitness evaluations for string length.

We introduce our new method called the Dependency Detection for Distribution Derived from *df* (DDDDD or D^5) and its mechanism is explained in

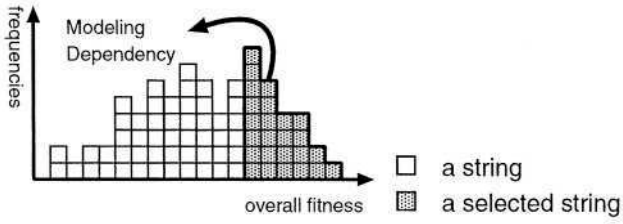


Fig. 1. Existing EDAs

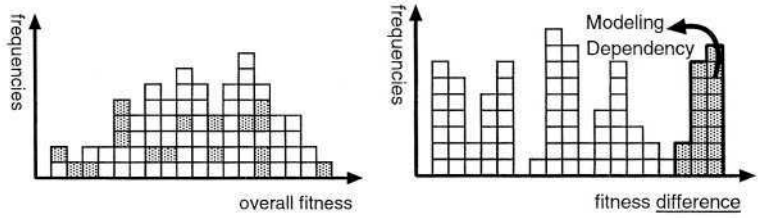


Fig. 2. Strings selected for modeling in D^5

section 2. The results of experiment are shown in section 3. The conclusion and discussion are mentioned in section 4.

2 Modeling Dependencies from Distributions of Strings Classified According to Fitness Differences

Existing EDAs sometimes fail to learn models for sub-problems having little contribution to overall fitness value. This is because they estimate sub-population selected according to the overall fitness value (Fig. 1) and such small contributions can not affect the overall fitness value. If EDAs try to capture such sub-problems, they need to prepar larger number of individuals. For example, the BOA needs $O(l^{1.55})$ if sub-functions are scaled uniformly but it requires $O(l^2)$ if sub-functions are scaled exponentially [12]. On the other hand, linkage identification methods using fitness differences can identify linkages for sub-problems which have small contribution, because the fitness differences for a sub-problem are not affected by the other sub-problems. However, in contrast to EDAs, which need no additional fitness evaluation for modeling, most linkage identification methods need relatively large number of fitness evaluations, $O(l^2)$ where l is string length, to check interdependency of a pair of loci even when a problem is scaled uniformly.

For each locus i, the D^5 calculates fitness difference by a perturbation at locus i, classifies strings according to the difference, and estimates classified strings in order to detect dependency. Because the D^5 selects sub-population according to fitness difference (Fig.2 right), even a sub-problem having only a small contribution can be captured. Although it needs additional fitness evaluations

1. initialize population with n strings
2. for each locus i
   a) calculate fitness difference $df_i(s^p)$ by a perturbation at locus i in string s^p $(p = 1, 2, \cdots, n)$
   b) classify strings according to their fitness differences into sub-populations (see Fig. 4 for detail)
   c) estimate sub-populations and construct linkage sets (see Fig. 5 for detail)

Fig. 3. Overall Algorithm of linkage identification in the D^5

to investigate fitness differences, required difference is by a single perturbation. Therefore, the number of evaluations is quasi linear for l, which is considerably less than that of linkage identifications $O(l^2)$ especially for large l.

In the followings, we show the detail of the algorithm and its mechanism.

2.1 The Algorithm of the D^5

Fig. 3 shows the proposed algorithm. The algorithm consists of three parts : (1) calculation of fitness differences, (2) classification of strings according to the differences and (3) estimation of the classified strings.

After initializing population, following procedures are repeated for each locus i : At first, locus i in each string s^p is perturbed and then fitness difference for the perturbation is calculated as follows:

$$df_i(s^p) = f(s^p) - f(s_i^p) \tag{1}$$

In the above equation, s_i^p is a string perturbed ($0 \rightarrow 1$ or $1 \rightarrow 0$) at locus i. Then, strings are classified into sub-populations according to the fitness difference $df_i(s^p)$. The classification method will be mentioned in subsection 2.2. Finally, the sub-populations are estimated in order to detect loci which depend on locus i. This stage will be mentioned in subsection 2.3.

2.2 Classification According to Fitness Differences

Fig.4 shows a classification method we employed. We employ a centroid linkage method for classification. In this method, the centroid of a cluster is determined by averaging $df_i(s^p)$ of all strings within that cluster. The distance between two clusters is defined as the distance between the centroids of the clusters. The pair of clusters having the smallest distance are merged until a termination criteria — number of clusters becomes smaller than the pre-defined threshold and distance $df_i(c)$ reaches to the above upper bound of distance — is satisfied.

Although this simple classification is enough for quasi-decomposable problem, as future work, other classification methods should be investigated for complex problem.

1. initialize classes
   one class c^p includes one string s^p and fitness difference of the class $df_i(c^p)$ is set to $df_i(s^p)$
2. estimate $|df_i(c^p) - df_i(c^p)|$ for all pair of class (c^p, c^q) and merge the pair with smallest difference.
3. update $df_i(c)$ of new class to average df_i of all strings in the class
4. repeat 2 and 3 until a termination criteria are met.

Fig. 4. Classification Algorithm

1. for each sub-population p classified by the Classification Algorithm
   a) initialize set of loci $v_1 = \{1, 2, \cdots, i-1, i+1, \cdots, l\}$ and $v_2 = \{i\}$
   b) while $|v_2| < K$, where K is pre-defined problem complexity
      i. calculate a entropy $E_j = E(v_2 \cup \{j\})$ for all locus $j \in v_1$
      ii. $h = \arg\min_{j \in v_1} E_j$
      iii. update $v_1 = v_1 - \{h\}$ and $v_2 = v_2 \cup \{h\}$
   c) $v_p = v_2$ and $E_p = E(v_2)$
2. select v_p with the smallest E_p as the linkage set for locus i

Fig. 5. Construct Linkage Set

2.3 Construction of Linkage Sets

Fig. 5 is the algorithm to construct a linkage set for locus i. First, the set is initialized as $\{i\}$. The locus which gives the smallest entropy joining the linkage set is merged repeatedly until the size of linkage set exceeds pre-defined problem complexity k. The entropy measure is used by Harik [3] and defined as

$$E(v_2) = -\sum_{x=1}^{2^{|v_2|}} p_x \log_2 p_x \tag{2}$$

where n is population size p_x is the appearance ratio of each schema x and $2^{|v_2|}$ is the number of all possible schema defined by v_2. In our algorithm, if $p_x = 0$, $\log_2 p_x$ is set to 0 for convenience. The reason why we can identify a linkage set by finding v_2 which gives the smallest $E(v_2)$ is described in the next subsection.

This procedure is applied for all sub-populations except those including small number of strings. This is because that estimation of distribution from small samples has risk of unreliable result. Finally the linkage set giving the smallest entropy is selected as linkage set v_i of locus i from linkage sets obtained on each sub-population. If there are several sub-populations which give the smallest entropy, then the one with larger sub-population size seems to have more reliability because small sub-populations are likely to have unexpected bias.

Table 1. Order 3 sub-problem

solution	fitness
111	30
110	0
101	0
100	14
011	0
010	22
001	26
000	28

Table 2. s, $f(s)$, $f(s_1)$ and df_1

s	$f(s)$	$f(s_1)$	df_1
110001000	54	81	27
111111101	60	30	-30
101010111	52	78	26
000111010	80	66	-14
110111000	58	85	27
001111011	56	30	-26
000110101	28	14	-14
100110111	44	58	14
011011101	0	30	30
100011100	28	42	14
111011100	44	14	-30
111000001	84	54	-30
110101000	28	55	27
111100100	58	28	-30
000011110	28	14	-14
110111101	30	57	27
000111010	85	71	-14
000000001	82	68	-14
000010001	81	67	-14
	...		
	...		

Table 3. Strings classified according to df_1

df_1	s	schema
30	011011101	011******
30	011111010	011******
30	011111011	011******
30	011001100	011******
30	011010001	011******
30	011110011	011******
27	110100100	110******
27	110111000	110******
27	110001000	110******
27	110011101	110******
27	110101000	110******
27	110111101	110******
27	110101001	110******
27	110101001	110******
26	101010111	101******
26	101101000	101******
26	101110011	101******
26	101100011	101******
26	101010000	101******
	...	
	...	

2.4 Mechanism of the D^5

In EDAs, estimated sub-population must have the information about problem structure to detect dependency of variables. In other words, the estimated sub-population must have a biased and determinate distribution. It is clear that there is not any kind of information in a completely random population i.e. an initial population and that no model can be constructed from such population. Most EDAs select strings with relatively high fitness value to obtain the information from them. However, as mentioned before, the methods have difficulties in capturing sub-problems having a small fitness contribution.

In this part, we describe how the D^5 obtains such biased sub-populations by classification according to fitness differences. First, we give an explanation using a concrete problem and then we show a formal exposition.

Table. 1 shows an order 3 deceptive problem which was used as an opponent of the messy GA [2]. Consider a fitness function composed of sum of the 3 sub-functions. For example, we obtain strings with $f(s), f(s_1)$ and df_1 for the problem as shown in Table. 2. In this table, $f(s_1)$ is a fitness value of a string perturbed at locus 1 and $df_1 = f(s_1) - f(s)$ is a fitness difference. These strings are classified as in Table. 3. It is clear that loci which depend on locus 1 (locus 2 and 3) have same gene values. In contrast to the case of locus 2 and 3, loci which does not compose a same sub-function with the 1-st locus distribute randomly.

In sub-population having $df_1 = 30$, linkage set $\{1, 2, 3\}$ has only schema 011 and $E(\{1, 2, 3\})$ should be zero. On the other hand, linkage set $\{1, 4, 5\}$ has schemata 010, 001, 010 and $E(\{1, 2, 3\})$ should be relatively large. Therefore the algorithm evaluate that a relationship between locus 1, 2, and 3 take place more likely than a relationship between locus 1, 4, and 5.

In the followings we give a further explantion using some equations.

We assume that problem is composed as sum of completely separable sub-problems like

$$f(s) = \sum_{v \in V} f_v(s). \tag{3}$$

where v is a set of loci which compose a sub-problem $f_v(s)$ and V is a set of linkage groups (a set of a set of loci). This class of problems are known as additively decomposable functions.

Let $\hat{v}$ the sub-problem including locus i then equation (3) is represented as

$$f(s) = f_{\hat{v}}(s) + \sum_{v \neq \hat{v}, v \in V} f_v(s). \tag{4}$$

It is clear that $f_v(s)$ is calculated independently of locus i if v does not include i. Therefore the following equation is true :

$$\sum_{v \neq \hat{v}, v \in V} f_v(s) = \sum_{v \neq \hat{v}, v \in V} f_v(s_i) \tag{5}$$

where s_i is a string having same values as s in all loci except locus i. From the above equation, $df_i(s)$ is represented as follows :

$$\begin{aligned} df_i(s) &= f(s) - f(s_i) \\ &= [f_{\hat{v}}(s) + \sum_{v \neq \hat{v}, v \in V} f_v(s)] - [f_{\hat{v}}(s_i) + \sum_{v \neq \hat{v}, v \in V} f_v(s_i)] \\ &= f_{\hat{v}}(s) - f_{\hat{v}}(s_i). \end{aligned} \tag{6}$$

Accordingly, it is said that $df_i(s)$ is defined independently of loci $j \notin \hat{v}$ and depend only on loci $j \in \hat{v}$.

The entropy $E(v_2)$ in equation (2) is a measure of uncertainty of distribution of schemata. Therefore, if schemata defined by v_2 distribute randomly, $E(v_2)$ takes large value. Because $df_i(s)$ dose not depend on loci $j \notin \hat{v}$, such loci distribute randomly, such linkage sets give large $E(v_2)$ over sub-population classified according to $df_i(s)$. On the other hand, if the distribution of schemata is biased, $E(v_2)$ becomes small. Since the sub-population is classified by $df_i(s)$ and $df_i(s)$ is depend on loci $j \in \hat{v}$, a correct linkage set $\hat{v}$ can identify by finding v_2 which minimize $E(v_2)$.

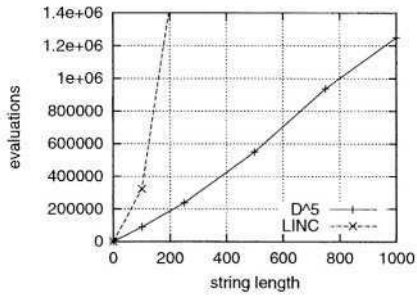


Fig. 6. Results of D^5 and LINC for 5-bit Trap Function

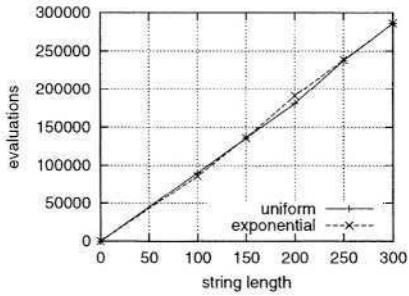


Fig. 7. Results of D^5 for Trap Function with Uniformly Scaled Sub-Functions and Trap Function with exponentially Scaled Sub-Functions

3 Experiments

In this section, we give simulation results of the D^5 in comparison to the LINC and its variations. Experiments in subsection 3.1 and 3.2 examine efficiency of the algorithms for a problem composed of uniformly scaled trap sub-problems and a problem composed of exponentially scaled trap sub-problems respectively. An experiment in subsection 3.3 investigates the ability of the algorithms for a problem having not only interdependencies in sub-functions but overall complexity.

3.1 Uniformly Scaled 5-Bits Trap Function

First test function is defined as the sum of uniformly scaled 5-bit trap sub-function defined as equations (7) and (8).

If accurate problem structure is known, following optimization will be success. Therefore, we examine how many evaluations of fitness function are needed to obtain correct dependency for all loci. The numbers of evaluations for detecting accurate problem structure are recorded for various string lengths (i.e. problem size). Since all test functions we use are composed of sum of sub-functions having deceptiveness, it is considered that when loci in the same sub-function with locus i's are detected perfectly for all locus i in 10 independent runs, problem structure is obtained accurately.

$$f(s) = \sum_{i=1}^{m} \mathrm{trap}_5(s_{5\cdot i}, \cdots, s_{5\cdot i+4}) \tag{7}$$

$$\mathrm{trap}_5(s_{5\cdot i}, \cdots, s_{5\cdot i+4}) = \begin{cases} 5 & \text{if} \quad u = 5 \\ 4 - u & \text{otherwise} \end{cases} \tag{8}$$

where u is the number of ones in each 5 bit substring $s_{5\cdot i}, \cdots, s_{5\cdot i+4}$ and m is the number of sub-functions. In our experiment, we try various m and change overall string length $l = 5 \times m$

Fig.6 shows the number of evaluations on the 5-bit trap function. It is clear that the number of evaluations required by the D^5 is far smaller than the LINC and exceeds $O(l)$ slightly.

3.2 Exponentially Scaled 5-Bits Trap Function

Second test function is the sum of 5-bit trap sub-function having exponentially increasing contribution.

$$f(s) = \sum_{i=1}^{m} 2^i \mathrm{trap}_5(s_{5\cdot i}, \cdots, s_{5\cdot i+4}) \tag{9}$$

where $\mathrm{trap}_5(s_{5\cdot i}, \cdots, s_{5\cdot i+4})$ is a same function as (8). Contribution of i-th sub-function is controlled by 2^{i-1}.

In this function, a solution which has high fitness values in 1, 2, $\cdots, m-1$ -th sub-functions and low fitness value in m-th sub-function can not be modeled due to inequality $\sum_{i=1}^{m-1} 2^i < 2^m$. Therefore, EDAs should model each sub-function one by one from the one having large contribution to the one having small contribution and can not get accurate model for sub-function having small contribution immediately. As mentioned in section 2, the BOA needs $O(l^{1.55})$ if sub-functions are scaled uniformly but it requires $O(l^2)$ if sub-functions are scaled exponentially.

Fig.7 shows the number of fitness evaluations for the uniformly scaled function and the exponentially scaled function by the D^5. The solid line shows result for the uniformly scaled function and the dotted line shows result for the exponentially scaled function. It is said that the D^5 can solve a problem which has exponentially scaled sub-problems as efficient as a problem which has uniformly scaled sub-problems because two lines overlap each other considerably. This is because fitness differences for perturbations of loci are not affected by nonuniform scaling. Note that also the LINC can perform on the uniformly scaled function as on the exponentially scaled function but it needs $O(l^2)$ fitness evaluations for both functions.

3.3 Overlapped Function

In this experiment, we use the sum of trap sub-functions, but having overall nonlinear interdependency as follow:

$$f(s) = [\sum_{i=1}^{10} \mathrm{trap}_5(s_{5\cdot i}, \cdots, s_{5\cdot i+4})]^m \tag{10}$$

Table 4. Average and Standard Deviation of Accuracy of linkage identification

m	D^5-800		D^5-1600		LIEM-32		LIEM-64		$LIEM^2$-32		$LIEM^2$-64	
	Avg.	S.D.	Avg.	S.D.	Avg.	S.D.	Avg.	S.D.	Avg.	S.D.	Avg.	S.D.
1	1.00	0.00	1.00	0.00	0.95	0.04	1.00	0.00	0.95	0.06	1.00	0.00
2	0.99	0.01	1.00	0.00	0.96	0.04	1.00	0.00	0.94	0.05	1.00	0.00
3	0.96	0.03	1.00	0.00	0.94	0.04	1.00	0.00	0.95	0.04	1.00	0.00
4	0.86	0.05	0.94	0.05	0.79	0.08	0.98	0.03	0.95	0.04	1.00	0.00
5	0.68	0.06	0.88	0.06	0.31	0.11	0.49	0.07	0.95	0.05	1.00	0.00

where $m = 1, 2, \cdots, 5$ and $\mathrm{trap}_5(s_{5\cdot i}, \cdots, s_{5\cdot i+4})$ is an equation (8). This function was proposed to ensure the performance of the LIEM and the $LIEM^2$, which are expansion of the LINC for overlapped function [9,10]. Problem length is fixed to $5 \times 10 = 50$ and a strength of overall complexity m is varied from 1 to 5. The accuracy of linkage identification is measured and is compared with the LIEM and the $LIEM^2$.

In the D^5, we employ fixed population size $n = 800$ and $n = 1,600$ for all $m = 1, \cdots, 5$ therefore the number of fitness evaluations is 40,800 and 81,600 respectively. In the classification phase, sub-populations are merged until the number of clusters becomes 8. In the estimation phase, sub-populations which have 8 or lower strings are ignored due to their unreliability. The population size for the LIEM and the $LIEM^2$ is $n = 32$ and $n = 64$. The numbers of fitness evaluations are 40,832 for $n = 32$ and 81,664 for $n = 64$.

Table. 4 shows the ratio of correct linkage groups in 10 independent runs for equation (10). The numbers following the name of the algorithms are their population size. As mentioned before, the number of fitness evaluations of D^5 using 800 strings is roughly the same as the number of the LIEM and the $LIEM^2$ using 32 strings. For the problem of $m = 1, 2, 3$, the D^5-800 can perform best for the number of fitness evaluations. However, when $m = 4, 5$, the $LIEM^2$-32 gives the best result of the three algorithms. The D^5-800 can perform better than the LIEM-32 for these problems.

The number of fitness evaluations of the D^5 using 1,600 strings is also roughly same as the number of the LIEM and the $LIEM^2$ using 64 strings. All of the three can identify all linkage sets accurately for the problem of $m = 1, 2, 3$. For the problem of $m = 4$, the D^5-1600 and the LIEM-64 also give high score but they can not archive complete identification like the $LIEM^2$-64. For the problem of $m = 5$, the accuracy of the LIEM-64 decreases considerably and it is lower than the D^5-1600 and even the D^5-800.

Because of relatively small problem size, the number of evaluations of all algorithms is set equivalent. However, it is clear from the experiments in sub-sections 3.1 and 3.2 that the number of evaluation of the D^5 is smaller than the LIEM and the $LIEM^2$ for large problem size.

The horizontal axes of the histograms in Fig. 8–10 show normalized fitness difference by the perturbation at a locus and the vertical axes is the number

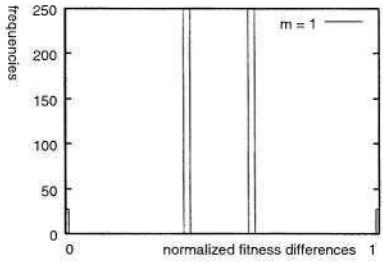


Fig. 8. Histogram of $m = 1$ problem

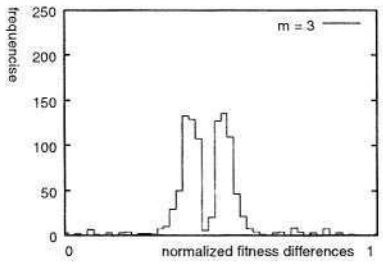


Fig. 9. Histogram of $m = 3$ problem

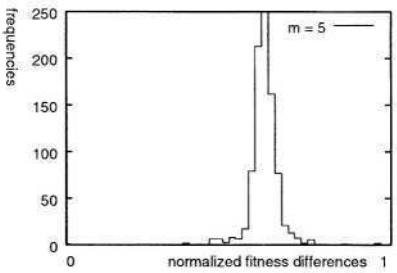


Fig. 10. Histogram of $m = 5$ problem

of strings showing the fitness difference. It is clear that the function of $m = 1$ has the character that is easy to be classified. The both sides of strings have sub-solution of (8) of $u = 5$ and sub-solution of $u = 4$ which became $u = 5$ by 1-bit perturbation respectively. On the other hand, it seems that the function of $m = 5$ has the character that is difficult to be classified.

It is said that although the D^5 uses only quasi linear fitness evaluations, it can solve problem having weak overall complexity.

4 Discussion and Conclusion

In this paper, we propose a novel approach to detect dependency between loci. The proposed D^5 estimates dependencies of loci based on sub-populations classified according to fitness differences by perturbation at each locus. The fitness difference depends only on loci constructing a same sub-problem as a perturbed locus. Therefore the classified sub-populations has biased distribution in such loci and and and dependencies of a problem are obtained by estimating the sub-populations.

The D^5 is hinted from two existing method (1) Linkage Identification Methods and (2) Estimation of Distribution Algorithms (EDAs). It can detect dependencies with smaller number of fitness evaluations than that of the linkage identifications, because it needs only $O(l)$ perturbations while linkage identifications need $O(l^2)$ perturbations for each pair of loci. In addition, the D^5 can solve

problems consisting of exponentially scaled sub-functions, which is difficult for the EDAs and is applicable to problems having weak overall complexity.

The D^5 can be located in the place where existing methods can not perform well. For future work, more powerful classification method should be investigated for complex problems.

References

1. Jeremy S. De Bonet, Jr. Charles L. Isbell, and Paul Viola. MIMIC: Finding optima by estimating probability densities. *Advances in Neural Information Processing Systems,* 9:424–430, 1997.
2. David E. Goldberg, Bradley Korb, and Kalyanmoy Deb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems,* 3(5):415–444, 1989.
3. Georges Harik. Linkage learning via probabilistic modeling in the ECGA. Technical Report IlliGAL Report No.99010, University of Illinois at Urbana-Champaign, Urbana, IL, 1999.
4. Georges Harik, Fernando G. Lobo, and David E. Goldberg. The compact genetic algorithm. *Proceedings of the 1998 IEEE Conference on Evolutionary Computation,* pages 523–528, 1998.
5. Robert B. Heckendorn and Alden H. Wright. Efficient linkage discovery by limited probing. In *Proceedings of the 2003 Genetic and Evolutionary Computation Conference,* pages 1003–1014. Morgan Kaufmann Publishers, 12–16 July 2003.
6. John H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.
7. Hillol Kargupta. The gene expression messy genetic algorithm. In *International Conference on Evolutionary Computation,* pages 814–819. Springer Verlag, 9 1996.
8. Heinz Mühlenbein and Thilo Mahnig. FDA – a scalable evolutionary algorithm for the optimization of additively decomposed functions. *Evolutionary Computation,* 7(4):353–376, 1999.
9. Masaharu Munetomo. Linkage identification based on epistasis measures to realize efficient genetic algorithms. In *Proceedings of the 2002 Congress on Evolutionary Computation,* pages 445–452, 2002.
10. Masaharu Munetomo. Linkage identification with epistasis measure considering monotonicity conditions. In *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning,* 2002.
11. Masaharu Munetomo and David E. Goldberg. Identifying linkage groups by nonlinearity/non-monotonicity detection. In *Proceedings of the 1999 Genetic and Evolutionary Computation Conference,* 1999.
12. Martin Pelikan. *Bayesian Optimization Algorithm: From Single Level to Hierarchy.* Doctroral dessertation, University of Illionis at Urbana-Champaign, 2002.
13. Martin Pelikan, David E. Goldberg, and Erick Cantú-Paz. BOA: The bayesian optimization algorithm. In *Proceedings of the 1999 Genetic and Evolutionary Computation Conference,* pages 525–532. Morgan Kaufmann Publishers, 1999.
14. Rafal P. Salustowicz and Jürgen Schmidhuber. Probabilistic incremental program evolution: Stochastic search through program space. In M. van Someren and G. Widmer, editors, *Machine Learning: ECML-97,* volume 1224, pages 213–220. Springer-Verlag, 1997.

The Edge-Set Encoding Revisited: On the Bias of a Direct Representation for Trees

Carsten Tzschoppe[2], Franz Rothlauf[1], and Hans-Josef Pesch[2]

[1] Department of Information Systems 1
University of Mannheim
68131 Mannheim/Germany
rothlauf@uni-mannheim.de
[2] Department of Applied Mathematics
University of Bayreuth
95440 Bayreuth/Germany
carsten.tzschoppe@gmx.de,hans-josef.pesch@uni-bayreuth.de

Abstract. The edge-set encoding is a direct tree representation which directly represents trees as sets of edges. There are two variants of the edge-set encoding: the edge-set encoding without heuristics, and the edge-set encoding with heuristics. An investigation into the bias of the edge-set encoding shows that the crossover operator of the edge-set encoding without heuristics is unbiased, that means it does not favor particular types of trees. In contrast, the crossover operator with heuristics is biased towards the simple minimum spanning tree (MST) and generates more likely trees that are MST-like. As a result, the performance of the edge-set encoding without heuristics does not depend on the structure of the optimal solution. Using the heuristic crossover operator results only in high genetic algorithm (GA) performance if the optimal solution of the problem is slightly different from the simple MST. However, if the optimal solution is not very similar to the simple MST a GA using the heuristic crossover operator fails and is not able to find the optimal solution. Therefore, it is recommended that the edge-set encoding with heuristics should only be used if it is known a priori that the optimal solution is very similar to the simple MST. If this is not known a priori, other unbiased search operators and representations should be used.

1 Introduction

A spanning tree of an undirected graph G is a subgraph that connects all vertices of G and contains no cycles. Relevant constraint minimum spanning tree (MST) problems are, for example, the optimal communication spanning tree (OCST) problem, or the degree-constrained minimum spanning tree problem [1,2,3]. The NP-hard OCST problem [4] seeks a spanning tree that connects all given nodes and satisfies their communication requirements for a minimum total cost. Genetic algorithms (GAs) have been applied with success to many constrained MST problems. As it is well known that the proper design of operators and representations is crucial for GA performance [5], a large variety of tree representations

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 258–270, 2004.

like NetKeys [6], the link-and-node-biased encoding [7], or Prüfer numbers [8, 9] have been developed. Recently, [3] proposed a new direct representation of trees, the edge-set encoding, which has successfully been used for the degree-constrained MST problem, and has outperformed other representations such as Prüfer numbers or NetKeys.

The purpose of this paper is to investigate the properties of the edge-set encoding. The paper focuses on the crossover operators, heuristic and non-heuristic KruskalRST*, used for the edge-set encoding and examines whether they are biased that means they overrepresent specific tree structures. Furthermore, the performance of the crossover operators is compared for OCST problems. The results show that the heuristic crossover operator is strongly biased towards the simple MST, whereas the non-heuristic crossover operator shows no bias. Consequently, GA performance increases when using the heuristic crossover operator for OCST problems where the optimal solution is only slightly different from the simple MST. In contrast, when applying the heuristic crossover operator to OCST problems where the optimal solution is not similar to the simple MST, GAs using the edge-set encoding fail.

The paper is structured as follows. The following section gives a short definition of the OCST problem and describes the functionality of the edge-set encoding with and without heuristics. Section 3 investigates the bias of the two variants of the crossover operator of the edge-set encoding. After investigating the bias, section 4 examines the influence of the crossover operator with and without heuristics on the performance of GAs when solving the OCST problem. The paper ends with concluding remarks.

2 Setting up the Stage: The OCST Problem and the Edge-Set Encoding

2.1 The OCST Problem

The optimal communication spanning tree problem (also known as minimum spanning tree problem or simple network design problem) was first introduced in [4]:

Definition 1 (Optimal Communication Spanning Tree Problem). *Let $G = (V, E)$ be a complete undirected graph. $n = |V|$ denotes the number of nodes in the graph and $m = |E|$ denotes the number of edges in the graph. To every pair of nodes (i, j) a non-negative weight w_{ij} and a non-negative communication requirement r_{ij} is associated. The communication cost $c(T)$ of a spanning tree T is defined as*

$$c(T) = \sum_{i,j \in V,\, i<j} r_{ij} \cdot w(p_{i,j}^T),$$

where $w(p_{i,j}^T)$ denotes the weight of the unique path from node i to node j in the spanning tree T. The OCST Problem seeks the spanning tree with minimal costs among all other spanning trees.

Definition 2 (Minimum Spanning Tree Problem). *The OCST problem becomes the minimum spanning tree (MST) problem if there are no communication requirements r_{ij} and $c(T)$ depends only on the weights w_{ij}. Then, T is the simple minimum spanning tree if $c(T) \leq c(T')$ for all other spanning trees T', where $c(T) = \sum_{(i,j)\in T} w_{ij}$.*

The similarity between two spanning trees T_i and T_j can be measured using the distance $d_{ij} \in \{0, 1, \ldots, n-1\}$ as $d_{ij} = \frac{1}{2}\sum_{u,v\in V,\, u<v} |l^i_{uv} - l^j_{uv}|$, where l^i_{uv} is 1 if a link from u to v exists in T_i and 0 if it does not exist in T_i.

Like many other constrained spanning tree problems, the OCST problem is NP-hard [10]. Even more, it was later shown that the OCST problem is $\mathcal{MAX}$ $\mathcal{SNP}$-hard [11], that means it cannot be solved using a polynomial-time approximation-scheme, unless $\mathcal{P} = \mathcal{NP}$. Nevertheless, the OCST problem has been studied extensively in the literature and many researchers have tried to develop efficient approximation algorithms. The current best approximation algorithm for the general OCST problem approximates the optimal solution with $c(T) = O(\log n \log\log n) \cdot c(G)$ [12], where $c(G)$ is the cost of the network when using G. $c(G)$ is a lower bound for the cost of a spanning tree $c(T)$ as the weight of the unique path between node i and j in a spanning tree T is greater or equal in comparison to the weight of the path with minimal weight connecting the nodes i and j in G. As there are no efficient approximation algorithms, many researchers used GAs for solving the OCST problem [13,14,15,16,5,17].

It was shown in [18] that the optimal solution of a OCST problem is similar to the simple MST, that means the average number of different edges between the OCST and the simple MST is significantly lower than the average number of different edges between a randomly generated tree and the simple MST. Therefore, as the optimal solution of an OCST problem is biased towards the simple MST, representations as well as operators that favor or overrepresent trees, which are similar to the MST are expected to solve the OCST problem more efficiently.

2.2 The Edge-Set Encoding without Heuristics

The edge-set encoding [3] is a direct tree representation that means it directly represents trees as sets of edges. In the following paragraphs we describe how initial populations are created and explain the functionality of the crossover and mutation operator of the edge-set encoding without heuristics.

Initialization: In order to create feasible solutions for the initial population, the edge-set encoding uses the Kruskal random spanning tree (RST) algorithm, a slightly modified version of the algorithm from Kruskal. In contrast to Kruskals' algorithm, KruskalRST chooses edges (i, j) not according to their weight w_{ij} but randomly. [3] have shown that this algorithm for creating random spanning trees, KruskalRST, has a small bias towards star-like trees.

procedure KruskalRST(V, E):
$T \leftarrow \emptyset$, $A \leftarrow E$;

while $|T| < |V| - 1$ **do**
  choose an edge $\{(u, v)\} \in A$ at random;
  $A \leftarrow A - \{(u, v)\}$;
  if u and v are not yet connected in T **then**
    $T \leftarrow T \cup \{(u, v)\}$;
return T.

[3] also presented two other RST algorithms (PrimRST, RandWalkRST) for generating the initial population, but RandWalkRST has an unlimited worst-case running time, and PrimRST has a stronger bias in comparison to KruskalRST.

Recombination: The functionality of the crossover operator is straightforward. To obtain an offspring T_{off} from two parental trees T_1 and T_2, KruskalRST is applied to the graph $G_{cr} = (V, T_1 \cup T_2)$. Therefore, the resulting crossover operator has high heritability as in the absence of constraints, only parental edges are used to create the offspring. Crossover becomes more complicated for constraint MST problems as it is possible that the RST algorithm can create no feasible tree from $G_{cr} = (V, T_1 \cup T_2)$. Then, additional edges have to be chosen randomly to complete an offspring. Based on KruskalRST, [3] distinguished two different recombination operators: The variant previously described is denoted KruskalRST crossover. The second variant is denoted KruskalRST* crossover. When using this variant, edges that are common to both parents T_1 and T_2 are included in the offspring T_{off} before KruskalRST crossover is applied. Results from [3] indicated a better performance of the KruskalRST* crossover for the degree-constraint MST problem.

Mutation: The mutation operator randomly replaces one edge in the spanning tree. This replacement can be implemented in two ways. The first variant of the mutation operator chooses randomly one edge from $E \setminus T$ and includes it in T. This creates a cycle. Then, the operator randomly chooses one edge from the cycle and removes it from T ("insertion before deletion"). The second variant first randomly deletes one edge from T and connects then the two disjoint connected components using a random edge from $E \setminus T$ ("deletion before insertion").

2.3 The Edge-Set Encoding with Heuristics

The following paragraphs describe how heuristics that rely on the weights w_{ij} can be included in the edge-set encoding.

Heuristic Initialization: To favor low-weighted edges when generating the initial population, the algorithm KruskalRST starts with sorting all edges in the underlying graph according to their weights w_{ij} in ascending order. The first spanning tree is created by choosing the first edges in the ordered list. As these are the edges with lowest weights, the first generated spanning tree is a simple MST. Then, the k edges with lowest weights are permuted randomly and more spanning trees are created again using the first edges in the list. Therefore, the

heuristic initialization results in a strong bias of the initial population towards the simple MST. With increasing k the bias of the randomly created trees towards the simple MST is reduced. The number of edges, which are permuted increases according to

$$k = \alpha(i-1)n/N,$$

where N denotes the population size, i is the number of the tree that is actually generated $(i = 1 \ldots N)$ and α is a parameter that controls the strength of the heuristic bias.

Heuristic Recombination: The heuristic recombination operator is a modified version of KruskalRST* crossover. Firstly, the operator transfers all edges $T_1 \cap T_2$ that exist in both parents to the offspring. Then, the remaining edges are chosen randomly from $E' = (T_1 \cup T_2) \setminus (T_1 \cap T_2)$ using a tournament with replacement of size two. If the underlying optimization problem is constrained, it is possible that the offspring has to be completed using edges not in E'.

Heuristic Mutation: The heuristic mutation operator is based on mutation by "insertion before deletion". In a pre-processing step, all edges in the underlying graph are sorted according to their weights in ascending order. Doing this, a rank is assigned to every edge. To favor low-weighted edges, edges are not chosen randomly but according to their ranks

$$R = \lfloor |\mathcal{N}(0, \beta n)| \rfloor \bmod (m+1),$$

where $\mathcal{N}(0, \beta n)$ is the normal distribution with mean 0 and standard deviation βn. β is a parameter that controls the bias towards low-weighted edges.

3 Investigating the Bias of the Edge-Set Encoding

A representation is unbiased if all possible solutions of the search space are represented uniformly [5]. Consequently, a search operator is unbiased if it does not overrepresent specific solutions, and the application of the search operator alone does not modify the statistical properties of a population. An unbiased search operator allows a uniform, non-directed search through the search space. A biased representation resp. operator should only be used if it is known a priori that the optimal solution of the underlying optimization problem is similar to the overrepresented solutions [19]. In contrast, unbiased representations resp. operators should be used if no a priori problem-specific knowledge is available. Then, the probability of finding the optimal solution is independent of the structure of the optimal solution.

To investigate if the crossover operator of the edge-set encoding with or without heuristics leads to an overrepresentation of MST-like individuals, we randomly generate an initial population with 500 individuals and apply the crossover operator iteratively. As no selection operator is used, no selection pressure exists that pushes the population to high-quality solutions. The crossover

operator is unbiased if the statistical properties of the population do not change by applying crossover alone. In our experiments we measure in each generation the average distance $d_{mst-pop} = 1/N \sum_{i=1}^{n} d_{i,MST}$ of the individuals T_i in the population to the simple MST. If $d_{mst-pop}$ decreases, the crossover operator is biased towards the simple MST. In contrast, if $d_{mst-pop}$ remains constant, the crossover operator is unbiased and no solutions are overrepresented.

To obtain meaningful results, we performed this experiment on 50 randomly generated ten and 16 node problem instances with random, resp. Euclidean weights w_{ij}. For every problem instance we performed 50 runs with different, randomly chosen initial populations. In each run, the crossover operator is applied 100 times (generations). The communication requirements of the problem instances with random and Euclidean weights w_{ij} are uniformly distributed real values from [0, 100]. The random distance weights w_{ij} are real values and uniformly distributed from [0,100]. When using Euclidean weights, all nodes are randomly placed on a 1000 × 1000 grid and the Euclidean distances between the nodes are taken as weights. As the weights w_{ij} are randomly created and $w_{ij} \neq w_{kl}$, $\forall i \neq l, j \neq l$, there is an unique optimum MST for every problem instance and distances to the simple MST are unique.
Figure 1 shows the mean and the standard deviation of the distance $d_{mst-pop}$ between the individuals in a population towards the simple MST over the number of generations for ten (top) and 16 (bottom) problem instances. The plots compare the non-heuristic KruskalRST* crossover with the heuristic KruskalRST* crossover operator (no selection is used). The results reveal that the crossover operator without heuristics is unbiased and does not modify the statistical properties of the population ($d_{mst-pop}$ remains constant over the number of generations). In contrast, the crossover operator with heuristics shows a strong bias towards the simple MST and the population converges quickly to the simple MST.

4 The Performance of the Edge-Set Encoding for the OCST Problem

4.1 Finding Optimal Solutions for OCST Problems

To investigate how the performance of the edge-set encoding depends on the structure of the optimal solution, an optimal or near-optimal solution must be determined. The following experiments, which should identify optimal or near-optimal solutions for OCST problems, are similar to the ones described in [18]. They examined the OCST problem and showed that optimal solutions of OCST problems are biased towards the simple MST.

To determine the optimal (or near optimal) solution we apply a GA n_{iter} times to an OCST problem using a population size of N_0. T_0^{best} denotes the best solution of cost $c(T_0^{best})$ that is found during the n_{iter} runs. In a next round we double the population size and again apply a GA n_{iter} times with a population size of $N_1 = 2 \cdot N_0$. T_1^{best} denotes the best solution with cost $c(T_1^{best})$ that

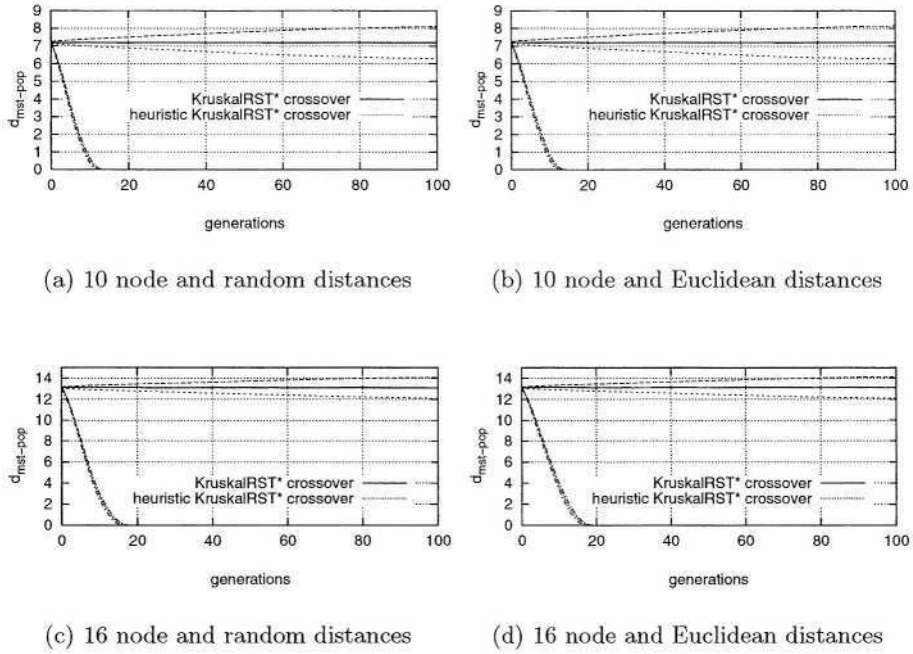


Fig. 1. The plots show the mean and the standard deviation of the distance $d_{mst-pop}$ between a population of 500 randomly generated individuals towards the simple MST over the number of generations when using crossover only (no selection pressure). Results are presented for ten (top) and 16 (bottom) problem instances using either random (left) or Euclidean (right) weights w_{ij}. The results show that the non-heuristic KruskalRST* crossover is unbiased that means the average distance between the population and the simple MST remains constant. The crossover operator with heuristics shows a strong bias towards the simple MST and the population converges to the simple MST in a few generations.

can be found in the second round. We continue these iterations and double the population size $N_i = 2N_{i-1}$ until $c(T_i^{best}) = c(T_{i-1}^{best})$. This means we stop if the cost of the best solution T_i^{best} found in round i equals the cost of the best solution T_{i-1}^{best} found in round $i-1$. We assume that the solutions found using this approach are optimal or near-optimal.

Figure 2 presents the results of our experiments. We show the number of problem instances over the distance $d_{opt,MST}$ between the best found solution and the simple MST for randomly created ten (Fig. 2(a)) and 16 (Fig. 2(b)) node OCST problem instances. We distinguish between random and Euclidean weights w_{ij}. For every problem instance we randomly generated 200 OCST problem instances. We used an initial population size $N_0 = 20$ for the ten node problem instances and $N_0 = 100$ for the 16 node problem instances, $n_{iter} = 20$, a standard GA with a NetKey representation [6], tournament selection without replacement

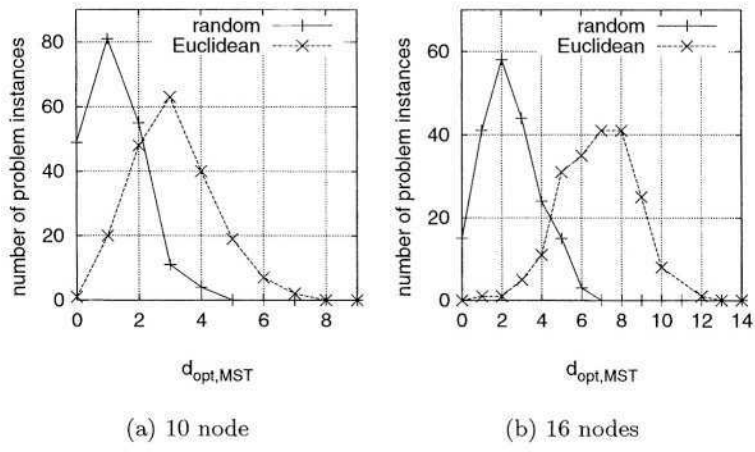


Fig. 2. We randomly generated 200 OCST problems and show the distribution of the problem instances over the distance $d_{opt,MST}$ between the best found solution and the simple MST when using either random or Euclidean distance weights for ten (left) and 16 (right) node problems. The plots show that the optimal solutions for OCST problems using random distance weights are stronger biased in comparison to using Euclidean weights.

of size two, uniform crossover and no mutation. The results are similar to the ones presented in [18] and show that the best found solution is strongly biased towards the simple MST. Furthermore, OCST problems using random distance weights show a stronger bias in comparison to OCST problems using Euclidean weights.

4.2 Comparing Heuristic and Non-heuristic Crossover Operators

After determining optimal or near-optimal solutions as described in the previous paragraph we examine the performance of the edge-set encoding on these 200 problem instances. We use the same randomly generated problem instances as in section 4.1 and investigate how the performance of the edge-set encoding using different types of crossover operators depends on the distance $d_{opt,MST}$ between the optimal (or near-optimal) solution and the simple MST. For comparing the performance of the two crossover operators, KruskalRST* and heuristic KruskalRST*, we use a standard GA with no mutation and tournament selection without replacement of size two. The initial population is generated using the non-heuristic KruskalRST (compare Sect. 2.2). Each run is stopped after the population is fully converged or the number of generations exceeds 200. We perform 100 runs for each of the 200 problem instances.

The population size N is chosen with respect to the performance of the crossover operator without heuristics (KruskalRST*). The aim is to find the

optimal solution with a probability of about 50 %. Therefore, we choose for the ten node problems a population size of $N = 60$ (random weights) resp. $N = 100$ (Euclidean weights) and for the 16 node problems a population size of $N = 200$ (random weights) resp. $N = 450$ (Euclidean weights).

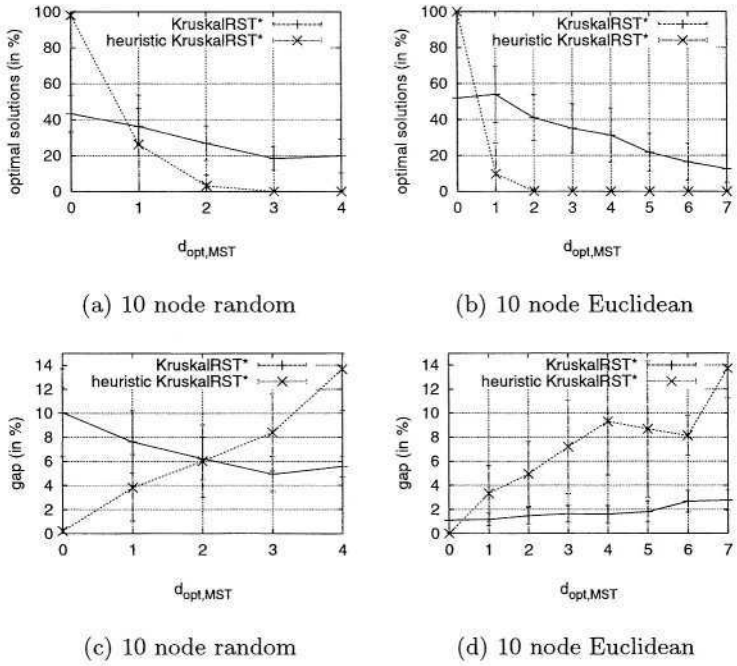


Fig. 3. The plots compare the performance of different crossover operators of the edge-set encoding (KruskalRST* versus heuristic KruskalRST*) for randomly generated ten node OCST problem instances. The plots 3(a) and 3(b) show the mean and standard deviation of the percentage of optimal solutions that can be found over $d_{opt,MST}$. The plots 3(c) and 3(d) show the mean and standard deviation of the gap between the cost of the best found solution and the cost of the optimal solution determined in section 4.1. The results are averaged over 200 randomly created OCST problems using either random (left) or Euclidean (right) weights. The plots show that the heuristic KruskalRST* crossover outperforms the non-heuristic version only if the optimal solution is very similar to the simple MST ($d_{opt,MST} \approx 0$). If the difference between the optimal solution and the simple MST becomes greater the heuristic KruskalRST* crossover results in low GA performance and the optimal solution cannot be found. In contrast, when using the non-heuristic KruskalRST* crossover, GA performance remains about constant with increasing $d_{opt,MST}$.

The results of our experiments are presented in Figure 3 (ten nodes) and Figure 4 (16 nodes). We show results for random (left) and Euclidean (right) weights.

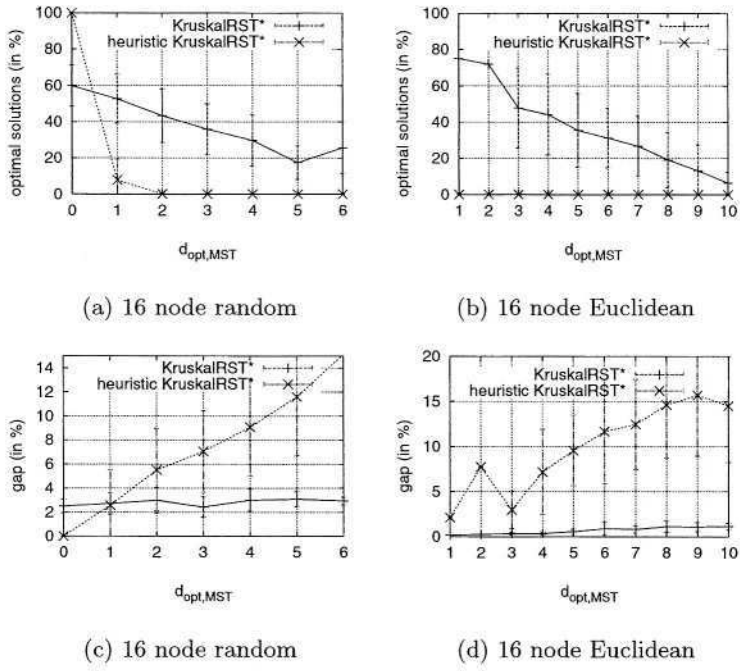


Fig. 4. The figures compare the performance of different crossover operators for randomly generated 16 node OCST problem instances. The plots 4(a) and 4(b) show the mean and standard deviation of the percentage of optimal solutions that can be found over $d_{opt,MST}$. The plots 4(c) and 4(d) show the mean and standard deviation of the gap between the cost of the best found solution and the cost of the optimal solution determined in section 4.1. The results are averaged over 200 randomly created OCST problems using either random (left) or Euclidean (right) distance weights. The plots show that the heuristic KruskalRST* crossover outperforms the non-heuristic KruskalRST* crossover only if the optimal solution is very similar to the simple MST ($d_{opt,MST} \approx 0$).

The top of Figure 3 and 4 shows the percentage of GA runs that correctly identify the optimal solutions at the end of a run over the distance $d_{opt,MST}$ between the optimal solution (compare section 4.1) and the simple MST. The bottom of the figures show the gap, $\frac{c(T_{found})-c(T_{opt})}{c(T_{opt})}$ (in percent), between the cost of the best found solution and the cost of the optimal solution that was identified in section 4.1 over $d_{opt,MST}$. The results reveal that the heuristic crossover operator, heuristic KruskalRST*, always finds the optimal solution if the optimal solution is the simple MST ($d_{opt,MST} = 0$). However, with increasing $d_{opt,MST}$ GA performance is significantly reduced and for $d_{opt,MST} \geq 3$ the optimal solution can not be found any more. In contrast, the performance of the crossover operator

without heuristics decreases only slightly with larger $d_{opt,MST}$ and allows the GA to correctly identify the optimal solution even for larger $d_{opt,MST}$.

The direct comparison between the performance of the two crossover operators reveals that the heuristic crossover performs well only for problems where the optimal solution is slightly different from the simple MST. Otherwise, GAs using the edge-set encoding with heuristic crossover fail. These results are confirmed when examining the gap $\frac{c(T_{found})-c(T_{opt})}{c(T_{opt})}$ (bottom of Figure 3 and 4). Heuristic crossover shows perfect performance if the optimal solution is the simple MST. However, with increasing $d_{opt,MST}$ the quality of the solutions strongly decreases and the non-heuristic KruskalRST* outperforms the heuristic variant.

5 Summary and Conclusions

This work investigates two different crossover variants of the edge-set encoding, heuristic KruskalRST* crossover versus KruskalRST* crossover, which were proposed by [3]. Section 2 defines the optimal communication spanning tree (OCST) problem and describes the functionality of the edge-set encoding. In section 3 an investigation into the bias of the crossover operators is performed. Based on an analysis of optimal solutions for randomly generated instances of the OCST problem, section 4.2 investigates how the performance of the crossover operators of the edge-set encoding depends on the similarity between the optimal solution of an OCST problem and the simple minimal spanning tree (MST).

The investigation into the bias of the crossover operators of the edge-set encoding reveals that the heuristic KruskalRST* crossover is strongly biased towards the simple MST. In contrast to the unbiased, non-heuristic KruskalRST* that results in an uniform search through the search space, the population converges quickly towards the simple MST if the heuristic KruskalRST* crossover is used. Therefore, due to the strong bias towards the simple MST, GAs using the edge-set encoding with the heuristic KruskalRST* crossover can easily solve OCST problems if the optimal solution is the simple MST. However, with decreasing similarity between the optimal solution of an OCST problem and the simple MST, the edge-set encoding with heuristics fails as heuristic search gets stuck at the simple MST. In contrast, GAs using the edge-set encoding with the unbiased KruskalRST* crossover operator show good performance for all different OCST problems independently of the similarity between the optimal solution and the MST. The results suggest that the edge-set encoding with the heuristic KruskalRST* crossover operator is not appropriate for solving OCST problems. This search operator can only be used successfully if the optimal solutions are the simple MST, or slightly different variants of it.

The problems of the heuristic crossover operator of the edge-set encoding emphasizes the difficulty of a proper design of representations and operators. Especially the design of direct representations is difficult as in contrast to indirect representations, the behavior of new, problem-specific search operators is often unknown. The analysis of the edge-set encoding has shown that although optimal solutions for the OCST problems are biased towards the simple MST

[18], direct representations resp. operators like the heuristic KruskalRST* that use this problem-specific knowledge and are biased towards the simple MST, can fail in solving most of the randomly created OCST problem instances. Therefore, the authors recommend the use of unbiased representations if no problem-specific knowledge is known a priori. Proper representations for tree problems are for example non-heuristic versions of the edge-set encoding or NetKeys [6]. In the case that biased representations resp. operators are used, it must be confirmed that the bias of the search fits to the properties of the optimal solutions. Otherwise failure is unavoidable.

References

1. Narula, S.C., Ho, C.A.: Degree-constrained minimum spanning trees. Computers and Operations Research **7** (1980) 239–249
2. Fekete, S., Khuller, S., Klemmstein, M., Raghavachari, B., Young, N.: A network-flow technique for finding low-weight bounded-degree spanning trees. Journal of Algorithms **24** (1997) 310–324
3. Raidl, G.R., Julstrom, B.A.: Edge-sets: An effective evolutionary coding of spanning trees. IEEE Transactions on Evolutionary Computation **7** (2003) 225–239
4. Hu, T.C.: Optimum communication spanning trees. SIAM Journal on Computing **3** (1974) 188–195
5. Rothlauf, F.: Representations for Genetic and Evolutionary Algorithms. Number 104 in Studies on Fuzziness and Soft Computing. Springer, Berlin (2002) 1st edition 2002. 2nd printing 2003.
6. Rothlauf, F., Goldberg, D.E., Heinzl, A.: Network random keys – A tree network representation scheme for genetic and evolutionary algorithms. Evolutionary Computation **10** (2002) 75–97
7. Palmer, C.C., Kershenbaum, A.: Representing trees in genetic algorithms. In: Proceedings of the First IEEE Conference on Evolutionary Computation. Volume 1., Piscataway, NJ, IEEE Service Center (1994) 379–384
8. Prüfer, H.: Neuer Beweis eines Satzes über Permutationen. Archiv für Mathematik und Physik **27** (1918) 742–744
9. Gottlieb, J., Julstrom, B.A., Raidl, G.R., Rothlauf, F.: Prüfer numbers: A poor representation of spanning trees for evolutionary search. In Spector, L., Goodman, E., Wu, A., Langdon, W.B., Voigt, H.M., Gen, M., Sen, S., Dorigo, M., Pezeshk, S., Garzon, M., Burke, E., eds.: Proceedings of the Genetic and Evolutionary Computation Conference 2001, San Francisco, CA, Morgan Kaufmann Publishers (2001) 343–350
10. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, New York (1979)
11. Papadimitriou, C.H., Yannakakis, M.: Optimization, approximation, and complexity classes. J. Comput. System Sci. **43** (1991) 425–440
12. Charikar, M., Chekuri, C., Goel, A., Guha, S., Plotkin, S.: Approximating a finite metric by a small number of tree metrics. In: Proc. 39th IEEE Symp. on Foundations of Computer Science. (1998) 111–125
13. Palmer, C.C.: An approach to a problem in network design using genetic algorithms, unpublished PhD thesis, Polytechnic University, Troy, NY (1994)

14. Berry, L.T.M., Murtagh, B.A., McMahon, G.: Applications of a genetic-based algorithm for optimal design of tree-structured communication networks. In: Proceedings of the Regional Teletraffic Engineering Conference of the International Teletraffic Congress, Pretoria, South Africa (1995) 361–370
15. Li, Y., Bouchebaba, Y.: A new genetic algorithm for the optimal communication spanning tree problem. In Fonlupt, C., Hao, J.K., Lutton, E., Ronald, E., Schoenauer, M., eds.: Proceedings of Artificial Evolution: Fifth European Conference, Berlin, Springer (1999) 162–173
16. Kim, J.R., Gen, M.: Genetic algorithm for solving bicriteria network topology design problem. In Angeline, P.J., Michalewicz, Z., Schoenauer, M., Yao, X., Zalzala, A., Porto, W., eds.: Proceedings of the 1999 IEEE Congress on Evolutionary Computation, IEEE Press (1999) 2272–2279
17. Chou, H., Premkumar, G., Chu, C.H.: Genetic algorithms for communications network design - an empirical study of the factors that influence performance. IEEE Transactions on Evolutionary Computation **5** (2001) 236–249
18. Rothlauf, F., Gerstacker, J., Heinzl, A.: On the optimal communication spanning tree problem. Technical Report 15/2003, University of Mannheim (2003)
19. Rothlauf, F., Goldberg, D.E.: Redundant representations in evolutionary computation. Evolutionary Computation **11** (2003) 381–415

A Gene Based Adaptive Mutation Strategy for Genetic Algorithms

Sima Uyar, Sanem Sariel, and Gulsen Eryigit

Istanbul Technical University, Electrical and Electronics Faculty
Department of Computer Engineering, Maslak TR-34469 Istanbul, Turkey
{uyar, sariel, gulsen}@cs.itu.edu.tr

Abstract. In this study, a new mechanism that adapts the mutation rate for each locus on the chromosomes, based on feedback obtained from the current population is proposed. Through tests using the one-max problem, it is shown that the proposed scheme improves convergence rate. Further tests are performed using the 4-Peaks and multiple knapsack test problems to compare the performance of the proposed approach with other similar parameter control approaches. A convergence control scheme that provides acceptable performance is chosen to maintain sufficient diversity in the population and implemented for all tested methods to provide fair comparisons. The effects of using a convergence control mechanism are not within the scope of this paper and will be explored in a future study. As a result of the tests, promising results which promote further experimentation are obtained.

1 Introduction

Genetic algorithms belong to a class of biologically inspired optimization approaches that model the basic principles of classical Mendelian genetics and Darwinian theory of evolution. Due to their robust nature, genetic algorithms are used in a wide variety of applications. However one of the major drawbacks is that performance largely depends on the appropriate setting of some parameters: population size, crossover and mutation rates. These parameters interact with each other, making it even harder to find optimal settings. However mutation rate is considered to be the most sensitive of these parameters. Mutation has been traditionally regarded as a background operator that mainly works as an insurance policy protecting alleles from being lost from the population. There has been extensive work to investigate the exact nature of mutation and to find optimal settings for different classes of problems [2], [7], [8]. It has also been shown in further studies [2], [3], [12] that using a varying mutation rate strategy overcomes the difficulties of finding optimal mutation rate settings. The techniques developed to set the parameters are classified separately by Eiben et al. [5] Angeline [1] and Smith et al. [10] but the main underlying principles of the different classifications are similar. Parameter setting methods can be classified into two major categories: *parameter tuning* and *parameter control.* In parameter tuning, the parameter values are set in advance, before the run and are kept constant during the whole execution of the algorithm. In parameter control, parameters are initialized at the start of execution and their values are allowed to change during the run. The type

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 271–281, 2004.

of the change is defined in [5] to be one of the following: *deterministic* (the parameter value is updated according to some deterministic rule), *adaptive* (the parameter value is updated based on some feedback taken from the population) or *self-adaptive* (the parameter is evaluated and updated by the evolutionary algorithm itself).

In this study, an adaptive mutation rate strategy that increases or decreases the mutation rate for each locus on the chromosome, based on feedback obtained from the current population is introduced. Even though using feedback from the current state of the search seems to be a useful approach, it has not been studied much within the scope of canonical genetic algorithms [12]. This approach is tested against previously published methods for mutation rate control on a chosen set of test problems. The results are seen to be promising and promote further study. The rest of this paper is organized as follows: Section 2 introduces the proposed mutation rate adaptation approach section 3 presents the experimental setup, section 4 discusses the results of the experiments, and section 5 provides a conclusion and possible directions for future work.

2 A Genetic Algorithm with Gene Based Adaptive Mutation

Mutation as an insurance policy against permanent loss of genes is considered to be the most sensitive of the required GA parameters. Determining the optimum fixed mutation rate for different types of problems requires an empirical analysis. In this paper, a Gene Based Adaptive Mutation (GBAM) method is proposed. This approach experiments with adaptive mutation rate values during the run using feedback from the population. Therefore, instead of using a fixed optimum value for a mutation rate, a range can be specified which provides more flexibility. Different from other known mutation adaptation strategies, GBAM has its own mutation rate value for each locus. An adaptive approach for adjusting mutation rates for the gene locations based on the feedback obtained by observing the relative success or failure of the individuals in the population is used. Since the mutation rates at each locus depend mainly on whether the individuals with a specific allele value for that locus is successful or not, GBAM is more suited to problems in which the representation is binary.

In GBAM, there are two different mutation rates defined for each locus: p_{m1} for those genes that are "1" and p_{m0} for those that are "0". In the reproduction phase, the appropriate mutation rate is applied based on the gene allele value. Initially all of the mutation rates are set to an initial value in the specified boundaries. Then for each generation, the mutation probabilities p_{m1} and p_{m0} for each locus are updated based on feedback taken from the relative success or failures of those individuals having a "1" or "0" at that locus. The update rule for the two mutation rate values for one gene location can be seen in Eq.1. This update rule is applied separately for each locus. The p_{mi} value for a locus corresponds to the rate of mutation, which will be applied when the gene value is i in the corresponding gene location. S_{avg} is the average fitness of the individuals with an allele "1" for the corresponding gene location. P_{avg} is the average fitness of the population, γ is the update value for the mutation rates. For a maximization problem, if the ratio of S_{avg} to P_{avg} is greater than 1 or all the genes at that locus are 1, i.e. the average fitness value of the individuals having the allele "1" for that locus is higher than those having "0", the allele value "1" for the corresponding locus is assumed to generate more successful results. Therefore, a decrease in p_{m1}, and

an increase in p_{m0} for the corresponding locus are implemented. Similarly, if the S_{avg} to P_{avg} ratio is less than 1 or all the genes at that locus are 0, then the opposite operations are implemented on the mutation rate values. In the case of a minimization problem, the operations in Eq. 1 should be exchanged. As a result of the updates at each generation, p_{mi} values are allowed to oscillate within the limits defined by lower and upper bounds. If an update causes a mutation rate to exceed the limits, it is set to the corresponding boundary value. All GBAM parameters are determined empirically.

$$p_m^+ = \begin{Bmatrix} p_{m1}^+ = p_{m1} - \gamma \;\wedge\; p_{m0}^+ = p_{m0} + \gamma, & (S_{avg} / P_{avg}) \geq 1 \vee \forall\, gene = 1 \\ p_{m1}^+ = p_{m1} + \gamma \;\wedge\; p_{m0}^+ = p_{m0} - \gamma, & (S_{avg} / P_{avg}) < 1 \vee \forall\, gene = 0 \end{Bmatrix} \quad (1)$$

As will be shown in the analysis of the experiments, GBAM allows rapid convergence. For unimodal objective functions, this rapid convergence provides a valuable refinement. However, the premature convergence problem, which may cause the program to get stuck at local optima, arises especially for the multimodal objective functions. This problem is explored in detail in [9] for self-adaptive mutations, however the results can easily be extended to adaptive mutation schemes too. One way to remedy this is to implement a mechanism to maintain sufficient diversity in the population for escaping local optima. As a result of preliminary experimentation, a method that complements (p_{mi}=1.0) the genotype of a predefined percentage of the population during the reproduction phase when the population converges, and then resets the mutation rates to their initial values is observed to generate acceptable results. This convergence control method is used in GBAM to enable the program to escape from possible local optima.

3 Experimental Design

The aim of the experiments is to show that GBAM requires fewer generations to reach the optimum as well as exploring its performance compared to other parameter control approaches found in literature, based on two different types of problems. The testing phase consists of two stages. In the first stage, GBAM is compared with a simple canonical genetic algorithm using a fixed mutation rate to determine whether the addition of the proposed adaptive mutation rate strategy causes a performance improvement by reducing the amount of fitness evaluations to reach the optimum. There is no convergence control used at this stage. The one-max problem, which is unimodal and easy for the simple genetic algorithm, is used for this stage of the tests. In the second stage of the testing phase, representative parameter control approaches for each type of change scheme (deterministic, adaptive and self-adaptive) developed for canonical genetic algorithms are chosen from literature and are compared with GBAM. Two test problems are used during this stage: 4-Peaks and the multiple knapsack problems. For both of these problems, all tested approaches are equipped with the convergence control mechanism explained in Section 2 to provide fair comparisons. The results are evaluated based on their solution quality and the number of fitness evaluations it takes each approach to find those results.

3.1 Test Problems

One-Max: The main aim of this problem is to maximize the number of 1s in a binary represented string of length L. The optimum for this function is L.

4-Peaks: The fitness function for the 4-Peaks problem where each individual consists of 100 bits is given in Eq.2 where z(x) is the number of contiguous 0s ending in Position 100, o(x) is the number of contiguous 1s starting in Position 1, and T is a threshold. The problem has two global and two local optima. As explained in [4], by increasing T, the basins of attraction surrounding the inferior local optima increase in size exponentially while the basins around the global optima decrease at the same rate. Therefore, increasing T makes it harder for a GA to escape the local optima.

$$REWARD = \begin{cases} 100+T & if \quad o(x) > T \wedge z(x) > T \\ 0 & otherwise \end{cases}$$
$$f(x) = MAX(o(x), z(x)) + REWARD \tag{2}$$

Multiple Knapsack Problem: In the 0/1 Multiple Knapsack Problem (Mkp), there are m knapsacks of capacity c_j, n objects of profit p_i. The weights for the objects are different for each knapsack. w_{ij} represents the i^{th} object's weight for the j^{th} knapsack. A feasible vector solution for this problem can be defined as a vector $\vec{x} = (x_1, x_2, ..., x_n)$ where $x_i \in \{0,1\}$, such that $\sum_{i=1}^{n} w_{ij} * x_i \leq c_j$ for j= 1,2,..m. The value "0" for an object in the vector representation means that the object is not placed in any of the knapsacks. Otherwise, the object is placed in all of the knapsacks. The main objective is to find a feasible vector with maximum profit $P(\vec{x}) = \sum_{i=1}^{n} x_i * p_i$. The feasible vector solution should satisfy the constraint that no knapsack is overfilled. A penalty value is added to the objective function to enable the feasible individuals to have more survivability. The penalized objective function of [6] defined in Eq. 3 is used in this study.

$$f(\vec{x}) = \sum_{i=1}^{n} p_i * x_i - \frac{(\max\{p_i\}+1)}{\min\{w_{ij}\}} * \max_{j=1,2..m} (\max(0, \sum_{i=1}^{n} w_{ij} * x_i - c_j)) \tag{3}$$

3.2 Parameter Control Approaches Chosen for Comparisons

There are different formulations and implementations of various parameter control techniques in literature. A representative scheme that is shown to give good performance is chosen from each category and used for the comparisons.

Deterministic Approach: Deterministic mutation rate schedule provides the mutation rate to be deterministically altered at each generation. The mutation rate decreases from a value (generally 0.5) to the optimum mutation rate (generally 1/L) without using any feedback from the population. The deterministic mutation rate schedule suggested in [10] was reported in [12] as being successful for hard combinatorial problems. Time-varying mutation rate p_t is calculated based on the

formula given in Eq. 4. In this formula, $t \in \{0,1,...,T-1\}$ denotes the generation number, and T is the maximum number of generations.

$$p_t = \left(2 + \frac{L-2}{T-1} * t\right)^{-1} \tag{4}$$

Self-Adaptive Approach: In the self-adaptive approach, the parameters are encoded into the chromosomes and undergo mutation and recombination. The idea is that better parameter values lead to better individuals and these parameter values will survive in the population since they are brought together with the surviving individuals. In [3] a self-adaptation mechanism of a single mutation rate per individual is proposed. The mutation of this mutation rate $p \in]0,1[$ gives the new mutation rate $p' \in]0,1[$ according to Eq. 5. In this equation γ is the learning rate which controls the adaptation speed and it is taken as 0.22 in [3]. An individual consists of a bit string and an individual mutation rate p. The new individual is determined through bit wise mutation of n bits using the mutated mutation rate value p'. The mutation rate is not allowed to go below 1/L. In this approach the crossover is applied only to the binary vector and has no effect on p.

$$p' = (1 + \frac{1-p}{p}.\exp(-\gamma.N(0,1)))^{-1} \tag{5}$$

Adaptive Approach: Adaptive GA proposed in [11] is a kind of individually adaptive mutation rate strategy. The probabilities of crossover and mutation are adapted depending on the fitness values of the individuals. The adaptation of the p_c and p_m allows the individuals having fitness values of over-average to maintain their genetic material, while forcing the individuals with sub-average fitness values to disrupt. Therefore, the method auto-controls the convergence situation. The method is tested with only SGA in [11]. In Adaptive GA both the mutation and the crossover rates are adapted. However, since the effects of the crossover rate adaptation are not addressed in this study, only the mutation rate adaptation strategy of the Adaptive GA is used as a comparison method. The mutation rate adaptation rule is given in Eq. 6. In this equation, f denotes the fitness value of the individual, f_{max} denotes the best fitness value of the current generation, and f_{avg} denotes the average fitness value of the current generation. In [11], the constants k_2 and the k_4 are chosen as 0.5.

$$\begin{aligned} p_m &= k_2(f_{\max} - f)/(f_{\max} - f_{avg}), \quad f \geq \bar{f} \\ p_m &= k_4 \quad f < f_{avg} \end{aligned} \tag{6}$$

4 Experimental Results

As explained in the previous section, the experiments consist of two stages. For all the tests in each stage, the program implementation for each chosen approach on each test problem is run 100 times. In this section the results of each stage will be given separately. In tables, μ denotes mean values, σ standard deviations and the 99% CI

confidence intervals. Some parameter settings are kept constant through all tests. A population consists of 250 individuals. Parent selection is done through tournament selection with tournament sizes of two. Recombination is done through two-point cross over at a fixed rate of 1.0. The new population is determined using elitism where the best individual replaces the worst individual in the next generation.

4.1 Exploring the Effects of GBAM on Number of Generations to Find Optimum

The GBAM approach is expected to reduce the number of generations to locate an optimal individual. To investigate just the effect of the proposed mutation rate adaptation approach, GBAM and a simple canonical genetic algorithm (SGA) is applied to the One-Max problem. Since the problem space is unimodal, no convergence control is implemented. Maximum number of generations for both GBAM and SGA are 500. The mutation rate for SGA is 1/L, where L is the string length. For GBAM, the initial mutation rate is 0.02 and this mutation rate value is allowed to change between a lower bound of 0.0001 and an upper bound of 0.2 with update value γ=0.001 in Eq.1. These settings are determined empirically to provide the best performance for each approach. Tests are performed for three different string lengths: L=200, L=400, L=800. Both GBAM and SGA are able to locate the optimum individuals for all tested string lengths. The statistical calculations are given in Table1. Here 99%CI shows the 99% confidence interval of the difference between the means of GBAM and SGA. Based on the results in Table-1, it can be said with 99% certainty that the true mean value for the reduction in number of generations to reach the optimum when using GBAM for the one-max problem with the chosen parameter settings lies within the given CI ranges. This result confirms the expectation that GBAM locates the optimal individual much quicker than a SGA under similar circumstances.

Table 1. Statistical calculations for number of generations to reach the optimum

	L=200		L=400		L=800	
	μ	σ	μ	σ	μ	σ
GBAM	48.09	2.56	99.28	34.27	217.33	39.23
SGA	91.08	7.33	169.65	14.04	332.19	23.18
99%CI	41.45 to 44.53		62.97 to 79.97		103.1 to 126.64	

4.2 Comparison of GBAM with Other Parameter Control Methods

The aim of this second testing stage is to compare the performance of GBAM with the different parameter control approaches explained in section 3.2. Two kinds of test problems are used during these tests: 4-Peaks (section 3.1.2) and Mkp (section 3.1.3). For 4-Peaks problem, the methods given in [4] are tested for different T values between 11 and 25 and it is seen that after the value of 19 it becomes hard for simple GAs to find the optimum. Therefore in this study T=11 is chosen to test an easy 4-

Peaks problem and T=27 a difficult one. These values are chosen to compare the effectiveness of the algorithms for different levels of difficulty. The global best fitness value is 199, while local best fitness value is 100. The number of maximum generations is selected as 3500 for this problem. As a testbed for the Mkp Weing-7 and Weish-30 datasets [13] are selected. In Weing-7, there are 2 knapsacks and 105 objects. In Weish-30, there are 5 knapsacks and 90 objects. The known optima are reported as 1095445 for Weing-7 and as 11191 for Weish-30 in [13]. The number of maximum generations is selected as 6000. Because the chosen penalty approach assigns high negative fitness values to infeasible individuals, for the Mkp instances, the mutation rate adaptation is done based on only the feasible individuals.

A convergence control mechanism is implemented for each of the algorithms except the Adaptive GA which has its own convergence control. The implemented mechanism takes the complement of the 25% of the population when 90% of all the genes are converged for all gene locations. Chromosome length (L) is equal to 100 for 4-Peaks, and to the number of objects for the Mkp instances. The initial mutation rate for SA is 2/L. In GBAM, the initial mutation rate is 1/L for all problem instances. The upper and lower bounds for the mutation rate in GBAM are 0.2 and 0.0001 respectively, with an update amount of γ=0.001. The comparison criteria for the tests are chosen as the best fitness values and the number of generations to reach the best fitness both averaged over 100 runs. The following abbreviations are used in Table-2 and Table-3: SA (Self Adaptive), Adap (Adaptive), Det (Deterministic) and GBAM.

Table 2. Statistical results of 4-Peak Problem Instances

		4-Peaks (T=11)			4-Peaks (T=27)		
BEST FITNESS		μ	σ	99%CI	μ	σ	99%CI
	SA	143.94	7.98	141.80-146.08	43.44	8.24	41.23-45.65
	Adap	128.98	46.18	116.60-141.36	95.53	3.53	94.58-96.48
	Det	198.83	0.64	198.66-199.00	99.82	0.88	99.67-99.97
	GBAM	199.00	0.00	199.00-199.00	199.00	0.00	199.00-199.00
NO OF GENS		4-Peaks (T=11)			4-Peaks (T=27)		
		μ	σ	99%CI	μ	σ	99%CI
	SA	2481.02	801.55	2266.19-2695.85	2302.51	852.95	2073.90-2531.12
	Adap	3020.88	404.14	2912.57-3129.20	2962.98	403.61	2854.81-3071.15
	Det	3142.52	227.71	3081.49-3203.55	3099.12	293.37	3020.49-3177.75
	GBAM	301.85	209.47	245.71-357.99	654.07	466.40	529.07-779.07

Table-2 and Table-3 are given in two parts: one is for the best fitness values and the other is for the number of generations to locate the individual with the best fitness. To assess the success of an approach, both parts should be considered together. Based on the results in these tables, GBAM seems to be promising for all of the tested problems. For the 4-Peaks problem, GBAM finds the global optimum for all of the T values, while none of the other methods can. The confidence intervals also do not

Table 3. Statistical results of MKP Problem Instances

		MKP (Weing-7)			MKP (Weish-30)		
		μ	σ	99%CI	μ	σ	99%CI
BEST FITNESS	SA	1051736.88	15499.66	1047582.69-1055891.00	10030.18	381.97	9927.81-10132.55
	Adap	1083460.50	6146.87	1081813.00-1085108.00	10860.07	144.43	10821.36-10898.78
	Det	1094818.63	462.55	1094694.63-1094942.63	11163.94	16.18	11159.60-11168.28
	GBAM	1095085.25	604.02	1094923.38-1095247.13	11190.96	0.40	11190.85-11191.07
		MKP (Weing-7)			MKP (Weish-30)		
		μ	σ	99%CI	μ	σ	99%CI
NO OF GENS	SA	3520.13	1678.82	3070.18-3970.08	3527.80	1746.44	3059.73-3995.87
	Adap	5040.67	858.49	4810.58-5270.76	4739.13	1100.37	4444.21-5034.05
	Det	5484.20	430.95	5368.70-5599.70	5461.25	521.09	5321.59-5600.91
	GBAM	2586.80	1788.95	2107.33-3066.27	588.86	695.30	402.51-775.21

intersect, showing that in 99% of all trials, the mean values for all approaches will fall within these intervals, confirming that GBAM performs better in these cases. For Mkp, GBAM generates most successful results of all, both in the average fitness value and the number of generations needed to find the best fitness. The 99% confidence intervals for the average fitness values for the Weing-7 instance intersect for Det and GBAM, however it should be noted that the number of generations for GBAM to reach its best fitness value is much less than that for Det.

The best fitness values averaged over 100 runs for all generations can be seen in Fig 1 for the tested problems. The deterministic approach behaves as expected with increasing best fitness values for each generation. The convergence intervention applied to this algorithm is not effective due to the nature of the approach since convergence occurs towards the end of the run. However in GBAM convergence can occur in earlier steps, because this method forces the population to converge fast. In GBAM after the population converges and 25% of the population is complemented, the mutation rate values for the genes are again reset to the initial values. However, due to elitism, previously found good individuals are not lost. In GBAM the overall best individual fitness values are highest of all approaches, and they are found in earlier generations as mentioned before. The best fitness values found for the Mkp which are higher than 1095000 for Weing-7 (listed as succesful in [12]) and 11150 for Weish-30 can be seen in Table-4. Based on the graphical results in Fig. 1, the rise of the plot for GBAM is fastest of all the tested methods.

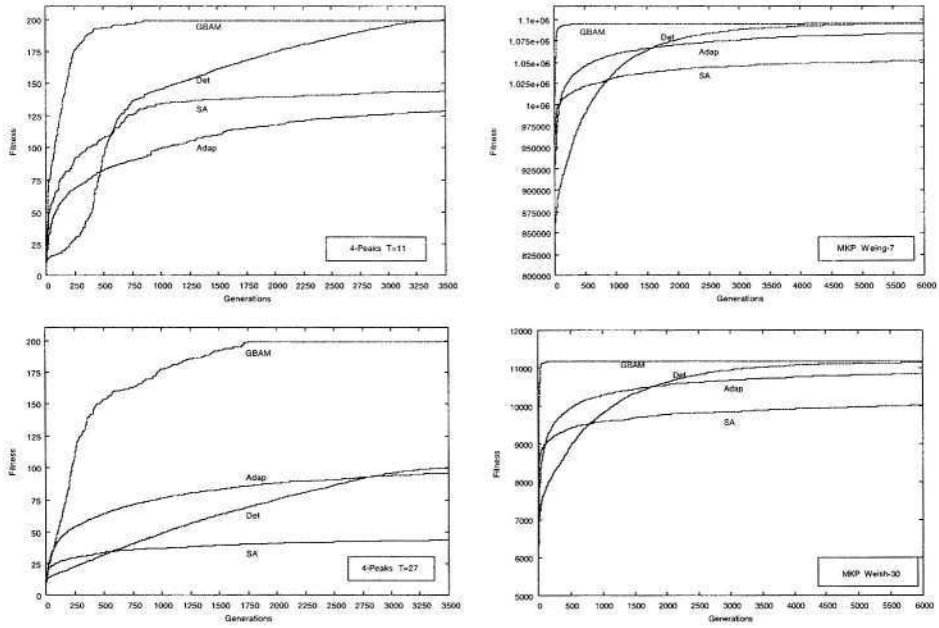


Fig. 1. Avg. best fitnesses over 100 runs

5 Conclusion and Future Work

In this study, a mutation rate adaptation approach (GBAM) for each gene location in a binary representation, based on the relative performance of individuals is proposed. GBAM is expected to increase the convergence rate. This has been confirmed through tests performed on a simple one-max problem. However, especially in multimodal problems, fast convergence may cause the population to get stuck on local optima. To overcome this problem, a convergence control mechanism that introduces diversity when the population converges, is added. This convergence control is applied to all tested algorithms but since GBAM again adapts the mutation rate parameter, it quickly converges after the introduction of diversity and thus is able to locate good solutions in fewer generations even for multimodal problems. This fact has been shown through tests using two instances of both the multiple knapsack problem and a problem with a fitness landscape that has local optima with large basins of attraction. In both cases GBAM is shown statistically to perform better than the other tested methods both in locating the optima and also in reaching the optima in fewer generations. One major drawback is that GBAM increases the computational costs since at every generation new mutation rate values for all gene locations (as many as the length of a chromosome) are recalculated. On the other hand, among the other tested methods, the deterministic approach seems to be able to reach acceptable fitness values with almost no additional computational costs, however it takes much

Table 4. Best fitness values higher than selected thresholds observed in 100 runs for MKP (1095000 for Weish-30 and 11150 for Weing-7)

Weing-7 Results				
	SA	Adap	DET	GBAM
1095445	-	-	-	10
1095382	-	-	5	18
1095357	-	-	3	12
1095295	-	-	-	1
1095266	-	-	-	6
1095264	-	-	3	3
1095262	-	-	-	2
1095232	-	-	1	1
1095207	-	-	2	5
1095206	-	-	4	2
1095202	-	-	1	-
1095195	-	-	1	-
1095157	-	-	1	3
1095141	-	-	1	1
1095137	-	-	3	4
1095132	-	-	-	1
1095114	-	-	1	1
1095112	-	-	3	2
1095081	-	-	2	-
1095065	-	-	-	2
1095062	-	-	3	1
1095057	-	-	-	1
1095056	-	-	-	1
1095039	-	-	1	-
1095035	-	-	1	3
1095014	-	-	1	-
1095007	-	-	2	-
Overall	0/100	0/100	39/100	80/100

Weish-30 Results				
	SA	Adap	DET	GBAM
11191	-	-	11	99
11187	-	-	5	1
11182	-	-	2	-
11181	-	-	1	-
11178	-	-	1	-
11177	-	-	1	-
11175	-	-	1	-
11174	-	-	2	-
11173	-	-	4	-
11172	-	-	2	-
11170	-	-	4	-
11169	-	-	4	-
11168	-	-	3	-
11167	-	-	5	-
11166	-	-	1	-
11165	-	-	1	-
11164	-	-	1	-
11162	-	-	3	-
11161	-	-	2	-
11160	-	-	5	-
11159	-	-	1	-
11158	-	-	3	-
11157	-	-	5	-
11156	-	-	1	-
11155	-	-	1	-
11154	-	-	1	-
11153	-	-	6	-
11152	-	1	-	-
11151	-	-	5	-
Overall	0/100	1/100	82/100	100/100

longer. So for problems similar to the ones used in this study, in the cases where it is more important to find good results in fewer generations, GBAM seems to be the better choice among the tested methods.

Even though as a result of these preliminary tests, the overall performance of GBAM seems to be very promising for the chosen type of problems, there is still more work to be done to be able to make healthy generalizations. First of all, the parameter settings for GBAM, such as the lower and upper bound values, initial mutation rate and the mutation update value have been determined experimentally. More experiments need to be performed to see the effects of these parameters on performance more thoroughly. Secondly, the test problem set can be extended to include different types of problem domains. Thirdly, a convergence control

mechanism has been implemented for this study, however its effects have not been thoroughly examined. More rigorous experimentation needs to be performed to be able to fully understand the exact nature of the convergence control method. Finally, it is observed that the adaptive nature of the proposed mutation mechanism might be suitable to be used in dynamic environments. Since the mutation rate adapts itself based on the relative fitness of individuals, it would also be able to adapt to a change in the environment causing a change in the fitness values of the individuals and it would not need to explicitly detect the change in the environment. Based on these observations, it seems to be a promising line of further study to use GBAM in dynamic environments in its pure form or even in combination with other approaches developed previously to cope with changing environments.

Acknowledgement. The authors would like to thank Jens Gottlieb for his helpful suggestions regarding the penalty function for MKP.

References

1. Angeline P. J.: Adaptive and Self-adaptive Evolutionary Computation. Computational Intelligence, A Dynamic System Perspective, IEEE (1995) 152-161
2. Bäck T.: Optimal Mutation Rates in Genetic Search. Proc. of 5th International Conference on Genetic Algorithms, Morgan Kaufmann (1993)
3. Bäck T., Schlütz M.: Intelligent Mutation Rate Control in Canonical Genetic Algorithms. Proc. Int. Symp. on Methodologies for Intelligent Syst. (1996) 158-167
4. Baluja S., Caruana. R.: Removing the Genetics from the Standard Genetic Algorithm. Proc. 12. Int. Conf. on Machine Learning, Morgan Kaufmann, (1995) 38-46
5. Eiben A. E., Hinterding R., Michalewicz Z.: Parameter Control in Evolutionary Algorithms. IEEE Trans. on Evolutionary Computation, Vol. 3, No.2. (1999) 124-141
6. Gottlieb J.: On the Feasibility Problem of Penalty-Based Evolutionary Algorithms for Knapsack Problems. Proc. of EvoWorkshops, Lecture Notes in Computer Science Vol. 2037, Springer (2001) 50-59
7. Hinterding R., Gielewski H., Peachey T. C.: The Nature of Mutation in Genetic Algorithms. Proc. 6. Int. Conf. on GAs, Morgan Kaufmann (1995) 65-72
8. Ochoa G.: Setting the Mutation Rate: Scope and Limitations of the 1/L Heuristic. Proc. Genetic and Evolutionary Comp. Conf., Morgan Kaufmann (2002)
9. Rudolph G.: Self-Adaptive Mutations May Lead to Premature Convergence. IEEE Trans. on Evolutionary Computation, Vol. 5., No. 4. (2001) 410-414
10. Smith J. E., Fogarty T. C.: Operator and Parameter Adaptation in Genetic Algorithms. Soft Computing 1, Springer-Verlag (1997) 81-87
11. Srinivas, M., Patnaik, L. M.: Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms. IEEE Trans. on Systems, Man and Cybernetics, Vol. 24. No. 4. (1994) 656-667
12. Thierens D.: Adaptive Mutation Control Schemes in Genetic Algorithms. Proc. of Congress on Evolutionary Computing, IEEE (2002)
13. weing7.dat, weish30.dat: http://elib.zib.de/pub/Packages/mp-testdata/ip/sac94-suite/

Subthreshold-Seeking Behavior and Robust Local Search

Darrell Whitley[1], Keith Bush[1], and Jonathan Rowe[2]

[1] Computer Science, Colorado State University, Fort Collins, CO 80523
[2] Computer Science, University of Birmingham, Birmingham B15 2TT, UK

Abstract. Subthreshold-seeking behavior occurs when the majority of the points that an algorithm samples have an evaluation less than some target threshold. We characterize sets of functions where subthreshold-seeking behavior is possible. Analysis shows that subthreshold-seeking behavior, when possible, can be increased when higher bit precision is used with a bit climber search algorithm and a Gray code representation. However, higher precision also can reduce exploration. A simple modification to a bit-climber can improve its subthreshold-seeking behavior. Experiments show that this modification results in both improved search efficiency and effectiveness on common benchmark problems.

1 Introduction and Background

The goal of an optimization algorithm is to find optimal points of a search space. However, it may sometimes be useful to try to locate points that are sufficiently good (e.g., within some threshold). We might also like to have some assurance that an algorithm is relatively effective on a wide range of problems.

We will say that a search algorithm is *robust* if it is able to beat random search across a wide variety of optimization problems. Christensen and Oppacher [1] have shown that the No Free Lunch theorem does not hold over broad classes of problems that can be described using polynomials of a single variable of bounded complexity. The algorithm that Christensen and Oppacher propose is robust in as much as it is able to out-perform random enumeration on a general class of problems. But the algorithm they propose is not very effective as a search algorithm. Can we do better than this? And what theoretical and practical implications does this question imply?

In this paper we generalize the approach of Christensen and Oppacher. We will say that an algorithm has *subthreshold-seeking behavior* if the algorithm establishes an aspiration threshold, and then spends more than half of its time sampling points that are below threshold. An algorithm with subthreshold-seeking behavior can beat random enumeration and side-step the No Free Lunch result by focusing on a special, but nevertheless general class of functions.

We next ask to what degree does a local search bit climber display robust, subthreshold-seeking behavior. We show that there are a number of conditions where a local search bit climber will display subthreshold-seeking behavior.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 282–293, 2004.

We also present several theorems that indicate how and when a local search bit climber can display subthreshold-seeking behavior. These results prove that subthreshold-seeking behavior increases when higher bit precision encodings are used. In addition to the theoretical analysis presented, we empirically show that a local search bit climber with sufficient precision will spend most of its time "subthreshold" on a number of common benchmark problems. We then make a very simple modification to a bit climber with restarts algorithm to allow it to spend more time "subthreshold." The modified algorithm is both more efficient and effective, finding better solutions faster than local search with restarts.

1.1 The SubMedian-Seeker

Suppose we have an objective function $f : [a, b] \rightarrow R$, where $[a, b]$ is a closed interval. We discretize this interval by taking N uniformly sampled points, which we label with the set $\mathcal{X} = 0, 1, ..., N - 1$. By abuse of notation, we will consider $f : \mathcal{X} \rightarrow R$, such that $f(x)$ takes on the evaluation of the point labeled x. Assume f is bijective as a function of $\mathcal{X}$ and that the median value of f is known and denoted by $med(f)$.

Christensen and Oppacher define a minimization algorithm called *SubMedian-Seeker.* The algorithm presented here is a similar to SubMedian-Seeker but is simpler and easier to understand.[1]

`EZ-SubMedian-Seeker`

1. If less than $\frac{|\mathcal{X}|}{2}$ points have been sampled, then choose a random sample point, $x \in \mathcal{X}$. Otherwise terminate.
2. While $f(x) < med(f)$ pick next sample $x = x + 1$. Else goto step 1.

Without lose of generality, we assume that x and its successor $x + 1$ are integers. The algorithm exploits the fact that for certain classes of functions, points that are adjacent to submedian points are more often than not also submedian points. The actual performance depends on $M(f)$, which measures the number of submedian values of f that have *successors* with supermedian values. Let M_{crit} be a critical value relative to $M(f)$ such that when $M(f) < M_{crit}$ SubMedian-Seeker (or EZ-SubMedian-Seeker) is better than random search.

Christensen and Oppacher [1] then prove:

> *If f is a uniformly sampled polynomial of degree at most k and if $M_{crit} > k/2$ then SubMedian-Seeker beats random search.*

[1] The original submedian seeker is able to detect and exploit functions where every other point is below submedian and thus a local optimum. EZ-SubMedian-Seeker will not detect this regularity. However, we generally are not concerned with functions that are maximally multimodal. Also, the Christensen and Oppacher proof still holds for EZ-SubMedian-Seeker.

This result holds because there are at most k solutions to $f(x) = y$ where y can be any particular co-domain value. If y is a threshold, then there are at most k crossings of this threshold over the sampled interval. Half, or $k/2$, of these are crossings from subthreshold to superthreshold values. Thus, $M(f) <= k/2$ for polynomials of degree k. In the case where the median is the threshold, step 1 has equal probability of sampling either a submedian or supermedian value. Therefore, as long as step 2 generates a surplus of submedian points before terminating at a supermedian point, the algorithm beats random search. We can think of step 1 as an exploration phase with balanced cost and step 2 as an exploitation phase that accumulates submedian points. If $M(f) \leq k/2 < M_{crit}$, then SubMedian-Seeker (and EZ-SubMedian-Seeker) will perform better than random enumeration because more time is spent below threshold during step 2. Christensen and Oppacher offer extensions of the proof for certain multivariate polynomials as well. In the next section we characterize a more general Subthreshold-Seeker algorithm.

2 Subthreshold-Seeker

We still assume f is 1-dimensional and bijective and $N = |\mathcal{X}|$. Set a threshold of α between 0 and 1/2. We are interested in spending time in the αN best points of the search space (ordered by f). We refer to these as *subthreshold* points. Addition is *modulo N* and the search space is assumed to wrap around so that points 0 and $N - 1$ are neighbors.

Let $\Theta(f)$ denote a threshold co-domain value such that exactly αN points of $\mathcal{X}$ have evaluations, $f(x)$, less than $\Theta(f)$. Subthreshold-Seeker works as follows:

1. Pick a random element $x \in \mathcal{X}$ that has not been seen before.
2. If $f(x) < \Theta(f)$ let $x = x + 1$ and $y = x - 1$; otherwise goto 1.
3. While $f(x) < \Theta(f)$ pick next sample $x = x + 1$.
4. While $f(y) < \Theta(f)$ pick next sample $y = y - 1$.
5. If Stopping-Condition not true, goto 1.

Once a subthreshold region has been found, this algorithm searches left and right for subthreshold neighbors. This minor variation on Submedian-Seeker means that a well defined region has been fully exploited. This is critical to our quantification of this process. We will address the "Stopping-Condition" later.

For theoretical purposes, we assume that the function $\Theta(f)$ is provided. In practice, we can select $\Theta(f)$ based on an empirical sample.

2.1 Functions with Uniform Quasi-basins

We will define a *quasi-basin* as a set of contiguous points that are below a threshold value. Note this is different from the usual definition of a basin: a quasi-basin may contain multiple local optima. Consider a function f where

all subthreshold points are contained in B equally sized quasi-basins of size $\alpha N/B$. We then ask how many superthreshold points are visited before all the subthreshold points are found. Suppose k quasi-basins already have been found and explored. Then there remains $B-k$ quasi-basins to find, each containing $\alpha N/B$ points. There are at most $N - k\alpha N/B$ superthreshold points unvisited. So the probability of hitting a new quasi-basin is (slightly better than)

$$\frac{(B-k)(\alpha N/B)}{N - k\alpha N/B} = \frac{(B-k)\alpha}{B - k\alpha}$$

This calculation is approximate because it assumes that superthreshold points are sampled with replacement. As long as the probability of randomly sampling the same superthreshold point twice is extremely small, the approximation will be accurate. For large search spaces this approximation should be good.

If the probability of "hitting" a quasi-basin is p, the expected number of trials until a "hit" occurs is $1/p$. This implies that the expected number of misses before a successful hit occurs is $1/p-1$. So the expected number of superthreshold points that are sampled before finding a new quasi-basin is approximately (slightly less than)

$$\frac{B - k\alpha}{(B-k)\alpha} - 1 = \frac{B(1-\alpha)}{(B-k)\alpha}$$

This means that the expected number of superthreshold points seen before the algorithm has found all quasi-basins is bounded above by

$$\sum_{k=1}^{B-1} \frac{B(1-\alpha)}{(B-k)\alpha} = \frac{B(1-\alpha)}{\alpha} \sum_{k=1}^{B-1} \frac{1}{(B-k)} = \frac{B(1-\alpha)}{\alpha} H(B-1) \qquad (1)$$

where H is the harmonic function. Note $H(B-1)$ is approximated by $(\log(B-1))$. Throughout this paper log denotes $\log_2$.

2.2 Functions with Unevenly Sized Quasi-basins

Now suppose that the quasi-basins are not evenly sized. If f is a uniformly sampled polynomial of degree at most k it can have at most $k/2$ quasi-basins and after fixing $\mathcal{X}$, there must be at least 1 subthreshold quasi-basin of size $\alpha N/(k/2)$ or larger. One way Subthreshold-Seeker can be used is to search until one quasi-basin of size at least $\alpha N/(k/2)$ is found. The waiting time to find such a basin is less than $\frac{N}{\alpha N/(k/2)} = \frac{k/2}{\alpha}$. It can be shown that more subthreshold points will be sampled than superthreshold points as long as

$$\alpha\frac{k/2}{\alpha} + \alpha N/(k/2) > (1-\alpha)\frac{k/2}{\alpha} \quad \text{which reduces to} \quad 2\alpha + \frac{\alpha^2 N}{(k/2)^2} > 1$$

and this does not even count smaller subthreshold quasi-basins that are exploited before finding one of size $\alpha N/(k/2)$.

What if we want to find all quasi-basins that contain at least M points, and we suppose there are B such quasi-basins. (If we happen to find some smaller ones, that is a bonus). Suppose we have found k such quasi-basins. The probability of finding another is

$$> \frac{(B-k)M}{(B-k)M + N - BM} = \frac{(B-k)M}{N - kM}$$

So the expected number of superthreshold points visited before this happens is

$$< \frac{N-kM}{(b-k)M} - 1 = \frac{N-BM}{BM-kM}$$

The total number of superthreshold points visited is thus

$$< \sum_{k=1}^{B-1} \frac{N-BM}{BM-kM} = \left(\frac{N-BM}{M}\right) H(B-1)$$

Theorem 1: *Let α define a threshold presenting some fraction of the search space. Suppose there are B quasi-basins each containing at least M points. If $M > \sqrt{\frac{NH(B-1)}{B}}$ then Subthreshold Seeker can find all B basins and will explore more subthreshold points that superthreshold points. For all $\alpha < 1/2$ Subthreshold Seeker beats random search.*

Proof:

$$\text{If } M^2 > \frac{NH(B-1)}{B} \text{ then } BM > \frac{NH(B-1)}{M} > \left(\frac{N}{M} - B\right) H(B-1).$$

BM is the total number of subthreshold points visited. The expected number of superthreshold points visited is given by $\left(\frac{N}{M} - B\right) H(B-1)$. Therefore, for sufficiently large N, when $M > \sqrt{\frac{NH(B-1)}{B}}$ more subthreshold points are visited than superthreshold points. □

Given information (or strong assumptions) about M and B we can restrict α and spend more time exploring the best regions of the search space compared to SubMedian-Seeker.

Table 1 calculates $M = \sqrt{\frac{NH(B-1)}{B}}$ rounded up to the nearest integer. It also computes α. This value is exact when $\alpha > BM/N$; otherwise it underestimates the percentage of the space that is below threshold. Clearly, as the number of quasi-basins goes up, fewer points occur in each quasi-basin. As the search space size increases, there are more points in each quasi-basin, but we can also lower α so that the number of subthreshold points becomes a smaller percent of the search space. The smaller the subthreshold value, the more Subthreshold-Seeker will beat random-search, and the more effective Subthreshold-Seeker becomes as a general search algorithm.

This perspective also provides another insight. Note that we can interpret the change in the size of the search space as a change in precision: the number of quasi-basins generally does not change (for polynomials the number of

Table 1. This table computes how large quasi-basins must be in order for more subthreshold points to be sampled than superthreshold points when there are different numbers of quasi-basins (B) and for different size search spaces (N). The threshold α is also given, assuming exactly B quasi-basins all of size M.

	B = Number of Quasi-Basins							
Size = N	2^5	2^7	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}
10^6	396	234	99	74	55	40	30	22
10^7	1250	740	313	232	172	126	93	68
10^8	3953	2339	988	733	542	399	293	214
10^9	12500	7395	3123	2317	1711	1260	924	676

Sizes of quasi-basins, M, as a function of B and N.

Size=N	2^5	2^7	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}
10^6	0.013	0.030	0.10	0.150	0.222	0.326	0.479	0.701
10^7	0.004	0.009	0.032	0.047	0.070	0.103	0.151	0.222
10^8	0.001	0.003	0.010	0.015	0.022	0.033	0.048	0.070
10^9	0.0004	0.001	0.003	0.005	0.007	0.010	0.015	0.022

Corresponding values of α

quasi-basins is bounded by $k/2$), but we sample more densely, thus effectively increasing the number of points in the search space. Higher precision allows the search to spend more time subthreshold (or to use a lower threshold). But if the precision is too high, search provides little in the way of exploration when making subthreshold moves.

The subthreshold algorithm, like the original SubMedian-Seeker, isn't really an efficient search algorithm. The goal of search usually is not to examine as many sub-threshold points as possible. However, understanding how such algorithms beat random enumeration can provide practical insights.

Two observations come out of this work. One observation is that precision matters. For any threshold, the relative proportion of points that are subthreshold does not change with a change in precision. However, the number of points that fall within a quasi-basin increases with precision. Assuming a change in precision does not change the boundaries of the quasi-basins, algorithms with subthreshold-seeking behavior will spend more time subthreshold at higher precision. The second observation is that sampling can potentially be used to establish threshold values that can be used to focus the search. In the remainder of the paper, we explore both of these ideas in conjunction with simple, but practical, local search methods.

3 Precision and Subthreshold Local Search

We first look at how precision affects the number of neighbors that exist within a certain distance from some reference point under Gray code.

Lemma 1: *Given a 1-D function of size $N = 2^L$ and a reference point R in the function, under a Gray encoding at most $\lceil \log(Q) \rceil$ bits encode for points that are more than a distance of D points away from R, where $D = \frac{1}{Q}N$.*

Proof: In the 1-dimensional case when the highest order bit is changed under Gray encoding this accesses the only neighbor that is in the opposite half of the search space. (This does not imply that the neighbor is necessarily far away.)

Bits are eliminated to remove the remaining half of the search space which does not contain the reference point. We continue to reduce the search space around the reference point by removing bits until $\lceil \log(Q) \rceil$ bits have been eliminated. The remaining search space is then at most $D = N/Q$ points since

$$\log(N/Q) + \log(Q) = \log(N) \quad \text{and} \quad N(1/2)^{\lceil \log(Q) \rceil} \leq N/Q \qquad \square.$$

As precision increases, the quantity N/Q becomes larger and thus $\log(N/Q)$ increases. However Q and $\log(Q)$ remain constant. Thus, at higher precision, the number of neighbors within a distance of N/Q points increases.

Now assume we are trying to isolate a quasi-basin of size N/Q points.

Theorem 2: *Given a quasi-basin that spans $1/Q$ of a 1-D function of size $N = 2^L$ and and a reference point R inside the quasi-basin, the expected number of neighbors of R that fall inside the quasi-basin under a reflected Gray code is greater than $\lfloor (\log(N/Q)) \rfloor - 1$.*

Sketch of Key Ideas:

The full proof for this theorem is long and is available in an expanded version of the current paper (see www.cs.colostate.edu/ genitor/Pubs.html). The proof averages over all possible points in the quasi-basin and all possible placements of reflections points; this accounts for all possible neighborhood configurations that can exist within a quasi-basin.

We define the lower triangle matrix M^x using a recursive definition such that $M^1 = [1]$. Matrix M^x can be decomposed into a 2^{x-1} by 2^{x-1} square matrix whose elements are all the integer x, plus 2 identical lower triangle matrices M^{x-1}. The square matrix occupies the first 2^{x-1} columns of the last 2^{x-1} rows of M^x. The first $2^{x-1} - 1$ rows of M^x correspond to the recursively defined matrix M^{x-1}. Finally, another copy of M^{x-1} is appended to the last $2^{x-1} - 1$ rows of the square matrix.

The following illustration represents a quasi-basin over 7 points (left). Each row represents the number of neighbors for each point, such that those neighbors falls inside the quasi-basin under a Gray code, where the main reflection point of the Gray code is at the location marked by the | symbol. The lower triangle matrix on the right is the matrix M^3 for a search space of size $2^3 - 1 = 7$.

```
2 3 2 3 3 3 2 |
  2 2 3 3 2 3 | 1                    1
    1 3 2 3 3 | 2 2                  2 2
      2 3 3 3 | 2 3 2                2 2 1
        2 3 2 | 3 3 3 2              3 3 3 3
```

```
2 2 | 3 3 2 3 1          3 3 3 3 1
  1 | 3 2 3 3 2 2        3 3 3 3 2 2
    | 2 3 3 3 2 3 2      3 3 3 3 2 2 1
```

Let $F(x)$ compute the average value over the elements of matrix M^x. By induction one can show:

$$x - 1 < F(x) = \frac{2^x x}{2^x - 1} - 1 < x$$

The full proof shows via two constructive subproofs that as the size of the quasi-basin grows from $2^x - 1$ points to $2^{x+1} - 2$ points the expected number of neighbors is always greater than $F(x) > x - 1$ where $x = \lfloor(log(N/Q))\rfloor$.

Corollary: *Given a quasi-basin that spans* $1/Q$ *of the search space and a reference point R that falls in the quasi-basin, the majority of the neighbors of R under a reflected Gray code representation of a search space of size N will be subthreshold in expectation when* $\lfloor(log(N/Q))\rfloor - 1 > \log(Q) + 1$.

A local search algorithm currently at a subthreshold point can only move to an equal or better point which must also be subthreshold. And as precision increases, the number of subthreshold neighbors also increases, since $\lfloor(log(N/Q))\rfloor - 1$ increases while Q remains constant. This assumes the quasibasin is not divided by increasing the precision. The above analysis would need to hold for each dimension of a multidimensional search space, but these results suggest there are very general conditions where a bit-climbing algorithm using a Gray code representation can display subthreshold-seeking behavior. This also assumes the search algorithm can absorb the start-up costs of locating a subthreshold starting point.

4 Algorithms

Under favorable conditions a bit-climbing algorithm using a Gray code representation can display subthreshold seeking behavior, but do they display subthreshold seeking behavior on common benchmarks? In this section, we compared two versions of bit-climbers. Both algorithms use steepest ascent Local Search (LS) which evaluates all neighbors before moving. One algorithm, denoted LS-Rand, uses random restarts. Another algorithm, denoted LS-SubT, uses sampling to start search the bit climbing process at a subthreshold point.

LS-SubT first samples 1000 random points, and then climbs from the 100 best of these points. In this way, LS-SubT estimates a threshold value and attempts to stay in the best 10 percent of the search space.

LS-Rand does $100+y$ random restarts. LS-Rand was given y additional random starts to compensate for the 1000 sample evaluations used by the LS-SubT algorithm. To calculate y we looked at the size of the bit encoding and the average number of moves needed to reach a local optimum.

4.1 Experiments and Results

Both LS-Rand and LS-SubT were tested on benchmarks taken from Whitley et al. [2] who also provide function definitions. The test function included Rastrigin (F6) and Schwefel (F7) which are both separable. The other functions include Rosenbrock (De Jong's F2), F101 and Rana functions as well as a *spike* function similar to one defined by Ackley [3] where:

$$F(x,y) = -20e^{-0.2\sqrt{(x^2+y^2)/2}} - e^{(cos2\pi x + cos2\pi y)/2} + 22.7, \quad x_i \in [-32.768, 32.768]$$

All problems were posed as 2-dimensional search problems. Experiments were performed at 10 and 20 bits of resolution per parameter. A *descent* corresponds to one iteration of local search, which will locate one local optimum. A *trial* corresponds to one run of the respective algorithm, composed of 100 descents for LS-SubT and $100 + y$ descents for LS-Rand. An *experiment* corresponds to 30 trials. Each experiment is a configuration of search algorithm, test function and parameter resolution. Statistics are computed over each experiment. All chromosomes were encoded using standard Gray code.

Table 2. Results of steepest ascent search at 10 bit resolution per parameter in 2-dimensional space. LS-Rand (here Rand) used 104 restarts. LS-SubT (here SubT) restarted from best 100 of 1000 random points. 0.0* denotes a value less than 1×10^{-13}. Evals were rounded to the nearest integer. The † denotes a statistically significant difference at the 0.05 level using a t-Test.

Function	ALG	Mean	Std	Best	Std	Sub	Evals	Std
ackley	Rand	2.72	0.71	0.18	0.0*	62.4	19371	663
	SubT	0.79†	0.32	0.18	0.0*	79.7	16214†	163
f101	Rand	-29.2	0.0*	-29.2	0.0*	71.7	22917	288
	SubT	-29.2	0.0*	-29.2	0.0*	84.0	18540†	456
f2	Rand	0.10	0.01	0.001	0.002	61.4	23504	3052
	SubT	0.10	0.01	0.0004	0.0*	72.0	666†	1398
griewangk	Rand	0.86	0.16	0.010	0.011	59.5	13412	370
	SubT	0.75†	0.11	0.005	0.009	80.1	9692†	125
rana	Rand	-37.8	0.84	-49.65	0.59	49.5	22575	2296
	SubT	-39.7†	0.68	-49.49	0.52	57.6	19453†	1288
rastrigin	Rand	4.05	0.20	0.100	0.30	63.5	18770	495
	SubT	4.00	0.28	0.0	—	75.4	14442†	343
schwefel	Rand	-615.8	11.8	-837.9	0.0*	53.5	17796	318
	SubT	-648.0†	10.1	-837.9	0.0*	68.0	14580†	414

The results of 10 and 20 bit resolution experiments are given in Tables 2 and 3 respectively. *Mean* denotes mean solution over all descents in all trials. (This is also the mean over all local optima found.) *Best* denotes the *best solution per trial* (i.e., the best optimum found over 100 or $100 + y$ descents). *Sub* denotes the percentage of all evaluations that were subthreshold. *Evals* denotes the

Table 3. Results of steepest ascent search at 20 bit resolution per parameter in 2-dimensional space. LS-Rand (here Rand) used 101 restarts. LS-SubT (here SubT) restarted from best 100 of 1000 random points. 0.0* denotes a value less than 1×10^{-7}. Evals were rounded to the nearest integer. The † denotes a statistically significant difference at the 0.05 level using a t-Test.

Function	ALG	Mean	Std	Best	Std	Sub	Evals	Std
ackley	Rand	2.84	0.66	0.0001	0.0*	75.1	77835	1662
	SubT	0.65†	0.28	0.0001	0.0*	89.9	73212†	1194
f101	Rand	-29.2	0.0*	-29.22	0.0*	84.7	84740	1084
	SubT	-29.2	0.0*	-29.22	0.0*	92.3	77244†	1082
f2	Rand	0.0*	0.0*	0.0*	0.0*	86.0	2×10^7	4×10^5
	SubT	0.0*	0.0*	0.0*	0.0*	85.9	2×10^7	3×10^5
griewangk	Rand	0.75	0.20	0.0045	0.003	80.3	66609	1109
	SubT	0.60†	0.09	0.0049	0.003	90.0	59935†	1103
rana	Rand	-40.63	0.93	-49.76	0.47	74.2	3×10^6	8×10^5
	SubT	-42.54†	0.66	-49.83	0.51	85.0	3×10^6	8×10^5
rastrigin	Rand	4.10	0.22	0.033	0.18	81.5	76335	1734
	SubT	3.94†	0.21	0	—	88.5	68019†	1018
schwefel	Rand	-622.7	13.8	-837.97	0.0*	73.5	75285	969
	SubT	-660.4†	13.4	-837.97	0.0*	84.8	69372†	1340

mean number of test function evaluations per trial averaged over all trials in the experiment. *Stddev* denotes the standard deviation of the value given in the adjacent left-hand column.

In general, the results indicate that LS-SubT sometimes produces statistically significant better solution quality compared to LS-Rand. LS-SubT never produces statistically significant worse performance than LS-Rand.

The data suggests two observations about subthreshold-seeking behavior. First, the sampling used by LS-SubT results in a higher proportion of subthreshold points compared to LS-Rand as shown in Tables 2 and 3. Second, a larger proportion of subthreshold neighbors are sampled for searches using higher precision. At 20 bits of precision per parameter, at least 70 percent of the points sampled by LS-Rand were subthreshold, and at least 80 percent of the points samples by LS-SubT were subthreshold. At 10 bits of precision per parameter, LS-SubT sampled subthreshold points 57 to 84 percent of the time.

At 10 bits of precision, LS-SubT also did fewer evaluations, meaning that it reached local optima faster than LS-Rand. This makes sense in as much as it starts at points with better evaluations. Sometimes the difference was dramatic. Thus, the majority of the time LS-SubT also produced solutions as good or better than LS-Rand, and it did so with less effort.

At 20 bits of precision, there is less difference between LS-Rand and LS-SubT. This follows from our theory, since higher precision implies that both algorithms spend more time subthreshold after a subthreshold point is found, but this does not necessarily result in faster search.

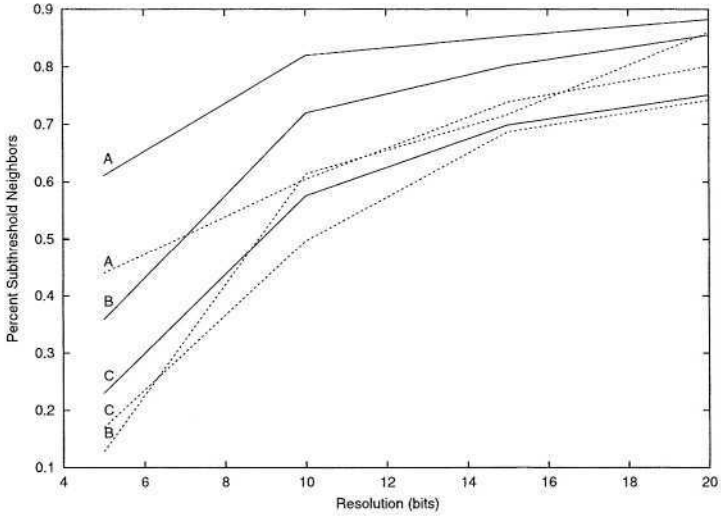


Fig. 1. The graph shows the number of subthreshold points for LS-SubT (the solid line) and LS-Rand (the dashed line) as a function of precision. The functions are A=Griewangk, B=F2, and C=Rana.

To further explore the impact of bit precision, we performed local search using LS-Rand and LS-SubT at 5, 10, 15 and 20 bits of resolution. The test functions used were Griewangk, F2, and Rana functions. Figure 1 shows the percentage of points that were subthreshold at different resolutions. Across all three functions, higher resolution produced a greater percentage of subthreshold sampling. At 5 bit precision less than half of all neighbors are subthreshold for LS-Rand on all functions. With too few bits, both neighborhood sampling and random sampling misses quasi-basins. But with too many bits, search is slowed down because the stepsize can become too small.

There is still much the heuristic search community does not understand about the impact of using different bit precisions. Unexpected results were encountered on two functions. The number of evaluations that were executed on the Rana and F2 functions at 20-bit resolution is huge. Examination of the search space shows that both of these functions contain "ridges" that run at almost 45 degrees relative to the (x, y) coordinates. In this context, the steepest ascent bit-climber is forced to creep along the ridge in very small, incremental steps. Higher precision exaggerates this problem, which is hardly noticeable at 10 bits of precision. This is a serious problem we are continuing to research.

5 Conclusions

The No Free Lunch theorem formalizes the idea that all blackbox search algorithms have identical behavior over the set of all possible discrete functions [4]

[5] [6]. Schumacher et al. [7] review different variants of the No Free Lunch theorem; they show that No Free Lunch holds over a finite set if and only that set is closed under permutation.

The Subthreshold-Seeker algorithm can focus search in the better regions of the search space. Conditions are outlined that allow a subthreshold seeker to beat random enumeration on problems of bounded complexity.

A simple sampling mechanism can be used to initialize local search at subthreshold points, thereby increasing the potential for subthreshold seeking behavior. Of course this strategy also has its own failure modes. Assume that an "important" basin of attraction, or a quasi-basin, is very large above threshold, yet small below threshold; then it is possible that random restarts could have an advantage over subthreshold restarts if success were measured in terms of finding and exploiting this "important" region. Of course the problem with random restarts is that the search can converge to local optima that are superthreshold.

Acknowledgments. This work was supported by National Science Foundation grant IIS-0117209.

References

[1] Christensen, S., (2001), F.O.: What can we learn from No Free Lunch? In: GECCO-01, Morgan Kaufmann (2001) 1219–1226

[2] Whitley, D., Mathias, K., Rana, S., Dzubera, J.: Evaluating Evolutionary Algorithms. Artificial Intelligence Journal **85** (1996) 1–32

[3] Ackley, D.: A Connectionist Machine for Genetic Hillclimbing. Kluwer Academic Publishers (1987)

[4] Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation **4** (1997) 67–82

[5] Radcliffe, N., Surry, P.: Fundamental limitations on search algorithms: Evolutionary computing in perspective. In van Leeuwen, J., ed.: Lecture Notes in Computer Science 1000. Springer-Verlag (1995)

[6] Culberson, J.: On the Futility of Blind Search. Evolutionary Computation **6(2)** (1998) 109–127

[7] Schumacher, C., Vose, M., Whitley, D.: The No Free Lunch and Problem Description Length. In: GECCO-01, Morgan Kaufmann (2001) 565–570

Ruffled by Ridges: How Evolutionary Algorithms Can Fail

Darrell Whitley, Monte Lunacek, and James Knight

Computer Science, Colorado State University, Fort Collins, CO 80523

Abstract. The representations currently used by local search and some evolutionary algorithms have the disadvantage that these algorithms are partially blind to "ridges" in the search space. Both heuristics search and gradient search algorithms can exhibit extremely slow convergence on functions that display ridge structures. A class of rotated representations are proposed and explored; these rotated representations can be based on Principal Components Analysis, or use the Gram-Schmidt orthogonalization method. Some algorithms, such as CMA-ES, already make use of similar rotated representations.

1 Introduction

Various genetic algorithms and local search methods are more or less blind to "ridges" in the search space of parameter optimization problems. In two dimensions, the ridge problem is essentially this: a method that searches north, south, east and west will not see improving moves that are oriented at a 45 degree angle in the search space.

The *ridge problem* is relatively well documented in the mathematical literature on derivative free minimization algorithms [1,2]. However, there is little discussion of this problem in the heuristic search literature. Our exploration of the "ridge problem" was motivated by three concerns.

First, over the last few years experiments have shown that genetic algorithms are more sensitive to local optima induced by different bit representations than was previously believed. Until recently, much of this work has focused on how representations such as Gray codes destroy local optima [3]. Our work focuses on when Gray codes *create* new optima: this happens only along *ridges.*

Second, some algorithms have semi-random behaviors that don't seem to make any sense from either a theoretical or intuitive perspective, and yet they work relatively well on certain benchmarks. What we now believe is that these "weaker" algorithms are sometimes better able to avoid becoming trapped on ridges.

Third, we have been working on real world applications for computing inverses for prediction problems in weather and geophysics. However, we have found that genetic algorithms and evolution strategies do not work on these inverse problems. We now know that there are ridges in these search spaces that induce "false" optima in the representation spaces, or ridges otherwise slow down the progress of search.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 294–306, 2004.

2 Local Optima and Ridges under Gray Encoding

Let $\Omega = \{0, 1, \ldots, 2^\ell - 1\}$ be the search space which can be mapped onto a hypercube. Elements $x, y \in \Omega$ are *neighbors* when (x, y) is an edge in the hypercube. Bit climbing search algorithms terminate at a *local optimum,* denoted by $x \in \Omega$, such that none of the points in the neighborhood $N(x)$ improve upon x when evaluated by some objective function. Gray codes are often used for bit representations because, by definition, adjacent integers are adjacent neighbors.

Let the objective function be defined on the unit interval $0 \leq x < 1$. We discretize the interval by selecting n points. The *natural encoding* is then a map from Ω to the graph that has edges between points x and $x + 1$ for all $x = 0, 1, \ldots, n-2$. Under a Gray encoding adjacent integers have bit representations that are Hamming distance 1 neighbors which results in the following property.

Theorem 1. *A function $f : \Omega \to \mathbb{R}$ cannot have more local optima under Gray encoding than it does under the natural encoding.*

Under a Gray code, local optima of the objective function considered as a function on the unit interval can be destroyed, but no new local optima can be created. In particular if a function is unimodal under the natural encoding, it is unimodal under Gray code.

But are there unimodal functions where the natural encoding is multimodal? If the function is 1-dimensional, the answer is no. But if the function is not 1-dimensional, the answer is yes. "False" local optima are induced on ridges.

A simplified *ridge* problem appears in Figure 1. Changing one variable at a time will move local search to the diagonal. However, looking in either the x-dimension or the y-dimension, every point along the diagonal appears to be a local optimum. There is actually gradient information if one looks *along* the diagonal; however, this requires either 1) changing both variables at once, or 2) transforming the coordinate system of the search space so as to "expose" the gradient information.

This limitation is not unique to local search, and it is not absolute for genetic algorithms. Any method that searches 1-dimension at a time has the same limitation, including local search as well as simple "line search" methods.

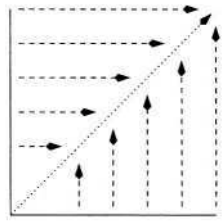


Fig. 1. Local search moves only in the horizontal and vertical directions. It therefore "finds" the diagonal, but becomes stuck there. Every point on the diagonal is locally optimal. Local search is blind to the fact that there is gradient information moving along the diagonal.

A genetic algorithm is not absolutely trapped by ridges. Early population sampling of schema can allow the search to avoid being trapped by "ridges." But genetic algorithms quickly lose diversity and then the search must use mutation or otherwise random jumps to move along the ridge. For example, simple 1-point crossover inherits "fixed but mixed" parameters from parents for the most part. That is, the inherited parameters come directly from the parents without changes except for one parameter that is broken by crossover. Uniform crossover would seem to have the ability to move along ridges: every bit is independently inherited from the 2 parent structures. But Syswerda [5] points out that when using uniform crossover, bits that are common between the two parents are inherited and all non-common bits are *randomly reset* because 0 or 1 is randomly inherited. So the ability of uniform crossover to move along ridges may be no better than that of random mutation.

Kazadi motivated a representation for genetic algorithms called a conjugate schema. Kazadi asserts that sometimes crossover disrupts the efficiency of a real-valued genetic algorithm by producing low fitness offspring. Kazadi proposed a new basis, called a conjugate schema, that minimizes the functional dependencies between the parameters of the objective. This basis creates local separability in the objective function and, hopefully, allows parents to cross with a higher likelihood of strong children. Kazadi uses the eigenvectors of the absolute Hessian matrix to find optimal basis. As noted by the author, "this adaptation is curiously successful on our test functions" [6]. One practical problem is that Hessians require twice differential functions that are difficult to compute.

Wyatt and Lipson [7] take a more "linkage" theoretic view of the crossover problem to arrive at similar conclusions. *Genetic linkage* is a measure of the correlation between functional dependency and gene location. Representations with high linkage will push separable parts of the problem together, and these parts will become useful building blocks for genetic search. In order to find an optimal gene ordering, Wyatt and Lipson also use the eigenvectors of the Hessian around the current best point, and use this ordering for the entire population.

Neither of these studies were concerned with mutation or the direction of the gradient. The proposed ideas may indeed be useful, but these studies take a very narrow view of how the representation is being changed. However, work starting with Salomon, and more recently work on the CMA-ES algorithm, focuses on rotations and concerns about search direction and step size.

Salomon [8] showed that most benchmarks become much more difficult when the problems are *rotated.* Searching a simple 2-D eliptical bowl is optimally solved by one iteration of line search when the ellipse is oriented with the x and y axis. But when the space is rotated 45 degrees (or my some random value), the bowl becomes a ridge and the search problem is more difficult for many search algorithms. Salomon showed that some genetic algorithms, such as the Breeder Genetic Algorithm [9], had been tuned to behave much like line search, largely moving in one dimension at a time. This allows $\mathcal{O}(N\ ln\ N)$ convergence proofs using the assumption that the search problem is decomposable into N subproblems and solved in a piecewise fashion [9].

Salomon points out that *Evolution Strategies* are invariant under rotation. But being invariant under rotation and being able to exploit ridge structures is not quite the same. Oyman et al. [10] show that Evolution Strategies also "creep" on ridge functions. The problem occurred when a (1+10)ES was used for a simple parabolic ridge problem with a 1/5 rule to adjust the step size. Longer jumps in the search space result in poorer evaluation near the ridge. Thus, the adaptive mutation mechanism reduces the step size, until finally the algorithm also creeps along the ridge.

2.1 Benchmarks, Ridges, and Direct Search Methods

Common benchmarks contain a couple of problems with "ridges" features. Figure 2 shows 2-D illustrations of 2 benchmarks. F2 from the De Jong test suite [11] is relatively easy, while the "Rana" function is difficult. F2 was created specifically by Rosenbrock around 1960 to illustrate the weakness of methods which change only one variable at a time during search. Rosenbrock showed that even gradient methods move very slowly on this function because the direction of the gradient significantly changes at each time step.

Rosenbrock proposed a search method that uses the Gram-Schmidt orthogonalization algorithm to adapt the search coordinate system. Later the Nelder-Mead Simplex method was introduced [12], in part to deal with this problem. These methods often compute a direction of improvement based on a sample of points; then, line-search type methods are often used to look for improving moves. In theory, these methods should be able to follow ridge structure *if* they select the correct direction. The potential disadvantage of these methods is that they heuristically compute a direction based on very little information.

One of the fundamental problems that is encountered when trying to compare direct search methods, local search methods, and even different evolutionary algorithms, is the representation and precision used for constructing the search space.

Genetic algorithms and local search (e.g., the Random Bit Climber (RBC) [13]) tend to use low precision bit encodings. Evolution Strategies and direct search methods such as Nelder-Mead use high precision, real-valued representations. The search community has long struggled with the debate over which is better, bit representations or real-valued representations? Unfortunately, different experiments seem to support different conclusions, even when compared on the same test functions. This seems odd and confusing.

Recent work suggests that the choice of real-valued versus bit encodings may not be nearly as important as the level of precision. Precision can dramatically change the rate of convergence. One of the potential advantages of using bit encoding is that they can use lower precision for many applications and achieve faster convergence compared to real-valued encodings. This also makes comparing real-valued representations and bit encoded representations difficult. However, forcing the bit encodings and real-valued encoding to use 32 bit precision just to make them the same is probably not a reasonable solution: precision matters.

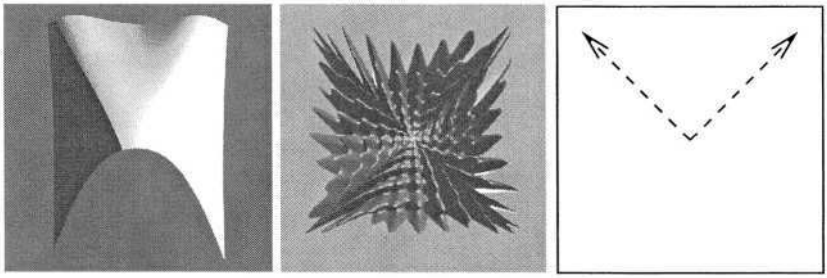


Fig. 2. The leftmost figure is F2 from the De Jong Test Suite, also known as Rosenbrock's banana function. The middle figure is F102, or Rana's function. The rightmost figure is a cartoon showing the "ridges" that lead to the global optimum as well as other competitive local optima.

2.2 Local Search, Ridges, and Precision

We ran a Steepest Ascent Bit Climber on the F2 and Rana test problems at 10 and 20 bits. Results are shown in table 1. The number of steps required to reach a local optimum jumps from about 200 steps under local search at 10 bits of precision to 200,000 or more steps at 20 bits of precision.

As shown in Figure 2, both of these functions have ridge like properties. Search must move toward the upper corners to find the global optimum, as well as the better local optima. The behavior of local search on F2 is shown in Figure 3. The first south to north move occurs in 1 step. The second west to east move occurs in 1 step. The ridge is then encountered after 2 moves. Because the ridge is not exactly at 45 degrees, local search is not completely blind and does not stop. Instead, the ridge becomes a "staircase." Local search makes the smallest move possible and therefore "creeps." The problem is exponentially worse at high precision because the steps of the staircase are exponentially smaller.

Genetic algorithms are often used at 10 bits of precision. Genetic algorithms at 20 bits of precision can be 10 to 100 times slower to converge using 20 versus 10 bits of precision.

Table 1. Results of steepest ascent bit climbing with 100 restarts at 10 and 20 bit resolution. Results are averaged over 30 runs. Mean is calculated using the best of the 100 restarts. Steps is the average number of steps needed to reach a local optimum.

Function	Precision	Mean	Std	Steps	Std
F2, 2-D	10-bits	0.001	0.002	235	30
F2, 2-D	20-bits	4×10^{-7}	1×10^{-7}	2×10^{5}	4×10^{3}
Rana, 2-D	10-bits	-501.9	06.0	225	22
Rana, 2-D	20-bits	-503.0	04.8	3×10^{6}	8×10^{3}

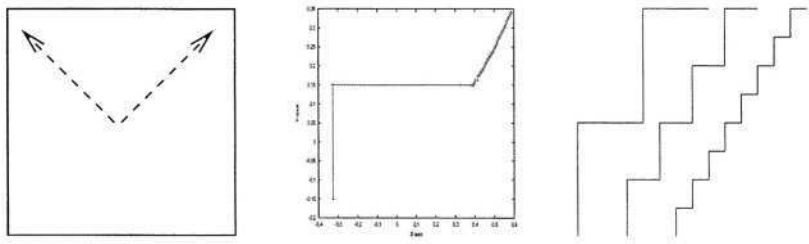


Fig. 3. The "arrows" figure shows the approximate ridge location. The middle figure tracks the movement of an actual run of local search on F2. After 2 moves the ridge is encountered. The rightmost figure: adding 1 bit of precision doubles the number of steps.

3 Ridges and Temperature Inversion

The temperature inversion problem is an atmospheric science application that involves searching for 43 temperatures which produce a set of radiance observations using a forward model.

$$\mathbf{MODEL(temperature.vector)} \longrightarrow \mathbf{radiance.observations}$$

Figure 4 is a 2-D slice taken from the temperature inversion problem in the area around the global optimum. There is a distinct ridge. The companion image shows the location of "false" local optima under Gray code: no improving exists in the x or y dimension. There are a surprising number of false local optima that trap local search. Such ridges occur *throughout* the search space and respond to the nonlinear interaction between variables: changing a parameter in dimension x simultaneously changes the location of the ridge in many other dimensions.

Empirical results show that many evolutionary algorithms are trapped in local optima at about the same rate as local search, including a Steepest Ascent

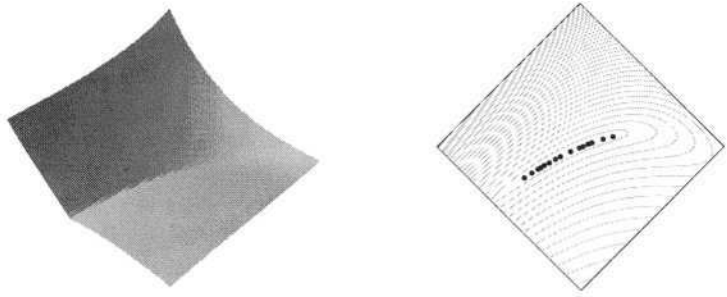


Fig. 4. On the left is a 2-D slice near the global optimum of the 43-D temperature inversion problem. The points in the rightmost figure shown where "false" local optima occur.

Table 2. Results of steepest ascent bit climbing (SABC) and a rotated local search method. No restarts were used in these experiments. **Mean** is the mean best over 30 experiments, and best is the best of the 30 experiments.

Functions	Search	Best	Mean	Std	Steps	Std	Evals	Std
F2	SABC	+4.5E-07	+5.4E-07	+1.2E-07	6,193	814	247,710	32,541
	PCA SABC	+3.1E-10	+2.5E-07	+2.5E-07	138	58	7,603	3,189
Rana	SABC	-510	-417	87	208	321	8,305	12,822
	PCA SABC	-511	-480	24	23	6	1,262	341

Bit Climber, CHC, a (30,210)ES and a (30+210)ES with standard mutation adaptations and no rotations, PBIL and Differential Evolution.

4 Rotating Search Coordinates

One way to deal with ridges in the search space is to change the search coordinates. One way to do this is to use a rotated representation. Evolution Strategies use a heuristic rotation by adapting a set of rotation strategy parameters via evolution [14]. However, adapting rotation "strategy" parameters of the form used by Evolution Strategies is too imprecise and impractical for large problems.

A standard way of computing a rotation is to use Principal Component Analysis. Given a data set of sample points, an eigenvalue/eigenvector decomposition is performed. The eigenvectors are represented by a rotation matrix $\mathbf{R}$. Let Λ be the diagonal eigenvalue matrix. Let $\mathbf{X}$ represent a matrix of data vectors. Using PCA we find $\mathbf{R}$ and Λ such that

$$\mathbf{R} \cdot \mathbf{X}\mathbf{X}^T = \Lambda\mathbf{R}$$

For a single search point represented by the vector $\mathbf{x}$ we compute $\mathbf{xR}$, which is the projection of the point $\mathbf{x}$ into the space defined by $\mathbf{R}$. The rotation matrix is orthonormal, so a simple correction is also needed to translate and re-center the rotation during search.

To find a structure such as a ridge, PCA can be used to sample locally and isolate a subset of the better sample points. For example, sample 20 points and then apply PCA analysis to the 10 best solutions. While this can give a correct rotation, the direction of maximal variance might be in the direction of the gradient if the samples are on a ridge, or the maximal variance may be orthogonal to the gradient if the sample is drawn from a sloping plateau.

Another approach is to use the Gram-Schmidt (GS) orthogonalization algorithm to rotate the space. Often the Gram-Schmidt algorithm is used to construct a representation that is orthogonal with respect to two points in the search space–such as the best two points seen so far. This is a heuristic way of determining a useful "rotation" for changing the problem representation.

In Table 2 Steepest Ascent Bit Climbing (SABC) with a Gray Code representation is compared with SABC using PCA to rotate the search space. For the PCA, 15 points were sampled, with PCA applied to the best 8. The speed-up is *dramatic* using rotated representations.

5 Constructing Higher Dimensional Test Problems

We have found that there are no good test functions that are difficult and which also scale up to higher dimensions. Researchers have rotated existing test functions to make them more difficult. But there still exists a single rotation that converts the problem back into a relatively easy problem; and this does nothing to address scalability.

Functions with two variables are often scaled to higher dimensions using an *expansion* function. Different expansion functions have different properties. Sometimes subfunctions are added together with no interactions between subfunctions. The problem here is the linear combination of subfunctions results in a test problem that is *separable* when there are no interfunction interactions between the parameters. Specifically, it is possible to optimize each component (e.g., $f(x_1, x_2)$) independently of the other parameter values.

Expansion functions that create *non-separable* problems are often generalized in the following way:

$$\text{Expansion Method 1:} \quad f(x) = \sum_{i=1}^{n-1} f(x_i, x_{i+1})$$

However, the symmetry that exists in this function can make some problems easier to solve in higher dimensions. Figure 5 (topmost) shows the two dimensional slices taken from the three dimensional Rosenbrock function when generalized in this way. In general, the surfaces are not as difficult as the original Rosenbrock function, and the problem becomes *much* easier in higher dimensions. In order to retain the original ridge structure, the following expansion can be used:

$$\text{Expansion Method 2:} \quad f(x) = \sum_{i=1}^{\lfloor n/2 \rfloor} f(x_{2i-1}, x_{2i}) + \sum_{i=1}^{\lfloor (n-1)/2 \rfloor} f(x_{2i+1}, x_{2i})$$

This creates a *non-separable,* higher dimension problem that preserves the ridge features of the original problem. Figure 5 (bottom) illustrates using Expansion Method 2 to create the three dimensional Rosenbrock function. The first two slices retain the long narrow ridge that characterizes the Rosenbrock function. The interaction between parameters one and three creates a multimodal surface from which it is difficult for a coordinate strategy to escape. This same pattern extends to slices of 5 and 10 dimensional functions contructed using Expansion Method 2: more of the features that make the primitive 2-D function difficult are preserved in the expanded functions.

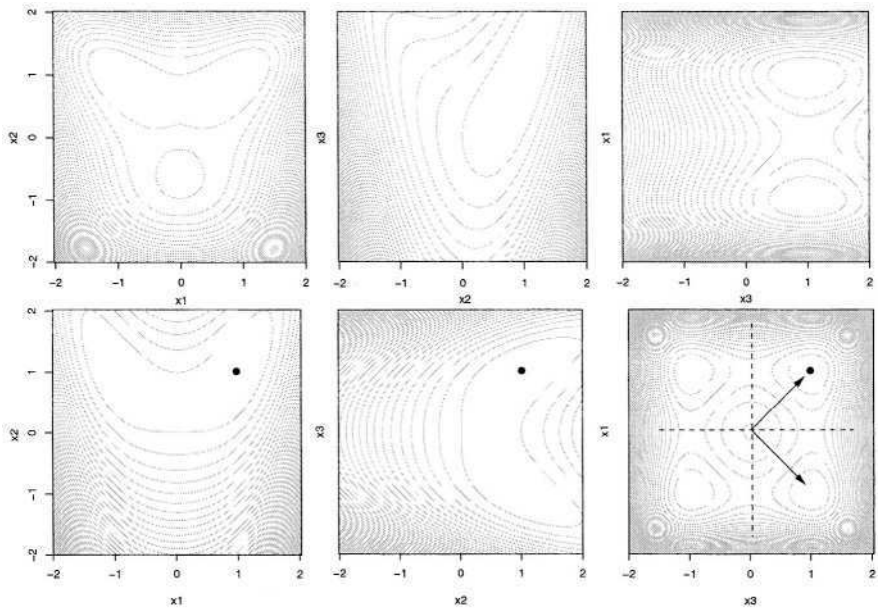


Fig. 5. Results of expanding Rosenbrock's function in three dimensions using Expansion Method 1 are shown in the top 3 slices of the space: with the possible exception of the first slice, the search space is simpler and easier than the original problem. Results of expanding Rosenbrock's function in three dimensions using Expansion Method 2 are shown in the bottom 3 slices of the space: the first two slices retain the long narrow ridge and the third slice is a multimodal surface.

We applied the local search SABC algorithm with and without a PCA rotated representation to 5-D and 10-D versions of Rosenbrock's banana function (F2) and the Rana function. We also ran a 10-D temperature inversion problem looking for temperature for the first 10 kilometers of the atmosphere. PCA was applied after every step, which adds to the number of evaluations. The number of steps taken during search is 5 to 10 times less under SABC with PCA compared to the non-rotated representations on the 5-D problems. The number of steps is 2 to 3 times less under SABC with PCA compared to the non-rotated representations on the 10-D problems. Using a rotated representation is more effective on F2 than Rana. This may be because the ridge is the only thing that makes F2 difficult, while Rana is also extremely multimodal. There is a clear advantage using PCA on the 5-D problems; the advantage is less clear at 10-D for Rana. For the 10-D temperature problem, using PCA significantly reduces both the error and the number of evaluations.

Table 3. The results of applying local search SABC with and without PCA rotated representations on 5-D and 10-D versions of the Rana and F2 functions using Expansion Method 2. At 5-D, PCA used the best half of 40 samples; at 10-D PCA used the best half of either 80 or 74 samples to compute rotations.

Function	Search	Best	Mean	Std	Steps	Std	Evals	Std
F2, 5-D	SABC	2.4E-06	2.4E-06	8.1E-08	11,663	1,415	1,166,268	141,503
	PCA SABC	5.3E-07	2.4E-06	1.3E-06	1,057	177	148,042	24,800
Rana, 5-D	SABC	-386	-313	49	506	915	50,551	91,540
	PCA SABC	-399	-310	42	112	167	15,662	23,318

Function	Search	Best	Mean	Std	Steps	Std	Evals	Std
F2, 10-D	SABC	5.9E-06	6.1E-06	1.3E-07	29,437	1,865	5,887,321	372,921
	PCA SABC	3.8E-06	5.9E-06	2.2E-06	8,915	406	2,496,201	113,735
Rana, 10-D	SABC	-427	-354	40	758	940	151,628	187,959
	PCA SABC	-411	-308	47	525	632	146,917	176,958
Temp, 10-D	SABC	11,607	16,026	2,497	373	64	41,064	7,092
	PCA SABC	5,353	10,121	2,910	109	36	20,100	6,722

6 Discussion and Related Work

The goal of this paper is to highlight and explain the ridge problem and explore the use of rotated representations. Rotated representations could be used in conjunction with various types of evolutionary algorithms. And despite work in this direction, most of the evolutionary computation field has not looked seriously at rotated representations.

There is at least one algorithm that already makes extensive use of rotated representations and has mechanisms to address some of the key questions that arise when using rotated representations: *Covariance Matrix Adaptation,* or CMA, rotates and scales the mutation operators used by an Evolution Strategy. The key question is what kind of sample should be used when computing rotations. If a localized sample is taken and then the best points (e.g., the best half of the sample) are used to calculate the eigenvectors and eigenvalues, the direction of maximum variance can be in the direction of the gradient or it can be orthogonal to the gradient.

Recall that the Gram-Schmidt algorithm is often used to construct a representation that is orthogonal with respect to two points in the search space–such as the best two points seen so far. This kind of approach used less information, but emphasizes knowledge about gradient based on the most recent move

or moves. This is a more localized and heuristic way of determining a useful "rotation" for changing the problem representation.

In effect, PCA exploits information about variance, whereas Gram-Schmidt uses path information. The path information acts as a kind of "momentum" term that keeps the search moving in it's current direction. Simple empirical experiments show that path information is most useful when approaching a ridge, or when following a straight ridge. But path information is sometimes misleading, for example on a curved ridge. On a curved ridge, calculating the direction of maximum variance helps to track the changing gradient.

These ideas are already exploited by the CMA-ES algorithm.

6.1 CMA-ES

Traditional *evolution strategies* produce offspring based on adaptive strategy variables that attempt to improve the likelihood of producing better offspring. *Strategy parameters* are coded onto the chromosome along with the objective parameters and are adapted indirectly based on the assumption that highly fit individuals will carry better strategy parameters. The mutation strength of the strategy parameters must be high enough to create significant differences between offspring while minimizing adaption time [15]. Ostermeier et al. attempted to dampen the mutation strength without compromising adaption speed [16]. Hansen et al. offers a solution that completely removes the mutation strength when adapting the strategy parameters [15]. Unfortunately, this solution cannot be easily extended to other strategy parameters that control the angle of rotation between each dimension, as necessary in *correlated mutations.* Without this rotational adaptation, the algorithm's success is limited on ridge functions.

Covariance Matrix Adaptation, or CMA, uses a covariance matrix to rotate and scale the mutation distribution [15]. The covariance matrix is an estimate based on the evolution path and, when applicable, information extracted locally from strategies with large populations [17]. Hansen and Ostermeier define the reproduction phase from generation g to generation $g+1$ as:

$$x_k^{(g+1)} = \langle x \rangle_\mu^{(g)} + \sigma^{(g)} \mathbf{B}^{(g)} \mathbf{D}^{(g)} z_k^{(g+1)}$$

where $z_k^{(g+1)}$ are randomly generated from an $N(0, I)$ distribution. This creates a set of base points that are rotated and scaled by the eigenvectors ($\mathbf{B}^{(g)}$) and the square root of the eigenvalues ($\mathbf{D}^{(g)}$) of the covariance matrix C. The single global step size, $\sigma^{(g)}$, scales the distribution based on adaptation. Finally, the points are translated to center around $\langle x \rangle_\mu^{(g)}$, the mean of the μ best parents of the population.

Instead of only using a single generation to compute covariance, CMA-ES utilizes the entire evolution path, called *cumulation.* The evolution path updates at each generation using a weighted sum of the current evolution path, $p_c^{(g)}$, with the vector that points from the mean of the μ best points in generation g to the mean of the μ best points in generation $g+1$. When a larger population (λ) is

used, the best μ individuals may help describe the topology around the mean of the current generation. This is potentially useful information. Assuming $\mathbf{Z}^{(g+1)}$ is the covariance of the μ best individuals, and $\mathbf{P}^{(g+1)}$ is the covariance of the evolution path, the new covariance matrix is:

$$\mathbf{C}^{(g+1)} = (1 - c_{cov})\mathbf{C}^{(g)} + c_{cov}\left(\alpha_{cov}\mathbf{P}^{(g+1)} + (1 - \alpha_{cov})\mathbf{Z}^{(g+1)}\right)$$

Where c_{cov} and α_{cov} are constants that weight the importance of each input.

7 Conclusions

The CMA-ES algorithm has already raised interesting issues about how best to implement Evolution Strategies. Traditionally an ES encodes $\mathcal{O}(N^2)$ rotation or covariance parameters onto the chromosome to be evolved along with the N object parameters. Such rotations are simply not practical on large problems. Empirical tests of the CMA-ES algorithm are very positive. The use of rotated representations could produce a fundamental change in the theory and application of Evolution Strategies. The use of rotated representations also needs to be explored for a wider range of evolutionary algorithms.

Acknowledgments. This work was supported by National Science Foundation grant IIS-0117209.

References

1. Rosenbrock, H.: An automatic method for finding the greatest or least value of a function. Computer Journal **3** (1960) 175–184
2. Brent, R.: Algorithms for Minization with Derivatives. Dover (1973)
3. Whitley, D., Barbulescu, L., Watson, J.: Local Search and High Precision Gray Codes. In: Foundations of Genetic Algorithms FOGA-6, Morgan Kaufmann (2001)
4. Rana, S., Whitley, D.: Representations, Search and Local Optima. 14th National Conf on Artificial Intelligence AAAI-97, (1997) 497–502
5. Syswerda, G.: Simulated Crossover in Genetic Algorithms. In Whitley, D., ed.: FOGA - 2, Morgan Kaufmann (1993) 239–255
6. Kazadi, S.T.: Conjugate schema and basis representation of crossover and mutation operators. Evolutionary Computation **6** (1998) 129–160
7. Wyatt, D., Lipson, H.: Finding building blocks through eigenstructure adaptation. In: GECCO, Morgan Kaufmann (2003) 1518–1529
8. Salomon, R.: Reevaluating Genetic Algorithm Performance under Coordinate Rotation of Benchmark Functions. Biosystems **39(3)** (1960) 263–278
9. Mühlenbein, H., Schlierkamp-Voosen, D.: Predictive Models for the Breeder Genetic Algorithm. Journal of Evolutionary Computation **1** (1993) 25–49
10. Oyman, A., Schwefel, H.B.H.: Where elitists start limping: Evolution strategies at ridge functions. ppsn5, Springer-Verlag (1998) 34–43
11. DeJong, K.: An Analysis of the Behavior of a Class of Genetic Adaptive Systems. PhD thesis, University of Michigan, Dept. of Computer and Communication Sciences, (1975)

12. Nelder, J., Mead, R.: A simplex method for function minimization. Computer Journal **7** (1965) 308–313
13. Davis, L.: Bit-Climbing, Representational Bias, and Test Suite Design. In Booker, L., Belew, R., eds.: Proc. of the 4th Int'l. Conf. on GAs, Morgan Kaufmann (1991) 18–23
14. Bäck, T.: Evolutionary Algorithms in Theory and Practice. Oxford University Press (1996)
15. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation **9** (2001) 159–195
16. Ostermeier, A., Gawelczyk, A., Hansen, N.: A derandomized approach to self-adaptation of evolution strategies. Evolutionary Computation **2** (1994) 369–380
17. Hansen, N., Mãller, S., Koumoutsakos, P.: Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation. Evolutionary Computation **11** (2003) 1–18

Non-stationary Subtasks Can Improve Diversity in Stationary Tasks

Christopher Willis-Ford and Terence Soule

University of Idaho, Department of Computer Science, Moscow, ID 83844, USA
cwillisf@acm.org, tsoule@cs.uidaho.edu

Abstract. Low diversity in a genetic algorithm (GA) can cause the search to become stagnant upon reaching a local optimum. To some extent, non-stationary tasks avoid this problem, which would be a desirable feature of GA for stationary tasks as well. With this in mind, we show that several methods of introducing artificial non-stationary elements help to promote diversity in a GA while working on an inherently stationary task. By analyzing online and offline diversity and fitness measures, we show that it is possible to improve overall performance through this technique, and that some measures intuitively related to performance are misleading.

1 Introduction

Considerable research has been done in the field of non-stationary tasks in genetic algorithms (GA). Search techniques such as random restart, adaptive mutation [1], and the island model [2] can be quite effective in helping an evolutionary system track a dynamic problem. One effect of this tracking is that the GA is not allowed to converge on a particular local optimum – a common pitfall when solving stationary problems using GA. Relatively little exploration has been done to investigate whether this benefit can be applied to stationary tasks as well.

One problem with the GA approach is the tendency for the search to become 'stuck' on local optima. If one individual within the population happens to find a very good solution relative to the other individuals, it is likely to have considerable influence over future generations. This effect compounds over several generations and frequently results in a near total loss of diversity in the population: premature convergence. Such low diversity results in redundancy: two individuals representing the same solution store less information than two representing different solutions. This in turn means that fewer solutions are explored, and the search is less complete.

In this paper we propose that lessons can be learned from non-stationary tasks and applied to stationary ones. Specifically, we show that diversity is improved when a stationary problem is made non-stationary, and that the benefit of doing so outweighs the cost.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 307–317, 2004.

2 Terminology

One may expect that GA will perform better when diversity is high, and when convergence is avoided. In order to test this intuition, it must be made precise. This section will define the terms that will be used for the remainder of this discussion. When necessary, we present a concrete definition for a term with respect to both a binary-valued and a real-valued problem.

The binary-valued problem we will address is target matching: the goal is to evolve a solution identical to a predetermined target. The fitness function for an individual $\boldsymbol{a}$ is equal to the Hamming distance between that individual and the target $\boldsymbol{t}$, and is to be minimized.

Our real-valued problem is a function optimization, which is similar to target matching. For ease of reference, we will discuss a two-dimensional problem, in which the fitness function $f(\boldsymbol{a}) = F(a_x, a_y)$ is to be maximized. This leads naturally to discussion of the fitness landscape as a three-dimensional space, in which peaks represent good solutions, and valleys represent bad ones.

Performance of the GA will be measured in terms of its offline performance: the fitness value of the single most fit individual encountered at any point during the run. Online performance – the average fitness of individuals per generation – will be noted, but is not the primary measure of performance. In fact, an increase in diversity will almost guarantee a drop in online performance, as fewer individuals will cluster at peaks in the fitness landscape.

Diversity in a GA run is a critical measure of the behavior of the search. Unfortunately, it is also a very complex and ambiguous term. We define two types of diversity: online and offline. *Online diversity* refers to the 'spread' of individuals across the search space in a given generation, measured in, for instance, standard deviation. *Offline diversity* is that of the entire run viewed as a whole – an approximation of search space coverage.

The base diversity function for a given population $D(P_i)$ will be defined differently for each problem, but will commonly be based on standard deviation or average Hamming distance between individuals in the population.

3 Background

Many low-cost methods for increasing diversity have been researched. Perhaps the simplest method is to increase the crossover or mutation rate inherent in the GA. Another low-cost method of promoting diversity is the random restart method, in which the entire population is re-seeded with random values. This restart may happen at a fixed interval, or when diversity becomes low, etc. These low-cost methods share one downside: they cause little improvement in the performance of the search [3].

On the other hand, fitness sharing [4] is extremely effective in increasing diversity, and has a marked positive effect on performance as well. In fitness sharing, the fitness score associated with a particular location in the search space is seen as a resource; individuals at that location must share that resource. As a

result, if many individuals share one particular high-fitness location, the fitness of each individual might be lower than that of a single individual situated in a region of lower fitness. The effect of fitness sharing is that individuals are forced to spread out over a larger amount of the search space, helping to avoid the loss of diversity associated with population convergence and encouraging the GA to seek better solutions. Many diversity-promoting techniques that are effective in improving fitness also suffer fitness sharing's major drawback: a prohibitive computational cost [4].

4 Improving Diversity

One might expect that diversity levels in non-stationary tasks would be higher than those in stationary tasks – if nothing else, a dynamic fitness landscape could be expected to discourage convergence on one particular location in the search space. It would seem then that transforming a stationary task into a non-stationary one would yield higher levels of diversity. A stationary problem is made non-stationary by changing the environment within the GA during the run – effectively making a static fitness landscape dynamic. One immediate benefit of this method is that the computational cost is directly controllable: in fact, all of the methods presented here add a constant-time operation to a fitness evaluation, and no more.

This type of change is best expressed in terms of an altered fitness function with an added dependence on time. In general, if the original fitness function is $f_{\boldsymbol{a}} = f(\boldsymbol{a})$, and t represents the current time in the context of the GA run, then the new fitness function will be of the form

$$f'_{\boldsymbol{a}} = f'(t, \boldsymbol{a}, f_{\boldsymbol{a}}) \quad . \tag{1}$$

This investigation focuses on two optimization problems of two real-valued variables. The first function (Problem 1) being optimized was

$$f_1(\boldsymbol{a}) = \frac{1 - (|\boldsymbol{a}_x| + |\boldsymbol{a}_y|)}{2} \cdot \frac{\cos(\boldsymbol{a}_x \cdot \pi \cdot peaks) + \cos(\boldsymbol{a}_y \cdot \pi \cdot peaks) + 2}{4} \quad , \tag{2}$$

where *peaks* = 8. The fitness landscape is depicted graphically in Fig. 1. This landscape is somewhat difficult, but primarily serves an illustrative purpose. The second function (Problem 2) was the Griewangk function, frequently used as a benchmark for genetic algorithms. The Griewangk function is scalable to any number of parameters, but becomes simpler as it scales up [5]. The two-parameter Griewangk landscape, the one used here, is shown in Fig. 2. The function was expanded and simplified to

$$f_2(\boldsymbol{a}) = 1 + \frac{\boldsymbol{a}_x^2 + \boldsymbol{a}_y^2}{4000} - \cos x \cdot \cos \frac{y}{\sqrt{2}} \quad . \tag{3}$$

Both functions were normalized to output in the range of 0 to 1, and the Griewangk function was inverted in order to make it a maximization problem.

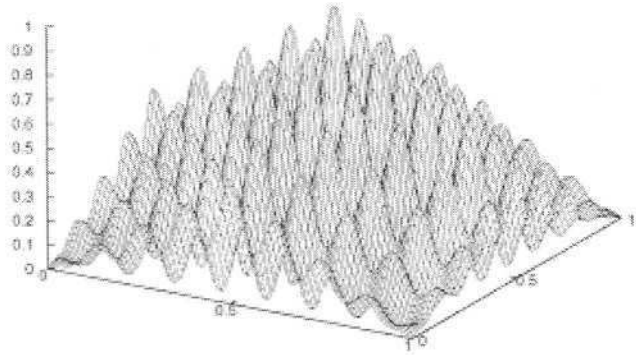


Fig. 1. Fitness landscape for Problem 1. The global optimum is at the center, coordinates (0.5, 0.5). Every depression has a point of zero fitness at its center, including those surrounding the global optimum

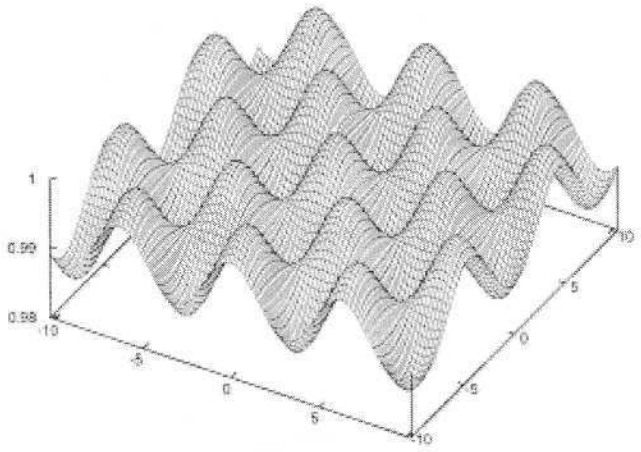


Fig. 2. Fitness landscape for Problem 2, the Griewangk function. The global optimum is again at the center, at coordinates (0, 0). This is a detail view of the center at 50x zoom. Note that pits do not reach zero fitness

The domain of each parameter in the first landscape was -1 to 1; in the second, -512 to 512. These were normalized to take inputs in the range from 0 to 1..

All GA runs were made to run for exactly 300 generations. GA parameters are depicted in Table 1, and were the same for both problems. Solutions were represented as ordered pairs of floating point numbers: (a_x, a_y). As such, mutation of a coordinate was done by adding a random number (Gaussian distributed about 0). Crossover of two parents was implemented to yield two new coordinates that lay along the segment between the parents, 1% of the distance from either end. Mathematically, if a' was the first child, and a and b were the

parents, then

$$\boldsymbol{a}' = \left(\frac{\boldsymbol{a}_x * 99 + \boldsymbol{b}_x}{100}, \frac{\boldsymbol{a}_y * 99 + \boldsymbol{b}_y}{100} \right) \quad , \tag{4}$$

and similarly for $\boldsymbol{b}'$ with weights reversed. Elitism was based on the raw fitness function $f(\boldsymbol{a})$ rather than $f'(\boldsymbol{a})$.

Table 1. Parameters for each GA run

Population size:	500
Selection:	Tournament of 3
Chance of Crossover:	100%
Chance of Mutation:	0.1% per coordinate ($\boldsymbol{a}_x$ and $\boldsymbol{a}_y$)
Elitism:	2 individuals

4.1 Baseline: Random Noise

Perhaps the simplest method of making the landscape dynamic is to randomly modify each individual's score during fitness evaluation. This randomness ensures that individuals that are on or near a peak in the unmodified fitness landscape are not necessarily the individuals that receive the highest modified score – effectively, peaks are constantly shifting at a rate faster than once per generation. This could cause individuals near a good solution to not always be favored, and thus help maintain diversity. Of course, there is also a clear downside: in some cases, this practice could cause the GA to completely miss a good solution.

Depending on the size of the random change, relative to the unmodified fitness, the effects of the random noise approach vary wildly. In extreme cases, the search either exhibits behavior indistinguishable from the original GA if noise is low, or effectively becomes random search if noise is high. It is worth noticing that in random search, convergence is avoided and so diversity is very high. Thus, it is clear that diversity by itself does not improve GA performance, as random search performs poorly. In moderation, the 'noisy fitness' tends to have slight effects toward both improving diversity and delaying convergence, but does not have a significant effect on performance [6].

Our goal in this research is to discover whether it is possible to use non-stationary modifications to a stationary GA in such a way as to improve both diversity and fitness, with low computational cost. To this end, we explored three methods detailed in the following sections.

4.2 The Moving Target Approach

The fitness function representing the "moving target" approach (MT) is

$$f'(t, \boldsymbol{a}, f_{\boldsymbol{a}}) = w f_{\boldsymbol{a}} + (1 - w)\, dist(x, T(t)) \quad , \tag{5}$$

where w is a weighting factor, $dist\,(x, y)$ calculates the distance between two individuals, and the function $T\,(t)$ constructs an individual that represents the target at time t. The $dist\,(x, y)$ function measures the average Hamming distance in the population for bit string problems, or the sum of the standard deviation across all values in real-valued problems.

The implementation of MT discussed here regards a secondary target with an additive effect on the raw fitness function, with linear falloff. The target begins at a random location within the search space, and moves to new random locations throughout the run. Moves occur at most every 20 generations, but only when online diversity falls below 0.1% of the search space.

The desired effect of the secondary target is to temporarily encourage the population to explore in a direction that it would not otherwise. The secondary target 'bends' the landscape around a new peak, such that population travel toward that peak earns higher fitness than travel away. For instance, if the secondary target positions itself in the middle of a low-fitness region, that region is then easier for the population to enter or cross. A potentially beneficial side effect is that if the target then leaves that region, the population should then leave th low-fitness region for higher-fitness regions of the search space – which have not necessarily already been explored.

One potential drawback of the MT approach is that the secondary target may "distract" the GA from finding the global optimum. An extreme example is that of a deceptive fitness function: if the secondary target is placed such that the population is encouraged to explore the deceptive portion of the landscape, it will then be difficult to lead exploration back toward the true optimum, even if the secondary target is moved to encourage such behavior. In fact, in order to reliably induce such action, the weight of the secondary target must dominate that of the raw fitness – and in that situation, the GA cannot be expected to find good solutions to anything but the secondary target's fitness function. Clearly this is an extreme example, but it is not difficult to see this situation arising in non-deceptive landscapes as well.

4.3 The Peak Removal and Peak Erosion Approaches

The peak removal (PR) approach attempts to guide the population away from good solutions after they have been found. The fitness function that describes this is

$$f'\,(t, \boldsymbol{a}, f_{\boldsymbol{a}}) = f_{\boldsymbol{a}} \cdot S\,(t, \boldsymbol{a}) \quad , \tag{6}$$

where the function S calculates the 'squash factor' associated with that particular location in the search space at that particular time. This method is similar in concept to Tabu search, described in [7]. Here, we focus exclusively on the idea of simplistically modifying the fitness landscape.

The squash factor is determined by keeping a list of local optima that have been found recently, and negating fitness values within that range. That is,

$$S(t, \boldsymbol{a}) = \begin{cases} -1 \text{ if } dist(\boldsymbol{a}, \boldsymbol{p}_i) < \delta \text{ for any of the past } n \text{ local optima } \boldsymbol{p}_i \\ 1 \quad \text{otherwise.} \end{cases} \tag{7}$$

Whenever diversity falls below a certain level, the current best position in the area is recorded and added to the set of previously found peaks, $\boldsymbol{p}$. If the list is longer than n, the oldest element is discarded. From then on, any individual within a certain distance δ of that known peak is penalized, encouraging exploration of other regions of the search space. It is important to note that, though selection is based on the modified fitness, elitism is based on the raw fitness. This prevents the GA from completely losing a good solution when it is squashed.

One critical factor in using PR is choosing the right δ. If it is chosen too large, peaks may be missed; if it is too small, then some peaks (with more rounded tops) will not be eliminated. It is conceivable that some sort of adaptive method might determine a good value for δ every generation (or even for every peak), but developing such a method is beyond the scope of this discussion.

Peak Erosion (PE) is a relaxed form of PR. Rather than completely inverting a peak if it is on the peaks list, the fitness is decreased according to how many times that region of the search space is found on the list. Specifically, the squash factor for PE is defined as

$$S(t, \boldsymbol{a}) = 0.9^m \ , \tag{8}$$

where m is the number of the past n local optima $\boldsymbol{p}_i$ satisfying $dist(\boldsymbol{a}, \boldsymbol{p}_i) < \delta$.

Peak Erosion is intended to avoid the possibility of prematurely forcing the population to abandon a particular peak. Rather than immediately removing a peak, PE causes the population to gradually become 'disinterested' in that region of the search space. This allows a few generations of further exploration, perhaps yielding a slightly better solution, before the population moves on to the next area of interest.

For this investigation, δ was chosen to be equal to $\frac{1}{16}$ of the range of each parameter for Problem 1. That is, a squashed peak implied a circular "pit" in the fitness landscape, with radius equal to $\frac{1}{16}$ of the horizontal or vertical span of the square space. For Problem 2, the squash was much smaller: only approximately 0.5% of the range of each parameter. This is because peaks in Problem 2 are proportionally smaller than those in Problem 1. The list of recent peaks was 20 peaks long. Settings for both PR and PE were the same; only the $S(t, \boldsymbol{a})$ function was changed.

5 Results

Results were collected for 100 runs of each version of the GA: normal, MT, PR, and PE. The diversity, maximum fitness, and average fitness were calculated for each generation of each run. For Problem 1, Fig. 3 shows the average diversity

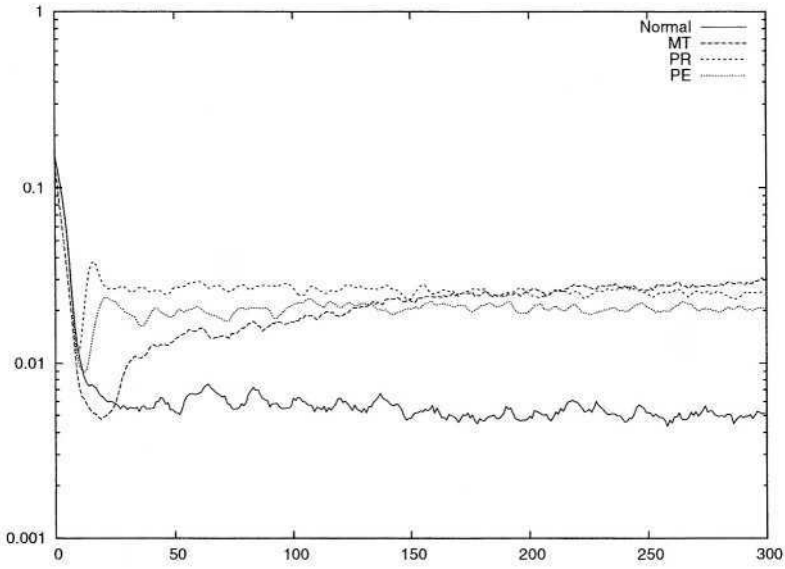


Fig. 3. Online population diversity with logarithmic scale. Diversity levels are comparable for all three modified GAs; lower diversity is seen in the unmodified GA

Table 2. Summary of averaged results. Each method was run 100 times; the values here represent the average of each measurement across those 100 runs

Problem 1:		
Method	Offline Fitness	Offline Diversity
Normal	0.927	0.0492
Moving Target	0.814	0.0963
Peak Removal	0.957	0.1115
Peak Erosion	0.964	0.1009
Problem 2:		
Normal	0.982	0.0495
Moving Target	0.917	0.0755
Peak Removal	0.995	0.0590
Peak Erosion	0.997	0.0568

at each generation for each version of the GA, Fig. 4 shows the average fitness, and Fig. 5, the maximum fitness at each generation[1]. The maximum fitness and offline diversity for each method, and both problems, is shown in Table 2.

Several interesting trends can be noted in these results. First, it is clear that all modifications had a positive effect on offline diversity for both problems (Table 2) – Student's two-tailed t-test verifies that fact with $P < 1\%$. Looking

[1] Graphs for Problem 2 are similar, omitted due to space constraints

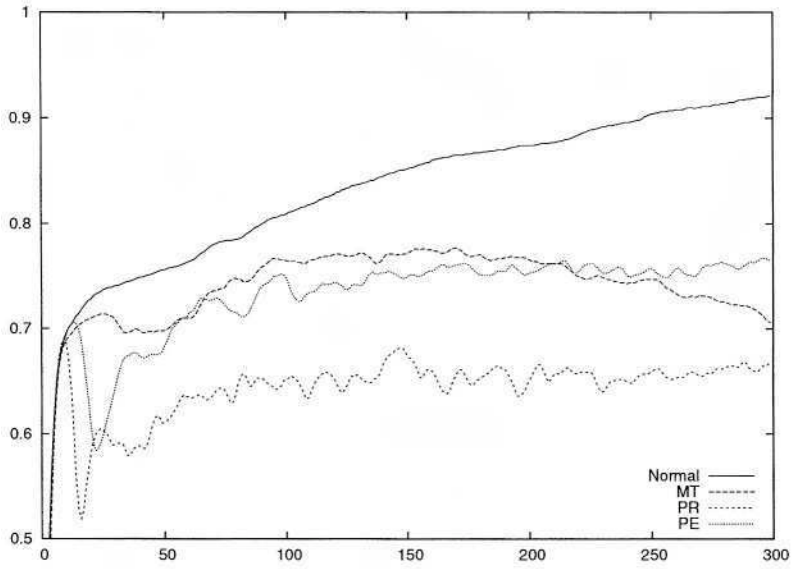


Fig. 4. Online fitness with lower limit of 0.5 to emphasize detail. The unmodified GA clearly has higher average fitness than the others, and Peak Removal is lowest. The Moving Target and Peak Erosion methods are similar in online fitness

at the Problem 1 graphs as a guide, initial diversity levels are high (as expected for an initially random population), then as poorly fit individuals are culled diversity drops dramatically. The stationary GA never recovers from this drop in diversity, though the others seem to exhibit a reactive bloom in diversity in just a few generations. Subsequently, PR and PE maintain a roughly static level of diversity, whereas with MT diversity maintains a slow rate of increase. Similar behavior was seen in Problem 2.

The fitness graphs are even more interesting. First, online fitness is significantly lower for MT, PR, and PE than in the stationary GA, with the lowest results in PR (Fig 4). On the other hand, offline fitness is significantly lower for MT, but higher in both PR and PE (Fig 5). All of these differences are statistically significant (Student's t-test $P < 1\%$). These results may not be intuitive, but do make sense: they imply that PR and PE are better at forcing the population to leave peaks that have already been found, thus forcing more individuals into low-fitness regions of the search space and decreasing average fitness. Again, results for Problem 2 are similar.

6 Conclusions

There are three conclusions that may be drawn from these results. First, diversity in static tasks can indeed be improved by adding non-stationary subtasks, which

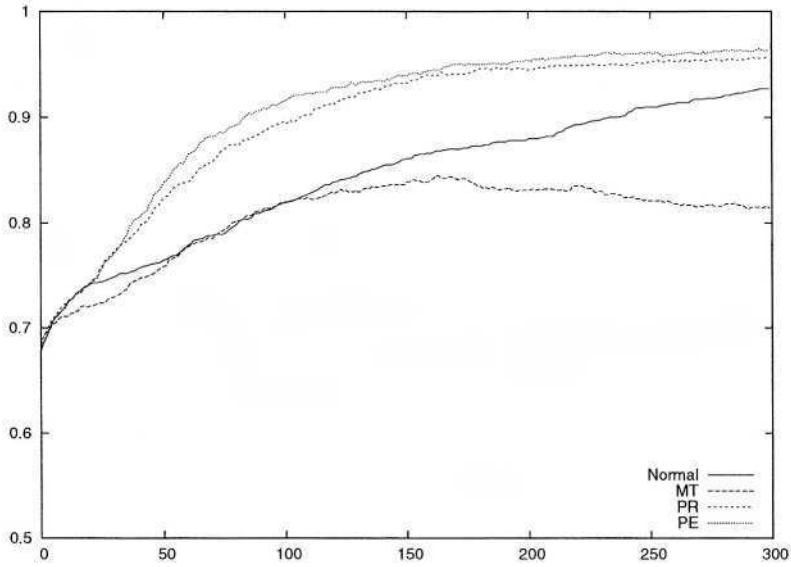


Fig. 5. Maximum fitness at each generation, with lower limit of 0.5. This represents offline fitness "in progress." The Moving Target method has a negative impact on offline fitness, whereas the Peak Removal and Peak Erosion methods have a positive influence, especially in early generations

can also be seen as making the static landscape dynamic. Second, non-stationary subtasks can improve performance – but not necessarily in proportion to the increase in diversity. Finally, as an extension of the former points, online fitness and diversity are not necessarily accurate predictors of offline fitness.

All three modifications exhibited significantly increased diversity. Intuitively, it makes sense that a dynamic fitness landscape would lead to increased diversity: if the local optimum that the GA is working toward repeatedly moves or changes before the population converges, then the loss of diversity related to full convergence will never occur. In the case of the Moving Target, the population is attracted in varying directions in order to pull individuals away from discovered peaks in the fitness landscape. In Peak Removal or Erosion, peaks found are made less attractive – metaphorically pushing the population away. In a rank-based selection method, these pulling and pushing effects are basically equivalent, as the diversity results suggest (Fig 3).

Though intuitively in may seem that increased diversity should lead directly to increased performance, it is clear that there is at least some decoupling between the two. PR and PE perform better than a normal GA; MT performs significantly worse – even though the diversity levels in PR, PE, and MT are comparable. It would seem that taking more information about the current landscape and population distribution into account will generate more intelligent – and more effective – methods of modifying that landscape at runtime. Of course,

in doing so one must consider the computational cost of the intelligence as compared to the resulting fitness benefit. We hope to test these ideas by exploring other potential dynamic subtasks in future work.

The relationship between online fitness and offline fitness is also indirect, largely due to the presence or absence of diversity. Specifically, a drop in online fitness does not necessarily correspond to a drop in offline fitness, as shown by PR and PE. In fact, it would seem that the ideal case would have high online and offline diversity, and low online fitness – though MT seems to fit this pattern and yields low offline fitness as well. More research needs to be done in order to discover the difference between the apparent 'good' and 'bad' sorts of diversity. Early results seem to indicate that diversity is only 'good' as long as some amount of convergence is allowed, and that convergence must be related to an optimum in the static landscape. Allowing total convergence is to be avoided, so perhaps an adaptive method which balances a diversification process against runtime-observed convergence would be effective. This is an area we hope to explore further.

References

1. D. Whitley, T. J. Starkweather, and C. Bogart, "Genetic algorithms and neural networks: Optimizing connections and connectivity," *Parallel Computing,* vol. 14, no. 3, pp. 347–361, 1990.
2. T. J. Starkweather, D. Whitley, and K. Mathias, "Optimization using distributed genetic algorithms," in *Parallel Problem Solving from Nature,* H.-P. Schwefel and R. Männer, Eds. Berlin: Springer Verlag, 1990, pp. 176–185.
3. C. R. Houck, J. A. Joines, and M. G. Kay, "Comparison of genetic algorithms, random restart and two-opt switching for solving large location-allocation problems," *Computers & Operations Research,* vol. 23, no. 6, pp. 587–596, 1996.
4. J. Horn, *GAs (with sharing) in search, optimization and machine learning.* San Mateo, CA: Morgan Kaufmann, 1997.
5. D. Whitley, S. B. Rana, J. Dzubera, and K. E. Mathias, "Evaluating evolutionary algorithms," *Artificial Intelligence,* vol. 85, no. 1-2, pp. 245–276, 1996.
6. J. M. Fitzpatrick and J. J. Grefenstette, "Genetic algorithms in noisy environments," *Machine Learning,* vol. 3, pp. 101–120, 1988.
7. F. Glover, "Tabu search: A tutorial," *Interfaces,* vol. 20, no. 4, pp. 74–94, 1990.

The Shifting Balance Genetic Algorithm as More than Just Another Island Model GA

Mark Wineberg and Jun Chen

University of Guelph
Computer and Information Science
wineberg@cis.uoguelph.ca, jun@uoguelph.ca

Abstract. The Shifting Balance Genetic Algorithm (SBGA) is an extension of the Genetic Algorithm (GA) that was created to promote guided diversity to improve performance in highly multimodal environments. In this paper a new behavioral model for the SBGA is presented. Based on the model, various modifications of the SBGA are proposed: these include a mechanism for managing dynamic population sizes along with population restarts. The various mechanisms that make up the SBGA are compared and contrasted against each other and against other Island Model GA systems. It was found that the mechanisms that characterize the SBGA, such as a repulsive central force from one population on the others, could improve the behavior of multi-populational systems.

1 Introduction

While the GA has proven to be very effective in various problems, the GA still can get trapped at local optima, which is known in the GA literature as premature convergence. To a great degree overcoming local optima and lessening premature convergence can be seen as the problem of diversity maintenance. Many mechanisms are developed to help the GA to maintain diversity in the population such as Restricting mating practice [1][2], adaptive mutation rate or selection pressure [3], random immigrant [4], restart / re-initialization mechanism [5] and multiple population solution [6] [7].

It would be useful if the diversity can be added in a guided fashion; less blindly. Blindly increasing the diversity can cause the GA to experience random drift in the search area and consequently lose its search efficiency. An algorithm has already been designed with these issues in mind: the Shifting Balance Genetic Algorithm (SBGA) [16]. The SBGA is a multi-populational GA (similar to many parallel GAs), but with mechanisms introduced to promote directed diversity.

The SBGA when introduced was designed with user chosen parameters for both the number and size of the populations. These were chosen based on an initial model of how the SBGA behaved and was never investigated further. In this paper we propose a new analysis of the behavior of the SBGA and show that modifications inspired by the new analysis improve the SBGA's ability to handle multimodal environments. We then examine the various mechanisms in the SBGA, such as the repulsive central force from the core population that act on the colony populations,

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 318–329, 2004.

and see if they are effective when added to a general multi-populational system such as an Island Model GA with a bi-directional ring topology.

2 Diversity and Adaptive Population Size

Maintaining Diversity

There are different ways measuring diversity, e.g. *pair-wise distances diversity, informational entropy diversity.* Maintaining diversity usually can be done in two ways: either by decelerating the process of gene fixation or by reintroducing the diversity after it has been lost. Common diversity maintenance or enhancing techniques are increasing mutation, preventing inbreeding, decreasing selection pressure, restarting the populations, and the use of multiple populations.

One technique for introducing diversity is simply increasing mutation. Grefenstette [4] introduced the idea of *random immigrants* into the population. Here randomly generated individuals are inserted into the population to be a source of random crossover segments.

Inbreeding promotes diversity loss. Consequently, some restricted mating practices such as using species identity tag and inbreeding prohibitions are used [2]. These techniques can monitor individual gene makeup, fitness, or mating history.

Decreasing the selection pressure is also another way to help maintain diversity. There are many different ways to do this, the simplest one is the reduction of the slope used in linear rank selection.

Re-initialization of the population, in whole or in part according to some criteria or policy can reintroduce diversity after it has been lost. In practice the criteria used can include: cost bounds, time bounds, improvement probability bounds and convergence bounds, along with various combinations [5].

A multi-populational GA (also called the Island Model) localizes members into sub-populations, called islands, where each island can potentially follow a different search trajectory through the search space and develop in more or less isolated niches [9].

Adaptive Population Size

There has been very little work on adaptive population size schemes, despite the fact that the population size is a critical parameter in a GA and it largely determines the quality of the solution [10]. The limited amount of work done can be broken down into two groups: direct evolution of the population size parameter [14]; and mechanisms that indirectly affect the population size, which include GAVaPS [11], SOS [13] and EGA [12]. We will be using ideas from the last approach so we will go into further detail for that system.

Schilerkamp-Voose and Mühlenbein [12] use subpopulations competition that changes the sizes of subpopulation in their breeder genetic algorithm (BGA). There the total number of individuals is fixed, whereas the size of a single group varies. In every generation the subpopulations compete based on the fitness of the best of individual of the group. At regular generation intervals a quality criterion is used on each subpopulation; the group with best quality increases the size and all other groups are decreased. A lower limit is used for the size of each group, so no population is lost. At

a later generation within the interval, the best individual from each subpopulation emigrates to all populations and the best fitness of the groups becomes equal. We will borrow these ideas for the SBGA when we introduce subpopulation competition among colonies (a colony is a specialized subpopulation), although the mechanism will be slightly modified.

3 Shifting Balance Genetic Algorithm

Introduction
The Shifting Balance Genetic Algorithm was originally designed to use multiple populations to solve moving peak problems in dynamic environment; but it also can help GA as a function optimizer to overcome local optima or lessen the premature convergence in multimodal environments. Since this paper re-examines the behavior of the SBGA and improves on the algorithm, a brief overview of the system is presented below. For a more detailed explanation of the SBGA, see [15] [16].

Core Group and Colonies
SBGA is a multiple population GA similar to the island model. But unlike other island model GAs it does not divide the population to several demes with the same size. In SBGA the populations are classified into two categories: A large central population, called the core and a series of smaller, helping population called colonies.

The core is responsible for exploring the area of the search space that seems most promising and performing exploitation through receiving the elite immigrants from the colonies, while the colonies explore the areas of the search space where the core population does not yet have a presence. The core has a large population size so that it has an increased ability to do search or perform hill climbing since the more members the GA has, the more children it can produce and hence generate more trials in that area.

The colony is set to have a smaller population size in order for each colony's gene make-up to more easily change, however this also can cause a reduced search power.

Selecting for Distance from the Core
For the SBGA to perform as desired, a mechanism is needed that allows a population to relocate to other places in the search space, thus escaping from a local maximum. The SBGA has a mechanism for determining whether an individual from the colony is searching in the same area as the core, which is implemented as a distance to the core: it is the average Hamming distance between a colony member and each member of the core. The distance is used as a fitness function for the members in the colony when the colony becomes too close to the core. The members will then be selected for reproduction, not only according to their objective function value but also according to their distance from the core. They will *evolve* into a new area of the search space.

Since we intend the colony to follow the objective function's landscape, even as it moves away from the core, we are faced with a bi-objective optimization task. We handle this bi-objective optimization using a modification of the VEGA multi-objective optimization system along with dynamically controlled partitioning:

The population for the next generation is split into two sections. The first section is selected under the objective function, and the second under the distance to the core. When filling a subsection, the parents are selected from the previous generation's entire population. However, the fitness function used during the selection is the one associated with the sub-population being filled.

In order to implement the above bi-objective approach to moving the colony away from the core, we need to determine the portion (κ) of the population governed by the distant to the core versus the objective function. This is dynamically calculated (instead of using fixed percentages as with VEGA) using a formula called the *percentage similarity,* see [16] for details.

Migration from Colony to Core

In the SBGA, the colony sends members, called *migrants,* back to the core. During migration the colony may send all of its members to the core or only some portion thereof. The migrants are chosen from the elite members of the colony (25% of the colony has been used).

Since migration of the colony members disrupts the core group, time is given for the colony to evolve potentially useful members. The number of generations between migrations is called the *migration interval.* To reduce the pressure on the core even more, only a few colonies are allowed to send migrants in the same generation.

Just like all multiple-population based GAs, the SBGA needs a method to integrate the migrants arriving from the colony into the core's population. The host population is temporarily enlarged by the migrants and then pared down again to its normal size after reproduction. In other words, selection for reproduction is performed on the extended population, host members plus immigrants, but the new population will only be the size of the original population.

4 Analysis of the Behavior of SBGA

Preliminary Experiments and Questions

In our description of the SBGA we assumed that the core population would behave as a regular GA enhanced by novel solutions found by the colonies. The large population of the core was assumed to be able to do the exploitation of an area with the small, more easily changed colonies providing the exploration. But whether the SBGA does search in this way was never really proved. One of the ways to help us to understand the SBGA's behavior is to monitor each colony's portion governed by the distant to the core, i.e. its κ value.

We should observe the following: at first the core should repel all the colonies away, driving most of their κ value to nearly 0. Some time later if the current best colony is even better than core, immigrants entering the core will force the core to overlap with the colony, thereby forcing the colony's κ higher. Shortly after the repulsive force acted on the colony by the core will cause the colony to move away from the core and the κ value will drop. Maybe several generations later another colony becomes the best and the process will repeat.

Experiments were performed to see if this behavior actually happened. The κ values of all colonies were recorded during various runs of the SBGA using the same fit-

ness function used to test the original SBGA system. Furthermore, a modified SBGA without the migration from the colonies to the core was used as a comparative system to the SBGA. The fitness function is the 6^{th} dimensional F8F2 minimization problem (see [17]). The definition of F8F2 function is

$$F8F2(\mathbf{x}) = \sum_{i=0}^{n-1}\left(1 + \frac{\left(100\left(x_i^2 - x_{i+1}\right) + \left(1 - x_i\right)^2\right)}{4000} - \cos\left(100\left(x_i^2 - x_{i+1}\right) + \left(1 - x_i\right)^2\right)\right). \quad (1)$$

The parameter settings used for the experiment is summarized in Table 1. Each experiment lasted 300 generations and was repeated 60 times.

In these 60 experiments we find the trend of the κ value is very similar. Fig. 1 shows a typical result.

Table 1. SBGA parameter setting

Population Size		**Selection**	
Core	1000	Linear rank selection	
Colony	100	Low=0.0 (high=2.0) Elitism	
Core		**Colonies**	
Mutation rate	0.006 bits/locus	Mutation rate	0.01 bits/locus
Prob. of crossover	0.7	Prob. of crossover	0.9
One point crossover for both Core and Colony		Migration size	25% colony size (elite)
		Migration interval to core	6
		Number of Colonies	10

In Fig. 1a where no immigration occurs, the colonies will soon be repelled away to other areas of the search space and no longer overlap the core. After 15-40 generations all the k values for colonies are almost 0, which continued throughout the rest of the run (not shown). This means the genetic composition of each colony becomes completely dissimilar to that of the core. Now look at Fig. 1b: when the colonies send immigrants to core, the κ values remain high over the entire course of the experiment for most of the colonies. We do not see individual colonies rise and decay sequentially as the core becomes interested in that colony's area. Rather the core seems to overlap with almost all the colonies simultaneously (this occur throughout the run – only the first half of the run is shown). This seems to mean that the core is trying to repel all of the colonies away at the same time, although the repulsive force is not constant for each of them.

Based on the result of the preliminary experiments, it seems the SBGA does not always work as originally expected. Perhaps there is a better explanation search behavior of the SBGA that allows it to better overcome local optima. In the following section we present a new hypothesis of the core's role in search.

New Analysis of the Core's Role for Search

In the above experiment the core overlaps with all of them throughout the experiment. One possible explanation is that, because of the core's large population size and the fact that the immigrants are the elite members; the immigrants from all colonies are entering the core and staying around long enough to create multiple niches in the core.

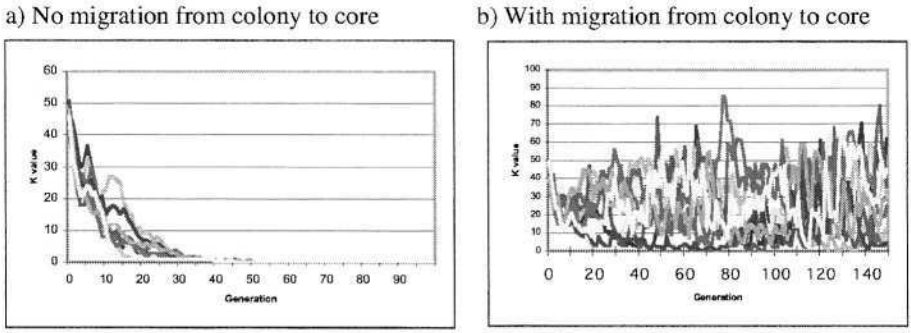


Fig. 1. Overlap between the Core and Colonies (dynamic κ)

This causes the core to become disorganized, which fragments core's gene pool and confuses the core on where in gene space to go. If this is the case, the core can be seen as a gathering center for the different immigrants from the colonies. If most of colonies continue to find better solutions and contributes them to the core, the core actually will not be stable and cannot focus on searching in just one area of the gene space.

However, the original experiments SBGA really do work better than GA when test on some multimodal environments [15][16]. What makes it better? One reason probably comes from simply using multi-populations. Another reason may be that the main power does not come from the core moving to where some colony is lying in the search space and just repelling it away and then exploring the area by itself; rather the mechanism kicks in at the beginning when the core tries to repel all the colonies away. This kind of repulsive force can make the colonies more active and search more areas, thus increasing the probability that one of them finds a better local optimum or even the global optimum.

From this point of view, the key role of the core is not seen as that of an exploiter who mines the area found by the current best colony, but rather more like the center of a repulsive force or possibly even a memory center who receives immigrants from all the colonies and tell them "you have already looked here, you can search some other place". The core may even be better than a simple memory since it improves on what was stored by evolving it.

As a natural consequence of this new explanation, it becomes feasible to redesign the SBGA and improve its search ability and robustness. In next section we will consider how to redesign the SBGA in light of our new understanding.

5 Redesigning the SBGA

Introduction

Through the analysis from the previous chapter we questioned the explanation of the behavior of the original SBGA and supplied some alternative explanations on why the SBGA is better than simple GA, especially when facing multimodal fitness land-

scapes. Based on this analysis, we redesign the SBGA along three lines, which are presented below.

Small Core or Big Core

According to the analysis in section 4 the core cannot be easily shifted into the area of the best colony; furthermore if it did manage to cover the new territory, it would not efficiently search it. Therefore, a big core size is probably a waste of resource. As a result in the modified SBGA we decided to shrink the core size and give more search labor to do exploration by increasing the colony number.

However, we must not choose a core size that is too small, for it should be large enough to repel all of the colonies simultaneously. Consequently we decided to reduce the core from 1000 members to 200 members.

Dynamically Modifying the Colonies' Population Sizes

If the core's population size is decreased, then the colonies population must increase. The new analysis of the SBGA's behavior leads us to the conclusion that it is the colonies and not the core that is responsible for exploiting a region in gene space. This suggests that we give the colony with the current highest performance a larger population size, thus giving it the greater search power. Consequently, we introduce competition among the colonies directly: a better colony will be given more members so the population size of the colony is adapted according to their performance.

In section 2 we saw that Schilerkamp-Voose and Mühlenbein added subpopulation competition for population size to their breeder genetic algorithm [12]. We will add that mechanism, with modifications, to the SBGA to determine which colony should have the greater search power.

Each colony will be evaluated over some generation interval (e.g. 5 generations). The colony with the best performance will have their population size increased, and likewise all other colonies will have their population size decreased.

The first step is the choice of which population is to have its size increased. This part of the algorithm is where we differ from the BGA the most. We removed their complex mechanism for "averaging" the sub-population's performance over the generations. We also removed the evening-out of the best of a population through broadcasting each population's elite member to all other populations. In their place, two criteria are used to select which population should have its size increased:

1. *Rank value*
   - Colonies with a higher fitness of its best member get a higher priority of consideration
2. *Colony stagnation*
   - We consider a colony stagnant if there is no improvement in last 10 generations

Expanding on the second criterion, the chosen colony needs to show it is not stagnant before it can be chosen. This means each colony will be monitored to see when the colony last improved. The improvement measure is based on the best fitness of individual in the population, not the average fitness of the population. So the generation gap between the current generation and the generation when the best fitness was found is calculated, and is called the *stagnation interval.* If this gap is bigger than a user defined parameter or threshold (we used the value of 10 to match the BGA *quality criterion,* which our mechanism replaces) the colony will not be chosen and the algorithm will automatically turn to next one on the fitness list. If all the colonies are

not eligible, the algorithm will randomly choose one. If there is a tie in the ranks for both criteria, a colony is randomly chosen from among the ties.

After one of the colonies is selected, all other colonies give a fixed percentage of their members (Schilerkamp-Voose and Mühlenbein used the value of 1/8 for their experiments) to the best non-stagnant colony. This is called the *gain criterion.*

Restarting Dead Colonies

Since the colonies is now seen as the focus of the both exploration and exploitation, colonies that are not productive are more deleterious than previously thought and should be replaced. When this happens, these colonies will restart using members from other colonies as seeds.

Three criteria are used for choosing the colony to restart

1. Choose the lowest ranking colony.
2. Choose a colony that has seen no improvements over the last few generations.
   - Preliminary experiments have shown that 10 generations is an effective choice.
3. Choose a colony that has a diversity that is lower than some threshold.
   - Preliminary experiments have shown that 0.15 is an effective choice.

Diversity is computed using the average information entropy at a locus across the population.

After the stagnant colony is chosen, we will restart it. The whole population will be re-initialized based on a seed individual from the colony that is having its population size increased. This seed individual, which will be the best individual from the donating population, will be used to create all of the individuals in the new population by performing bit mutation with a given probability (e,g., 30%). So the new population can be seen as a hyper-sphere around the seed in genotype space, with the radius of the hyper-sphere is decided by the probability of mutation.

Table 2. Parameter setting for two modified system and simple GA

Population Size	**Selection**
GA 2000, Core 200, Colony 100	Linear rank selection (max = 2), Elitism
GA and Core	**Colonies**
Mutation rate = 0.006 bits/locus	Mutation rate = 0.01 bits/locus, Prob. of xover = 0.9
Prob. of crossover = 0.75	Number of Colonies 18
Migration	**Special for SBGA with competition**
Size to core = 25 colony members	Initial Colony size 10 Competition begins 30
Size to colony = 10% colony	Min Colony size 4 Lowest diversity 0.15
Migration interval: to core = 6, to colony = 20	Evaluation interval Stagnation interval 10

6 Experimental Results and Analysis

Can the Repulsive Central Force Help an Island Model GA?

In this section, we want to test whether the repulsive central force mechanism is useful. Without this mechanism the SBGA can just be considered a rather limited variant

Table 3. GA systems with 'repulsive force' VS systems without 'repulsive force'

System	Lower Bound: 29% percentile	Median	Upper Bound: 71% percentile
SBGA	0.05133	0.08921	0.11012
SBGA-NoRepel	0.08926	0.11906	0.13722
SBGA-Com	0.01003	0.01378	0.03319
SBGA-Com-NoRepel	0.05220	0.08905	0.11877
SBGA-Bi	0.02093	0.04066	0.05960
SBGA-Bi-NoRepel	0.05982	0.08984	0.13231

Table 4. Bi-objective function VS Random Immigrant for Colonies

System	Lower Bound: 29% percentile	Median	Upper Bound: 71% percentile
SBGA	0.02228	0.05994	0.08895
Random-Imm	0.06009	0.11147	0.13092

of an Island Model GA. We therefore compare the original SBGA and an SBGA with a bidirectional ring added to it (to be more like a traditional Island Model GA) against those same systems with the repulsive central force turned off. To complete our comparisons we will also test the SBGA with the new colony competition system installed, with and without the central repulsive force.

The test function used is F8F2 on 6 dimensions. Each experiment lasted 300 generations and was repeated 60 times. The parameter setting of the original SBGA can be found in Table 1. The configuration for two new modified SBGA systems is given in Table 2.

Table 3 gives the results[1] for three systems and their counterparts[2]. From these results[3], we can see all three systems that use the bi-objective function in their colonies produce better value on average than their counterpart. According the experiment data for this problem we found that in all cases using the repelling central force improved the behavior of the system with statistical significance.

Finally we also see that the subpopulation competition is better than all of the other systems, showing that the new analysis did produce a system that improves the SBGA's behavior overall.

---

[1] We record the median instead of the mean because the results are not normally distributed.

[2] We are testing for statistical significance of the comparison between the 3 systems. If the upper and lower bounds don't cross, neither system is better than the other. This test is not as sensitive as using ANOVA with pair-wise Student's T tests. These were done (with a confidence level of 95%) and they confirm the results presented here.

[3] The upper and lower percentiles are obtained from the Thompson - Savur method for finding confidence intervals around a median, using a Bonforonni correction (we performed 97 independent comparisons in all our experiments; we therefore always divide our α by 97).

Table 5. Big Core SBGA VS Small Core SBGA

System	Dim	Lower Bound: 29% percentile	Median	Upper Bound: 71% percentile
Core200	4	0.00995	0.01018	0.02060
Core1000	4	0.00990	0.01286	0.02969
Core200	6	0.02228	0.05994	0.08895
Core1000	6	0.05133	0.08925	0.11012
Core200	8	0.13113	0.15580	0.21074
Core1000	8	0.15237	0.20850	0.24143

The Repelling Central Force vs. Random Immigrants

Perhaps the SBGA is just effective entirely because of increased diversity. We could try comparing against systems with high mutation rate, but as discussed in section 2, a better technique is to introduce diversity through a Random Immigrant system [4], where in every generation the population is partly replaced by randomly generated individuals. The Random Immigrant mechanism has the ability to constantly keep the diversity of the population high; but unfortunately it is a completely blind process and so can be very disruptive. The repelling force more gently introduces the needed diversity, but if the population is almost converged it loses its power. Comparing these two mechanisms in the same F8F2 (6 dimensions) function produces the following results in table 4.

Here we are using the original SBGA with the same configuration as in Table 1 except that the core size changed to 200 and 18 colonies are used (total population size is the same 2000) as was originally formulated. For the **Random Immigrant** system, every thing is the same but the bi-objective function of the colonies is replaced by the random immigrant mechanism (20% of population is replaced by randomly created individual in every generation). From Table 4 we see that the SBGA with bi-objective function is statistically better the SBGA with Random Immigrant.

Smaller Core with More Colonies vs. Big Core with fewer Colonies

In this experiment, we will see if various size core and colony combinations alone can have improved results on the F8F2 function in 4, 6 and 8 dimensions. The parameter setting for base SBGA is the same as in Table 2 but the core size is larger (**1000** vs. ***200***) and colony number is smaller (**10** vs. ***18***); however, the total population size is always fixed to 2000. Each experiment lasted 300 generations and was repeated 60 times.

The results seen in Table 5 can be summarized as follows[4]:

4 dim: Small Core SBGA | Big Core SBGA
6 dim: Small Core SBGA > Big Core SBGA
8 dim: Small Core SBGA > Big Core SBGA

[4] Here 'A>B' means system A is better than system B with statistical significance and '**A | B**' is means that system A is statistically indistinguishable from system B.

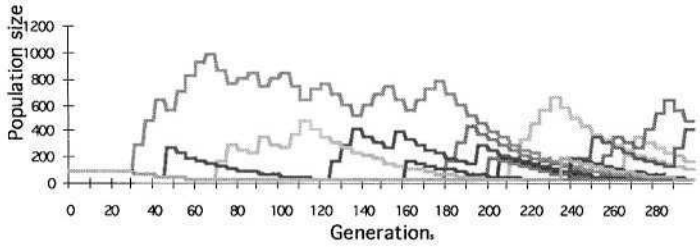


Fig. 2. Dynamical population size for each colony

We can see that on the F8F2 problem, a smaller core with more colonies works equally well or better than the SBGA with a large core and fewer colonies function. Furthermore the results get stronger as dimensionality increases. This means that as the fitness landscape becomes larger, more local optimal need to be overcome in order to get better result, and so the advantage accrued by the many colonies being spurred on by a small core becomes more obvious.

Dynamical Population Size on Graph

We have discussed the mechanism that allowed the SBGA colonies to compete for members and survival. However, that analysis was merely a thought experiment; how the system behaves in reality is still unknown. It will be very educational to see how each colony's population size changes every generation. Consequently, we choose a typical result of the 6 dimensional F8F2 experiment from the previous section and plot all colony population sizes by generation. Fig. 2 gives the result.

The mechanism used for colony competition can be summarized as follows: at some generation interval (5 generations here) only one colony is chosen as the winner to increase the population size and all others will decrease, except for those that have reached their minimal size. But if the winner wants to keep its position, it must also show improvement.

From Fig. 2, we can see that at the early stage of the competition (30 to 100 generation) only 3 of the colonies really get a chance to increase their population size; the rest of them quickly lose their members and reduce to the minimum size. However after generation 120, more colonies get a chance to increase their population size, which can be seen from the fact that more peaks appear on the diagram. One possible reason for this is that, as the colony evolved, many of them gradually lose diversity and also become stagnant. At this time they are qualified for restart. After a colony is restarted using a seed from the current colony winner they have more of a chance to win members.

7 Conclusions

Through the experimentation and analysis of the F8F2 problem, we found that the mechanisms that characterize the SBGA, such as the core's repulsive central force improve multi-populations GAs such as the bi-directional ring Island Model GA. We

also re-analyzed the dynamic behavior of the SBGA, verified various aspects of it, and proposed modifications: the addition of dynamic population sizes and population restarts. These additions to the SBGA were also seen to be effective and improved the SBGA's performance.

References

1. Eshelman, L.J., Schaffer J.D. (1991). Preventing Premature Convergence in Genetic Algorithms by Preventing Incest. 4th ICGA, pp.115-112.
2. Craighurst, R., Martin, W. (1995) Enhancing GA Performance through Crossover Prohibitions Based on Ancestry. 6th ICGA, pp. 130-135
3. T, Bäck, (1992). The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm. PPSN 2.
4. J..J. Grefenstette. (1992) GAs for changing environments. PPSN 2, pp.137-144.
5. Eshelman, L.J., (1991) The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic recombination. FOGA pp. 265-283.
6. E. Cantu-Paz. (1998) A survey of parallel genetic algoritms. Calculateurs paralleles, Reseaux et Systems Repartis, 10(2): 141-171.
7. E. Cantu-Paz (2000). *Efficient and Accurate Parallel Genetic Algorithms.* Kluwer.
8. D. Whitley, S. Rana and R. B. Heckendorn (1998) The Island Model Genetic Algorithm: On Separability, Population Size and Convergence, Proceedings of the AISB Workshop on Evolutionary Computation.
9. Munetomo, M., Takai, Y., & Sato, Y. (1993). An efficient migration scheme for subpopulation-based asynchronously parallel genetic algorithms. 5th ICGA. pg. 649.
10. Harik, G., Cantú-Paz, e., Goldberg, D., & Miller, B. L. (1997). The gambler's ruin problem, genetic algorithms, and the sizing of populations. In the 4th ICEC.. pp. 7-12
11. Michalewicz, Z. (1994) *Genetic Algorithms + Data Structures = Evolution Programs* (2nd edition). Springer-Verlag.
12. Dirk Schlierkamp-Voosen and Heinz Mühlenbein (1994). Strategy adaptation by competing subpopulations. PPSN 3, pages 199-208.
13. Jürgen Branke. (2002). Evolutionary Optimization In Dynamic Environments University of Karlsruhe, Germany. Kluwer Academic Publishers
14. Shisanu Tongchimaand and Prabhas Chongstitvatana. (2002). Parallel Genetic Algorithm with Parameter control.
15. M. Wineberg and F. Oppacher (2000). Enhancing the GA's Ability to Cope with Dynamic Environments. In GECCO-2000. pp. 3-10.
16. M. Wineberg (2000). Improving the behavior of the genetic algorithm in a dynamic environment. PHD Dissertation, Carleton University, Canada.
17. Whitley, D., K. Mathias, et al. (1996). "Evaluating Evolutionary Algorithms". Artificial Intelligence 85: 245-276.

Bistability of the Needle Function in the Presence of Truncation Selection

Alden Wright[1] and Greg Cripe[2]

[1] University of Montana, Missoula MT USA
[2] Spokane Falls Community College, Spokane WA USA

Abstract. It is possible for a GA to have two stable fixed points on a single-peak fitness landscape. These can correspond to meta-stable finite populations. This phenomenon is called *bistability,* and is only known to happen in the presence of recombination, selection, and mutation. This paper models the bistability phenomenon using an infinite population model of a GA based on gene pool recombination. Fixed points and their stability are explicitly calculated. This is possible since the infinite population model of the gene pool GA is much more tractable than the infinite population model for the standard simple GA. For the needle-in-the-haystack fitness function, the fixed point equations reduce to a single variable polynomial equation, and stability of fixed points can be determined from the derivative of the single variable equation.

1 Introduction

Intuitively, one would expect that a GA could have only a single stable fixed point on a single-peak fitness function. However, in this paper we show that a mutation/crossover/selection GA may have two stable fixed points on a single-peak fitness landscape. We call this phenomenon *bistability.*

In practical terms, this means that when the GA is started with a random population, the GA can stagnate without making any progress in the direction of the fitness peak. However, when the GA with the same parameters is started nearer to the fitness peak, the GA will move towards the peak. This behavior is described in more detail in section 5.

In our model, there cannot be bistability without both mutation and recombination. For the more general Vose model, there cannot be bistability without recombination [1] and it is conjectured that there cannot be bistability without mutation. Bistability only occurs for some settings of the parameters where there is the appropriate balance between mutation, recombination, and selection. For our model, the minimum string length for bistability is 4. As the string length increases, the range of mutation rates where bistability occurs increases rapidly.

Bistability is not an isolated phenomenon. While we use gene pool recombination and a needle-in-the-haystack fitness for our model, neither are necessary for bistability. We show that bistability occurs with uniform and one-point

[1] this follows from the Peron-Froebenius theorem

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 330–342, 2004.

crossover, and Wright, Rowe, Poli and Stephens [WRPS03] show that bistability occurs with fitness functions that show a more gradual increase in fitness as one gets close to the fitness peak This paper extends on [WRPS03] where bistability was analyzed in the case of proportional selection. This paper treats the case of truncation selection.

Bistability was first discovered and investigated by [BBN96] in the context of viral quasispecies and the AIDS virus. Other papers on bistability include [OH97] and [WRN02]. The last citation has a more complete review of the literature in this area.

Vose [Vos99] and others have developed an elegant dynamical system model for GAs. This model is commonly referred to as the "infinite population model" of a GA. Vose has proved elegant theorems relating fixed points to finite population GA behavior when the population is large. However, except in special cases, only numerical methods can be used to actually find the fixed points. Due to the large dimensionality of the space, numerical methods can be used only for relatively small string lengths. Further, it is difficult to use the applicable numerical methods to achieve understanding of the processes involved.

The model used in this paper makes a "linkage equilibrium" assumption that makes the infinite population model more tractable. Linkage equilibrium has been widely used as a simplifying assumption in approximate models of GAs. These include [SI99] and [PBR01].

We show that for the needle-in-the-haystack fitness functions, fixed points can be found by solving a single variable polynomial equation, and the stability of these fixed points can be determined from this single variable equation. Thus, we can rigorously determine the number of fixed points and their stability, something that has not been possible for the Vose model when there is nontrivial selection, mutation, and recombination. We can determine and plot the ranges of parameters where bistability occurs. The price that we pay for the linkage equilibrium assumption is that we no longer have an exact infinite population model of a standard GA that uses two-parent crossover. Instead, we have an exact model of a GA that uses a special form of recombination called "gene pool recombination". This GA will be described in more detail in section 2

Gene pool recombination is an alternative recombination method for GAs. An individual created by gene pool recombination is chosen from a probability distribution determined directly from the whole population rather than from two parents. Geiringer's theorem [Gei44] shows that gene pool recombination in the infinite population model can be viewed as the limit of repeated applications of two-parent recombination (without selection). For the infinite population model, gene pool recombination takes the population to linkage equilibrium in one step.

Gene pool recombination was proposed in [Sys93]. [MM01], [MM00] have investigated the UMDA (univariate marginal distribution algorithm) which is a GA that uses gene pool recombination and some kind of selection (proportional, tournament, truncation, or Boltzmann) and no mutation. They have experimentally verified that UMDA can successfully optimize a wide variety of fitness functions but is misled by deceptive problems. Further, gene pool recombina-

tion is used in population-based incremental learning (PBIL) [BC95], a machine learning technique.

Our empirical results in section 5 and the empirical results given in [WRPS03] suggest that the gene pool GA model is a reasonable approximation to the two-parent recombination GA for the classes of fitness functions investigated in this paper. Thus, the gene pool GA model can be viewed as an approximate model of the two-parent recombination GA.

2 The Gene Pool GA

In this section we give a more precise description of the gene pool GA. Our model is an exact model of this GA in the limit as the population size goes to infinity.

We assume a binary string representation. The string length is ℓ.

Gene pool recombination uses a sampling process to go from one population (the current population) to another (the new population). Each individual of the new population is created independently of the others, and in fact, each bit of each individual is created independently of the other bits.

The first step in the sampling process is to calculate the relative frequency of a 0 bit at each locus (string position) in the current population. For a given locus, this frequency is just the relative frequency of an order-1 schema whose only defined position is a 0 at this locus. For an individual of the new population, the probability of a 0 at this locus is the same as the relative frequency of a 0 at the same locus of the current population.

Note that expected result of gene pool recombination depends only on the relative frequencies of the order-1 schemata. Thus, the infinite population model can be defined in terms of these frequencies.

Note that since bits of individuals in the new population are chosen independently, there is no expected correlation between the bits at different loci. This is exactly the definition of linkage equilibrium, so in the infinite population limit, gene pool recombination produces a linkage equilibrium population.

Then the steps of the gene pool GA used in this paper are as follows:

1. Choose a random population.
2. Apply gene pool recombination.
3. Apply truncation selection.
4. Apply mutation.
5. Return to step 2 if termination criteria is not met.

3 The Infinite Population Model

Our infinite population model is represented in the Walsh basis. (See the appendix for the definition of the Walsh transform.) Since we only need to represent information equivalent to the order-1 schema frequencies, this is not as complicated as it seems. The model uses the order-1 Walsh coefficients, and these

coefficients are expressed simply in terms of the order-1 schema frequencies. This is explained below.

Let Ω be the search space of all length-ℓ binary strings. If $j \in \Omega$ is a binary string, $\#j$ represents the number of ones in j. Let $\mathcal{L}$ denote the set of strings j with $\#j = 1$.

A population is represented by a vector indexed over Ω. Thus, if x is a population, x_j is the relative frequency of string $j \in \Omega$.

For $k \in \mathcal{L}$, let $x_0^{(k)}$ and $x_1^{(k)}$ denote the schema frequencies of the order-1 schemata whose only defined locus is the locus of the 1 bit in k. (The string k can be thought of as a mask. The locus corresponding to k is the locus masked by k.) Thus, if $k = 00010$, then $x_0^{(k)}$ is the frequency of the schema $***0*$ and $x_1^{(k)}$ is the frequency of the schema $***1*$.

Let $\widehat{x}_k$ denote the kth coefficient of the Walsh transform of x. It can be shown [WRPS03] that for any population vector x and any $k \in \mathcal{L}$,

$$\widehat{x}_k = 2x_0^{(k)} - 1 \quad \text{and} \quad \widehat{x}_k = 1 - 2x_1^{(k)} \ . \tag{1}$$

Note that $x_0^{(k)} + x_1^{(k)} = 1$ by definition. For this purposes of this paper, these formulas can be taken as the definition of the order-1 Walsh coefficient $\widehat{x}_k$.

The value of $\widehat{x}_k$ ranges from – 1 when the frequency of a 0 in the locus masked by k is 0 to +1 when the frequency of a 0 in this locus is 1.

Now we can look at modeling the steps of the algorithm given in section 2. There are two important observations to make.

First, the result of the gene pool recombination step depends only on the frequencies of the order-1 schemata and hence of the order-1 Walsh coefficients. Thus, our model only needs to keep track of the order-1 Walsh coefficients.

Second, the expected frequencies of the order-1 schemata do not change in the gene pool recombination step, and thus the infinite population model of gene pool recombination is trivial: the order-1 Walsh coefficients remain unchanged.

So it remains to model mutation and selection.

The Walsh basis formula for mutation is very simple. If the mutation rate is μ, the effect of mutation in the infinite population model is to multiply $\widehat{x}_k$ by $1 - 2\mu$ [WRPS03]. One can see that the effect of mutation is to move the order-1 schema frequencies towards 1/2. In fact, if the mutation rate is 1/2, then mutation makes $\widehat{x}_k$ to be zero, which corresponds to schema frequencies of 1/2.

3.1 The Needle-in-the-Haystack Fitness and Truncation Selection

The needle-in-the-haystack (NEEDLE) fitness function assigns a fitness of 1 to all strings except the all-zeros string. The all-zeros string has a fitness greater than 1. (The exact value is unimportant since we are using truncation selection.)

In truncation selection, a fraction T of the population is kept and the remainder is discarded. For the NEEDLE fitness function, we only need to determine how the frequency of the all-zeros string is increased by selection. In our infinite population model of truncation selection, we assume that the frequency of all other strings is decreased by the same multiplicative factor.

Thus, let x denote the population before selection, and let x_0 denote the frequency of the all-zeros string in this population. Let y denote the population after selection. Then it is not hard to see that $y_0 = \min\left(1, \frac{x_0}{T}\right)$ where T is the truncation fraction. For any other string $j \neq 0$, $y_j = x_j \left(\frac{1-y_0}{1-x_0}\right)$.

Recall that $x_1^{(k)}$ is the frequency of the order-1 schema whose value is 1 in the locus masked by k. Since the all-zeros string is not a member of this schema, the effect of selection on the frequency of this schema is the same as the effect on any nonzero string. In other words, $y_1^{(k)} = x_1^{(k)} \left(\frac{1-y_0}{1-x_0}\right)$.

This formula can be transformed into a formula on the Walsh coefficients using the second equation of (1).

$$\widehat{y}_k = 1 - (1 - \widehat{x}_k)\left(\frac{1-y_0}{1-x_0}\right)$$

Let G denote the mapping that represents the complete model. Thus, if x is the population at the beginning of step 2 of the algorithm, then $G(x)$ is the expected next generation population at the same point in the algorithm. We can get a formula for G by multiplying by $1 - 2\mu$ to include the effect of mutation.

$$\widehat{G(x)}_k = (1-2\mu)\left(1 - (1 - \widehat{x}_k)\left(\frac{1-y_0}{1-x_0}\right)\right)$$

We are not done since this formula still includes the standard basis quantities x_0 and y_0, and we want a formula in terms of the order-1 Walsh coefficients. The key to eliminating these quantities is to note that selection occurs right after gene pool recombination in the algorithm, and gene pool recombination takes a population to linkage equilibrium. Thus we can assume that the population x (that selection is applied to) is in linkage equilibrium.

In [WRPS03], it is shown that

$$x_0 = 2^{-\ell} \prod_{j \in \mathcal{L}} (1 + \widehat{x}_j) \ . \tag{2}$$

There are two cases. The first case is when $x_0 \leq T$. In this case, $y_0 = \frac{x_0}{T}$, and some algebraic manipulation shows the first case of the formula below. The second case is when $x_0 > T$, and then $y_0 = 1$. This implies the second case of the formula below.

$$\widehat{G(x)}_k = \begin{cases} (1-2\mu)\left(1 - (1 - \widehat{x}_k)\left(\frac{T - \prod_{j \in \mathcal{L}}(1+\widehat{x}_j)/2}{T(1 - \prod_{j \in \mathcal{L}}(1+\widehat{x}_j)/2)}\right)\right) & \text{if } x_0 \leq T \\ (1-2\mu) & \text{if } x_0 > T \end{cases} \tag{3}$$

We can show that if $x_0 \geq T$, then $\widehat{x}_k = 1 - 2\mu$ is a fixed point since

$$\begin{aligned} x_0 &= 2^{-\ell} \prod_{j \in \mathcal{L}} (1 + \widehat{x}_j) \\ &= 2^{-\ell} \prod_{j \in \mathcal{L}} (1 + 1 - 2\mu) \\ &= (1 - \mu)^{\ell} \\ &> T \ . \end{aligned}$$

In the case where $x_0 \leq T$, formula (3) leads to the fixed point equations:

$$\widehat{x}_k = \frac{(1-2\mu)\left(1 - \left(\frac{T - \prod_{j \in \mathcal{L}} (1+\widehat{x}_j)/2}{T(1 - \prod_{j \in \mathcal{L}} (1+\widehat{x}_j)/2)}\right)\right)}{1 - (1-2\mu)\left(\frac{T - \prod_{j \in \mathcal{L}} (1+\widehat{x}_j)/2}{T(1 - \prod_{j \in \mathcal{L}} (1+\widehat{x}_j)/2)}\right)}$$

The right hand side of this equation is the same for all k. Therefore, if $\widehat{x}$ is a fixed point of G, then $\widehat{x}$ is symmetric in the sense that all $\widehat{x}_k$ for $k \in \mathcal{L}$ are equal.

If the GA model is started with a symmetric population, symmetry will be maintained. (In a symmetric population, all of the order-1 schema frequencies will be equal.) In this case, the recurrence equation can be written in terms of the variable w,

$$\widehat{G(w)} = \begin{cases} (1-2\mu) & \text{if } T < 2^{-\ell}(1+w)^{\ell} \ ; \\ (1-2\mu)\left(1 - (1-w)\frac{T - 2^{-\ell}(1+w)^{\ell}}{T(1 - 2^{-\ell}(1+w)^{\ell})}\right) & \text{otherwise.} \end{cases}$$

The fixed points occur when $\widehat{G(w)} = w$, or equivalently when equation (4) below holds.

Theorem 1. *The fixed points of the infinite population model of the gene pool GA algorithm are the solutions to the variable polynomial equation:*

$$w = \begin{cases} (1-2\mu) & \textit{if } T < 2^{-\ell}(1+w)^{\ell} \ ; \\ (1-2\mu)\left(1 - (1-w)\frac{T - 2^{-\ell}(1+w)^{\ell}}{T(1 - 2^{-\ell}(1+w)^{\ell})}\right) & \textit{otherwise.} \end{cases} \tag{4}$$

As we will see, understanding the solutions to this equation is far easier than understanding the solutions to the system of polynomial equations that come from a more general model.

As an example we solve (4) numerically using ℓ=8, μ=.1, and T=.4. We find $w = .02869$ and $w = .7222$. A third equilibrium occurs at $w = 1 - 2\mu = .8$. A graph of $G(w) = w$ is shown in Fig. 1. We see three fixed points; the first and third fixed points are stable and the middle fixed point is unstable relative to symmetric populations. However, stability in the figure does not necessarily imply stability in the space of all (possibly non-symmetric) populations.

We are interested in the stability of the fixed points in the space of all populations. It is well known for a discrete model that if all of the eigenvalues of

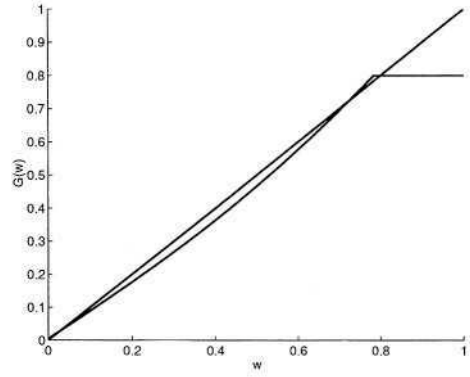


Fig. 1. Graph of $\widehat{G(w)}$ and 45° line for $\mu = .1$, $\ell = 8$ and $T = .4$.

dg_x have modulus less than 1, then x is an asymptotically stable fixed point of g. The following theorem found in Cripe [Cri03] shows the eigenvalue of $d\widehat{G}$ is equal to the derivative of the single-variable function $\widehat{G}$ defined in (4). Therefore the fixed points can not only be found by solving a single-variable polynomial but additionally their stability can be determined by taking a single variable derivative.

Theorem 2. *At a symmetric point $\widehat{x}$ where $\widehat{x}_k = w$ for all $k \in \mathcal{L}$, The largest modulus eigenvalue of $G_{\widehat{x}}$ is equal to $\frac{dG}{dw}$.*

Since the largest modulus eigenvalue of $d\widehat{G(x)}$ is equal to the derivative of the single variable function G, the stability of the fixed points in the cube $[-1,1]^\ell$ is the same as the stability of the fixed points in the one variable space of symmetric populations.

We have shown that we can find the fixed points of the model by solving a single variable polynomial of degree $\ell + 1$ and furthermore, the stability of the fixed points can be determined from this equation.

4 Explorations of Parameter Space

In this section we analyze the fixed points for our model to determine the parameter settings where bistability occurs. Due to space constraints the proofs of lemmas are not given when they are straightforward. Proofs are in [Cri03].

We begin to explore the parameter space by finding a relationship between T and w when w is a fixed point. Solving (4) for T, in the case $T > 2^{-\ell}(1+w)^\ell$,we find

$$T(w) = \frac{-(1-2\mu)(1-w)}{(w-1+2\mu-\frac{2\mu w}{x_0})} \quad . \tag{5}$$

That is, we define $T(w)$ to be the value of T for which w is a fixed point. Lemmas 1 and 2 show that the fixed points can occur in the region $0 < w < 1 - 2\mu$.

Lemma 1. $x_0 = 2^{-\ell}(1+w)^{\ell} < T(w)$ *for* $0 < w < 1 - 2\mu$ *and* $x_0 > T(w)$ *for* $1 > w > 1 - 2\mu$.

Proof.

$$T(w) - x_0 = \frac{x_0(1-x_0)(1-2\mu-w)}{-wx_0 + x_0 - 2\mu x_0 + 2\mu w}.$$

If $0 < w < 1 - 2\mu$, the denominator,

$$x_0(1-w) - 2\mu(x_0 - w) > x_0 2\mu - 2\mu x_0 + 2\mu w = 2\mu w > 0.$$

$(1 - 2\mu) > w$ implies $(1 - w - 2\mu) > 0$. The numerator is nonnegative. For $w > (1 - 2\mu)$, the numerator is negative and the denominator is positive. □

Lemma 2. $T(w) > 0$ *for* $0 < w < 1 - 2\mu$.

Recall that if $T < (1-\mu)^{\ell}$, then one fixed point occurs at $w = 1 - 2\mu$. In order for bistability to exist, there must be two additional fixed points, both less than $1 - 2\mu$. These are solutions to equation 5. The left hand drawing in Fig. 2 shows the plots of $T(w)$ for various values of μ. Bistability occurs for a fixed value of T if a horizontal line drawn at height T intersects the curve three times. The plot of $T(w)$ ends with a vertical line at $w = 1 - 2\mu$.

The right hand drawing in Fig. 2 shows the progression from three fixed points to one fixed point when T is increased. A fixed point occurs when $G(w)$ intersects the 45° line. When T =.4 there are three fixed points, one near zero, one at approximately $w = .5$, and one at the critical value of $w = 1 - 2\mu = .8$. When T = .43 the middle fixed point merges with the fixed point at w = .8. When T is further increased to .45, the last fixed point has disappeared completely, leaving only the fixed point near $w = 0$.

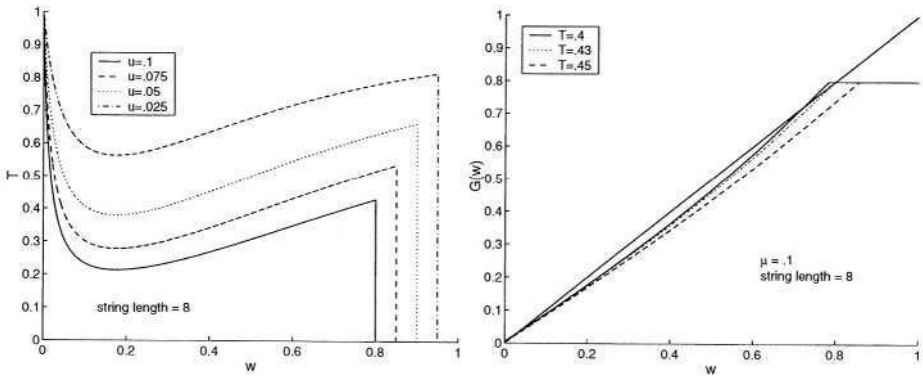


Fig. 2.

The (w, T) points where the curve has a zero slope can be critical values. As T increases through such a value, a pair of fixed points can appear or disappear.

Therefore it is useful to differentiate (5) and set the result equal to zero. This gives

$$0 = -\ell w^2 + (\ell + x_0 - 1)w + (x_0 - 1) \ . \tag{6}$$

The reader should note that since (6) is independent of μ, the minimums in the left hand side of Fig. 2 all occur at the same value of w.

Lemma 3 shows the conditions under which T has a local minimum in the region $0 < w < 1 - 2\mu$.

Lemma 3. $\frac{dT}{dw} = 0$ *has exactly one solution between* 0 *and* $1 - 2\mu$ *when* $1 - \ell\mu(1 - 2\mu) - \mu - (1 - \mu)^{\ell+1} < 0$.

Let w_c be the critical point of T between 0 and $1 - 2\mu$. Then $T(w_c)$ is a minimum since the first derivative of T passes from negative to positive. Since w_c is the only critical point, it must be a global minimum in the interval $0 < w < 1 - 2\mu$.

For the parameter values in the left hand side of Fig. 2, we calculate that $w_c = .1758$. If $.1758 < 1 - 2\mu$, or $\mu > .4121$ then T will not have an interior local minimum. We check that the hypothesis in Lemma 3 is not satisfied. $1 - \ell(1 - 2\mu) - \mu - (1 - \mu)^{\ell+1} > 0$ when $\mu > .4121$. By numerically analyzing the inequality, we find that the minimum string length that satisfies the hypothesis is $\ell = 4$.

Lemma 4. *In the case* $T < (1 - \mu)^{\ell}$, *if* $w_c < 1 - 2\mu$, *equivalently if*

$$1 - \ell\mu(1 - 2\mu) - \mu - (1 - \mu)^{\ell+1} < 0,$$

then there exists a value of T *that gives bistability.*

Proof. By Lemma 3 there exists horizontal lines that will cross the graph of $T(w)$ more than once. Each place of intersection represents a fixed point. Another fixed point of G exists at $w = 1 - 2\mu$. □

We also note that bistability exists if $T(w_c) < T < (1 - \mu)^{\ell}$. It remains to determine the stability of the fixed points.

Theorem 3. *If* $0 < \mu < 1/2$, *then there can be at most three fixed points for* $\widehat{G}$. *When there are three fixed points, they are stable, unstable and stable when ordered by* w *values. If there are two fixed points, then a small perturbation of either* T *or* μ *can give one fixed point.*

Proof. Since $\frac{dG}{dw} > 0$ then at a fixed point where the graph crosses from above to below the slope must be less than one and therefore must be stable. A fixed point where the graph crosses from below to above must have slope greater than one and is unstable.

Since $\widehat{G}(0) > 0$, when there are three fixed points, the graph must cross the diagonal from above to below, then below to above. □

We have exactly characterized the fixed points for the gene pool model on the NEEDLE fitness function. For fixed values of T and μ the location of the fixed points can be found using (4).

For a fixed value of μ the range of values of T which give bistability can be found. For example see the left drawing in Fig. 3. The area between the two curves is the region in (μ, T) space where bistability occurs. The top curve is $T = (1-\mu)^{\ell}$ and the lower curve is found by solving $\frac{dT}{dw} = 0$ to find w_c and then taking $T(w_c)$.

For a fixed value of T and ℓ the range of μ which give bistability can be found. This can be seen in the right hand drawing in Fig 3. This figure shows the region in (ℓ, μ) space for which bistability occurs. As noted before, a string length of 4 or more is needed for bistability. The top curve of each pair in this figure is $\mu = 1 - T^{\frac{1}{\ell}}$ The bottom curve of each pair was found for each ℓ by solving $T(w_c) = T$ for μ for T = .25, .3, .5, .7.

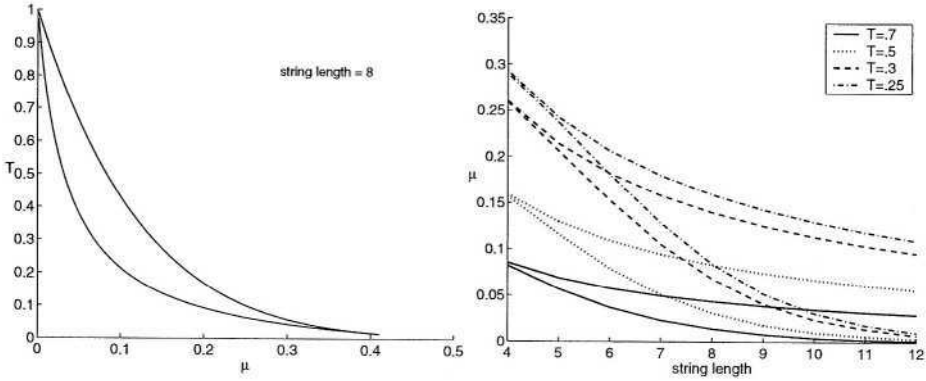


Fig. 3.

These results are surprising. Truncation selection with a small value of T is thought of as a very strong selection method. One might think that sufficiently strong selection would eliminate the possibility of bistability, but these results show bistability even with T = 0.1 and a string length of 4.

In [WRPS03] it was shown that the minimum string length for bistability with proportional selection is 6, and this occurs with a very small mutation rate and weak selection. (The extra increment in fitness of the needle string is a strength-of-selection parameter for proportional selection.) Here, the minimum string length for bistability is 4, and it can occur over a wide range of mutation rates for different values of T. This includes strong selection and high mutation.

5 Empirical Results

In this section we show that bistability can be observed in finite populations GAs with one point crossover and realistic settings for the mutation rate, the crossover rate, and the truncation fraction T.

The following procedure can be used to empirically test if a finite population GA with certain parameter settings has bistability for the NEEDLE fitness function. We run the GA with two different initial populations. First, we use a randomly chosen initial population. For the NEEDLE fitness, the population size should be large enough so that there are some copies of the optimal needle string. Second, we use an initial population with a high frequency of the needle string. If the GA with the first initial population does not move to a population with many copies of the optimal string, and with the second initial population maintains many copies of the optimal string, this indicates that the GA has bistability. (This experiment should be repeated several or many times since anything can happen on a single run of a GA.)

Simulations were performed using a Java program written by the first author and his students. Simulations were performed with truncation selection with $T = 0.5$, string length 8, mutation rate 0.06. (These values are in the middle of the bistability region in the right hand side of Fig. 3.) The initial population was either generated randomly or by choosing strings with a probability of 0.85 of a zero for each bit (biased initialization).

For these parameter settings, the model predicts fixed points at $w = 0.00355$ and $w = 0.8268$. Formula (2) gives the corresponding x_0 values of 0.004 and 0.6096. First, we verified the model by doing 100 runs with uniform crossover with crossover rate 1 and a population size 10,000. The average x_0 after 50 generations with random initialization was 0.00661 and with biased initialization was 0.6096. (Additional experiments for other parameters reported in [Cri03] also show very close agreement with the model for uniform crossover and genepool recombination.)

Next we did experiments with a more realistic fitness function and a weaker crossover. The fitness function was a simple royal road fitness with string length 36. The 36 loci were partitioned into 4 randomly chosen nonoverlapping sets (or blocks) of 9 loci each. These blocks were in general not contiguous on the string. A string received a fitness contribution of 1 for each block where the string had all zeros in that block. Thus, the fitness ranges from 0 to 4, and the all-zeros string is the only string with fitness 8. The experiments used one-point crossover with crossover rate 0.8, a mutation rate of 0.022, and truncation selection with $T = 0.5$. For biased initialization, the probability of a zero was 0.95. Population sizes were 100, and 10000. The following are the averages of 100 runs. The results reported are the average and maximum fitness at the end of 1000 generations. The standard error (1 standard deviation) follows in parentheses. The results clearly show a bistability-like phenomenon (perhaps there are more than 2 fixed points).

Population size	initialization	average fitness	maximum fitness
100	random	2.54 (0.19)	3.52 (0.52)
100	biased	2.69 (0.16)	3.98 (0.14)
10000	random	1.87 (0.37)	3.38 (0.49)
10000	biased	2.70 (0.01)	4.00 (0.00)

6 Conclusion

In this paper we have shown that an infinite population GPR model closely approximates the finite population two-parent uniform crossover GA when the fitness function exhibits a single peak. Under the gene pool recombination model, the complicated dynamical system of the finite population GA becomes tractable. In particular, we can explicitly calculate the fixed points and determine their stability by examining a single variable polynomial function. Finite population simulations suggest that TPR and GPR produce bistability. The fixed points produced in the simulations closely match those predicted by the model.

Furthermore, we have demonstrated that the infinite population GPR model correctly predicts the presence of bistability in the finite population GA. We have derived explicit formulas that relate the parameter values under which the bistability phenomena occurs.

There is a lesson for the practitioner. When the GA is initialized with a random population, bistability is a phenomenon that should be avoided since it may prevent the GA from finding peaks in the fitness. One way to do this is to keep the strength of recombination down. This can be done by reducing the crossover rate, or by choosing a "weaker" crossover (such as one-point or two-point instead of uniform). We are not saying that weaker recombination is always better since there have been studies, such as [SI99], that show that recombination can accelerate convergence to an optimum, even on a needle-in-the-haystack fitness function.

References

[BBN96] M. C. Boerlijst, S. Bonhoeffer, and M. A. Nowak. Viral quasi-species and recombination. *Proc. Royal Society London B,* 263:1577–1584, 1996.

[BC95] Shumeet Baluja and Rich Caruana. Removing the genetics from the standard genetic algorithm. In A. Prieditis and S. Russel, editors, *The Int. Conf. on Machine Learning 1995,* pages 38–46. Morgan Kaufmann Publishers, 1995.

[Cri03] Greg Cripe. Bistability of the needle function in the presence of truncation selection. Master's thesis, Univ. of Montana, Missoula, MT 59812, 2003.

[Gei44] H. Geiringer. On the probability of linkage in Mendelian heredity. *Annals of Mathematical Statistics,* 15:25–57, 1944.

[MM00] Heinz Mühlenbein and Thilo Mahnig. Evolutionary algorithms: From recombination to search distributions. In L. Kallel, B. Naudts, and A. Rogers, editors, *Theoretical Aspects of Evolutionary Computation,* pages 137–176. Springer Verlag, 2000.

[MM01] H. Mühlenbein and T. Mahnig. Evolutionary computation and beyond. In Y. Uesaka, P. Kanerva, and H. Asoh, editors, *Foundations of Real-World Intelligence,* pages 123–188. CSLI Publications, Stanford, CA, 2001.

[OH97] G. Ochoa and I. Harvey. Recombination and error thresholds in finite populations. In *Foundations of Genetic Algorithms 5,* pages 245–264. Morgan Kaufmann, 1997.

[PBR01] Adam Prügel-Bennett and Alex Rogers. Modelling genetic algorithm dynamics. In *Theoretical Aspects of Evolutionary Computing.* Springer Verlag, 2001.

[SI99] Hideaki Suzuki and Yoh Iwasa. Crossover accelerates evolution in gas with a babel-like fitness landscape: Mathematical analyses. *Evolutionary Computation,* 7(3):275–310, 1999.

[Sys93] G. Syswerda. Simulated crossover in genetic algorithms. In L. Darrel Whitley, editor, *Foundations of Genetic Algorithms 2,* San Mateo, 1993. Morgan Kaufmann.

[Vos99] M. D. Vose. *The Simple Genetic Algorithm: Foundations and Theory.* MIT Press, Cambridge, MA, 1999.

[WRN02] A. H. Wright, J. E. Rowe, and J. R. Neil. Analysis of the simple genetic algorithm on the single-peak and double-peak landscapes. In *Proceedings of the Congress on Evolutionary Computation (CEC) 2002,* pages 214–219. IEEE Press, 2002.

[WRPS03] A. H. Wright, J. E. Rowe, R. Poli, and C. R. Stephens. Bistability in a gene pool GA with mutation. In *Foundations of genetic algorithms (FOGA-7),* San Mateo, 2003. Morgan Kaufmann.

Appendix

The Walsh matrix W is a 2^ℓ by 2^ℓ matrix defined by $W_{i,j} = (-1)^{\#(i \otimes j)}$. The Walsh matrix is symmetric and $W^{-1} = 2^{-\ell}W$. If x is a population vector, then the Walsh transform of x is Wx and is denoted $\hat{x}$. Note that this definition uses a different scaling factor than the Walsh transform given by Vose [Vos99].

An Estimation of Distribution Algorithm Based on Maximum Entropy

Alden Wright[1], Riccardo Poli[2], Chris Stephens[3], W.B. Langdon[4], and Sandeep Pulavarty[1]

[1] Computer Science, University of Montana, Missoula, MT, 59812, USA
[2] Department of Computer Science, University of Essex, Colchester, UK
[3] Instituto de Ciencias Nucleares, UNAM, Mexico DF 04510
[4] Computer Science, University College, London, Gower Street, London, UK

Abstract. Estimation of distribution algorithms (EDA) are similar to genetic algorithms except that they replace crossover and mutation with sampling from an estimated probability distribution. We develop a framework for estimation of distribution algorithms based on the principle of maximum entropy and the conservation of schema frequencies. An algorithm of this type gives better performance than a standard genetic algorithm (GA) on a number of standard test problems involving deception and epistasis (i.e. Trap and NK).

1 Introduction

Genetic algorithms maintain a population of potential solutions to a problem. Each potential solution is assumed to have a fitness which is related to how well it solves the problem. The population undergoes selection alternated with crossover and/or mutation. Selection increases the frequency of high-fitness solutions and decreases the frequency of low-fitness ones, but does not add new solutions to the population. Crossover and mutation extend the sampling of the search space by creating new solutions from those currently in the population.

Estimation-of-distribution algorithms take a different approach to sample the search space. The population is used to estimate a probability distribution over the search space that reflects what are considered to be important characteristics of the population. A new population is then generated by sampling this distribution. This introduces diversity into the new population as well as preserving salient characteristics of the previous one. Thus, the sampling process usually replaces the crossover and mutation operators of the genetic algorithm. Estimation of distribution algorithms are reviewed in [1]. The MIMIC algorithm of [2] uses only second-order statistics and the Kullback-Leibler divergence.

The maximum entropy principle [3] has its historical roots in physics and is a widely used approach to estimating probability distributions from data. The key idea is that the associated probability distribution should agree with what is known but, apart from that, should express 'maximum uncertainty'. This allows the maximum possible freedom to adjust the distribution when new data become known. In practice observed data is used to derive a set of constraints on the

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 343–354, 2004.

estimated probability distribution. The entropy of this estimated distribution is then maximized subject to these constraints.

We assume a length-ℓ string representation, where the alphabet of symbols at any string position is an arbitrary finite set. A schema is a subset of the search space where the symbols at some string positions are defined (fixed) and at other string positions are arbitrary (variable). A schema is commonly denoted by a string of length ℓ where the special asterisk symbol $*$ is used to denote that any string symbol is possible at that position. The order of a schema is the number of defined (non-asterisk) positions. Given a frequency distribution over the search space and a schema, the corresponding schema frequency is just the sum of the relative frequencies of the elements of that schema.

One traditional way of "explaining" how genetic algorithms work is the building block hypothesis. This says that the GA builds high-fitness solutions by combining building blocks which are high-fitness, short, low-order schemata that can be thought of as representing partial solutions to the problem. This point of view came from an analysis of Holland's Schema Theorem [4] which, being an inequality, did not take into account the effects of schema construction. More recent exact schema theorems [5,6,7] have shown that, indeed, genetic algorithms "work" by combining building blocks. However, the relevant building blocks are dynamical, not static, of high "effective fitness" [5], not necessarily of high fitness, and, in general, are not short [8]. Although the traditional building block hypothesis is only a first order approximation, except under certain restricted circumstances, exact schema theorems can be used to understand the role of building blocks in a more rigorous way which does aid in our understanding of how these algorithms work.

Our approach is to use schema frequencies in a given population (after selection) to constrain the estimated probability distribution. The entropy of this distribution is subsequently maximized and the distribution sampled to produce a new population. The fitness of individuals in the new population is measured in the normal way. Schema frequencies are used because they are the most natural coarse-grained quantities associated with a population of strings. Since schema frequencies are sums of string frequencies, they should give more reliable estimates of properties of the fitness function than string frequencies. In addition, since the state of the algorithm can be completely described by schema frequencies, this should lead to tractable models of this class of algorithms.

On average, sampling from a maximum entropy distribution constrained by low-order schema frequencies preserves these schema frequencies in the new population. Therefore, we suggest that our algorithm will work well in problems where preservation of the building blocks corresponding the low-order schemata used in the algorithm is important. One might also imagine an extension of the algorithm so as to take into account the dynamical evolution of building blocks by changing dynamically the schemata whose frequency will be fixed.

We can now give an outline of the proposed algorithm.

1. Generate an initial random population.
2. Generate an initial collection of schemata.
3. Apply selection (as in a GA) to the population.

4. Calculate the frequencies of the schemata in the schema collection.
5. Sample the maximum entropy distribution to obtain a new population.
6. Go to step 3.

Steps 2 will be elaborated in section 3. Naturally, one should expect the performance of the algorithm to be sensitive to the schemata chosen in step 2.

2 Sampling from a Maximum Entropy Distribution Constrained by Schema Family Frequencies

In this section we give some basic definitions and then show how to sample the maximum entropy distribution constrained by the frequencies of an appropriate collection of schema families. We will use the term **maxent distribution** as shorthand for maximum entropy distribution. The results given in this section are proved in the appendix.

2.1 Entropy

If X is a finite set and $p(x)$ is a probability distribution over X, then the entropy of p is defined to be

$$S(X) = \sum_{x \in X} -p(x) \log(p(x))$$

where the base of the logarithm doesn't matter as long as one is consistent. We will assume base 2.

A **schema constraint,** (Q, a), on $p(x)$ consists of a set $Q \subset X$ and a real number $a \in [0,1]$ such that $\sum_{x \in Q} p(x) = a$. A **family of schema constraints** will be given by a pair $(\mathcal{T}, g)$, where $\mathcal{T}$ is a collection of schemata of X and g is a function from $\mathcal{T}$ to the unit interval [0,1]. Each schema constraint is of the form $(Q, g(Q))$ for some $Q \in \mathcal{T}$. The family $(\mathcal{T}, g)$ is **feasible** if there is some probability distribution that satisfies all of the schema constraints. If for each $Q \in \mathcal{T}$, $g(Q)$ is defined to be the frequency of Q in a given population, then $\mathcal{T}$ is feasible.

Lemma 1. Equal probability rule. *If $p(x)$ is the maximum entropy distribution on X subject to some schema constraints, and if x_1 and x_2 are indistinguishable with respect to these constraints (they are in the same schemata), then these points have equal probabilities, i.e., $p(x_1) = p(x_2)$.*

Given a single schema constraint on X that requires the sum of the probabilities of schema Q to be a, all points in Q will have the same probability $a/|Q|$ and all points in the complement will have probability $(1-a)/(|X|-|Q|)$. Where $|Q|$ is the number of strings which match Q and $|X|$ is the total size of the search space X. The following rule extends this observation.

Theorem 1. Disjoint sets rule[1] *Given a collection of schema constraints $(\mathcal{T}, g)$ such that the schemata of $\mathcal{T}$ are pairwise disjoint, the maxent distribution $p(x)$ subject to these constraints is given by:*

[1] This follows immediately from the equal probability rule.

$$p(x) = \begin{cases} \frac{g(Q)}{|Q|} & \text{if } x \in Q \in \mathcal{T} \\ \frac{1-\sum_{Q\in\mathcal{T}} g(Q)}{|X|-\sum_{Q\in\mathcal{T}}|Q|} & \text{if } x \notin Q \text{ for all } Q \in \mathcal{T} \end{cases}$$

A **schema family** is the collection of schemata which share a common set of defined positions. We will denote a schema family by a length-ℓ string over the alphabet $\{D,*\}$ where D denotes a position with a definite bit and $*$ denotes a position with a variable bit. Thus, if the alphabets are binary, $D*D**$ denotes the four schemata $\{0*0**,\ 0*1**,\ 1*0**,\ 1*1**\}$. Note that the schemata in a schema family are pairwise disjoint and thus the disjoint sets rule applies when all of the constraints are from a schema family.

We assume that the search space Ω can be written in the form:

$$\Omega = \mathcal{A}_1 \times \cdots \times \mathcal{A}_\ell$$

where each $\mathcal{A}_i$ is the finite alphabet of legal symbols at position i. Whenever we write Ω as a Cartesian product, such as $\Omega = X \times Y$ or $\Omega = X \times Y \times Z$, we will assume implicitly that each factor is a product of some of the $\mathcal{A}_i$. In other words, each factor corresponds to a set of string positions, and these sets form a partition of the set of string positions.

If $\Omega = X \times Y$, let Π_X and Π_Y denote the projections of Ω onto X and Y respectively. If Q is a schema, then $\Pi_X(Q)$ is the schema obtained by dropping all string positions corresponding to Y. For example, if X corresponds to the first three string positions and Y corresponds to the last three string positions, then $\Pi_X(1*0*01) = 1*0$. We will say that a schema or a schema family is **variable in** Y if all positions corresponding to Y are variable. I.e. all $*$. Note that if a schema Q is variable in Y, then $Q = \Pi_X(Q) \times Y$.

Suppose that $\Omega = X \times Y$ and that $p(x,y)$ is the maxent distribution on Ω, subject to a family of constraints $(\mathcal{T},g)$, where each $Q \in \mathcal{T}$ is variable in Y. Since $\Pi_X(Q) \times Y = Q$, g can naturally be extended to the sets $\Pi_X(Q)$ by defining $g(\Pi_X(Q)) = g(\Pi_X(Q) \times Y)$. It follows from the equal probability rule (lemma 1) that $p(x,y_1) = p(x,y_2)$ for any $x \in X$ and $y_1, y_2 \in Y$. Thus, the maxent distribution on X subject to the constraints $(\Pi_X(Q),g)$ is also the marginal distribution $p(x)$ of $p(x,y)$. We will call this distribution the maxent distribution on X subject to $\Pi_X(\mathcal{T},g)$.

Theorem 2. Non-overlapping schemata rule. *Let $(\mathcal{T}_X \cup \mathcal{T}_Y, g)$ be a collection of schema constraints on $X \times Y$ such that each $Q \in \mathcal{T}_X$ is variable in Y and each $Q \in \mathcal{T}_Y$ is variable in X. Then the maximum entropy distribution on $X \times Y$ is the independent product of the maximum entropy distribution $p(x)$ on X subject to $\Pi_X(\mathcal{T}_X,g)$ and the maximum entropy distribution $p(y)$ on Y subject $\Pi_Y(\mathcal{T}_Y,g)$. In other words, $p(x,y) = p(x)p(y)$.*

Example: Suppose that $\ell = 6$, and X corresponds to the first 2 string positions, Y corresponds to the third string position, and Z corresponds to the last 3 string positions. Let $\mathcal{E} = DD****$, $\mathcal{F} = **D***$, $\mathcal{G} = ***DDD$. Suppose that the frequencies of all of the schemata in these families are given by the function

g. We can first apply the non-overlapping schemata rule to schema families $\mathcal{E}$ and $\mathcal{F}$ to get the maxent distribution $p(x, y)$ on $X \times Y$ subject to $\Pi_{X \times Y}(\mathcal{E}, g)$ and $\Pi_{X \times Y}(\mathcal{F}, g)$. Then we can apply the rule again to the decomposition $(X \times Y) \times Z$ to get the maxent distribution on $X \times Y \times Z$ subject to all schema family constraints. For example, $p(101101) = g(10****)g(**1***)g(***101)$.

Theorem 3. Overlapping schemata rule. *Let $(\mathcal{T}_{X,Y} \cup \mathcal{T}_{Z,Y} \cup \mathcal{T}_Y, g)$ be a collection of schema constraints on $X \times Y \times Z$ such that $\mathcal{T}_{X,Y}$ is variable in Z, $\mathcal{T}_{Z,Y}$ is variable in X, and $\mathcal{T}_Y$ is variable in $X \times Z$. Let $p(y)$ be the maxent distribution on Y subject to $\Pi_Y(\mathcal{T}_Y, g)$, $p(x, y)$ be the maxent distribution on $X \times Y$ subject to $\Pi_{X,Y}(\mathcal{T}_{X,Y}, g)$, and $p(y, z)$ be the maxent distribution on $Y \times Z$ subject to $\Pi_{Y,Z}(\mathcal{T}_{Y,Z}, g)$. Assume that $p(y)$ is the marginal distribution of $p(x, y)$ and of $p(y, z)$. Then the maxent distribution on $X \times Y \times Z$ is given by*

$$p(x, y, z) = \frac{p(x, y)p(y, z)}{p(y)} \tag{1}$$

Example: Let $\mathcal{E} = DDD***$, $\mathcal{F} = *DD***$, $\mathcal{G} = *DDD**$. Suppose g is defined on all of the schemata of $\mathcal{E}$ and $\mathcal{G}$. Since all together probabilities must sum to 1 and since every schema in $\mathcal{F}$ is a disjoint union of schemata of $\mathcal{E}$, g is also implicitly defined on $\mathcal{F}$. Similarly, g is implicitly defined on $\mathcal{F}$ from the schemata of $\mathcal{G}$. In order to apply the overlapping schemata rule, these implicit definitions must agree. The overlapping schemata rule can be used to find the maxent probability on the schema family $DDDD**$ and hence on the whole search space. For example, $p(1001**) = g(100***)g(*001**)/g(*00***)$.

Now given constraints on the schema family $\mathcal{H} = **DDDD$, we can apply the overlapping schema rule again to incorporate these constraints assuming that the constraints on $**DD**$ defined implicitly from $\mathcal{G}$ agree with those from $\mathcal{H}$. (This would be the case as long as the constraints on $\mathcal{G}$ and $\mathcal{H}$ were derived from the schema frequencies of the same previous population.) For example, $p(100101) = g(100***)g(*001**)g(**0101)/(g(*00***)g(**01**))$.

Now suppose that the additional schema family was $\mathcal{J} = D**DDD$ instead of $\mathcal{H}$. In order to apply the overlapping schemata rule, the constraints on $D**D**$ defined implicitly from $\mathcal{J}$ would have to agree with the marginals of the probability distribution on $DDDD**$ derived above from $\mathcal{E}$ and $\mathcal{G}$. There is no reason to expect this to be the case even if all constraints were derived from the schema frequencies of the same previous population. The probability distribution on $DDDD**$ satisfies a conditional independence condition, whereas there is no reason to expect that the constraints on $D**D**$ derived from $\mathcal{J}$ and the schema frequencies of the previous population would satisfy this conditional independence condition.

3 Algorithms

In this section we give two algorithms based on maxent sampling.

3.1 The Order-1 Schemata Algorithm

This algorithm follows the general framework given in section 1 with the initial collection of schemata in step 2 being all order-1 schemata. Sampling from the maxent distribution is done using the non-overlapping schemata rule (theorem 3). It is not hard to see that this is just the UMDA (uniform multivariate distribution algorithm) of [9,10].

The UMDA algorithm is a simple estimation of distribution algorithm that preserves the frequencies corresponding of each variable independently of other variables. It works well [11] for problems without significant interactions between variables, but partial solutions of order more than one are disrupted and thus the algorithm may have difficulty solving problems where the fitness function has significant interactions between variables.

3.2 The Order-2 Contiguous Schemata Algorithm

A schema family is **contiguous** if the defined positions are sequential with no intervening variable positions. As a preliminary test of the idea of applying the idea of maximum entropy to estimation of distribution algorithms, we have the following algorithm that works with contiguous order-2 schemata. Since only contiguous schemata are used, one may expect that it will work best when interactions between variables tend to be between variables that are located close to each other on the string. The order of string positions could be rearranged to achieve this property using the techniques of [12] or [2], but we did not test this approach in this paper.

The order-2 contiguous algorithm follows the framework of section 1 with the collection of schemata in step 2 being all order-2 contiguous schemata. (The schemata may overlap.) The selection can be any selection method used in a GA. We have tested binary tournament and truncation selection—see section 4.

Sampling from the maxent distribution in step 5 of the algorithm is done by repeatedly applying the overlapping schemata rule (theorem 3). To create a new individual that is to be added to the new population, we initially start with the all $*$ schema. This is progressively refined, (using the overlapping schemata rule) with more and more bits being defined, until there are no $*$ left. When all positions are fixed the corresponding individual is added to the new population.

The first two positions are fixed by choosing a schema randomly from $DD*\ldots*$ using the schema frequencies as probabilities. The third position is fixed using the overlapping schemata rule using the schema family $*DD*\ldots*$. The fourth position is chosen using the schema family $**DD*\ldots*$, etc. (This will take $\ell - 1$ steps.)

For example, let us do this for $\ell = 4$. The new individual is initially represented by the schema $****$. A schema is chosen from the family $DD**$ according to the probabilities $g(00**)$, $g(01**)$, $g(10**)$, $g(11**)$. Suppose that $01**$ is chosen. The new individual schema is now $01**$. A schema is then chosen from $*DD*$. However, this must be a compatible schema, so one of $*10*$ or $*11*$ is chosen. (Two schemata are **compatible** if they agree wherever their defined positions overlap.) These two schemata are chosen using probability $g(*10*)/g(*1**)$

for $*10*$ and probability $g(*11*)/g(*1**)$ for $*11*$. Note that these probabilities add to 1. If $*10*$ is chosen, the new individual schema is then $010*$. One of the schemata $**00$ and $**01$ is then chosen in the same way.

One might ask why we are using order-2 schemata and not higher order schemata. Note that there is a chaining effect associated with the requirement that compatible schemata must be chosen on consecutive choices. For example, if both the schema $0000**$ and $1111**$ have high frequency, then our procedure is likely to choose either a sequence of zeros or a sequence of ones. In addition, using higher-order schemata can be a topic for further research.

If the frequency of an order-2 schema is zero in the population after the selection step, then it will be zero after step 5 as well. Thus, for the algorithm as described so far, such a schema will have zero frequency from this point on. This implies that the algorithm will eventually converge to a single individual. As this behavior may be undesirable we have added a mutation procedure to the algorithm. This is done by modifying the sampling procedure of step 5 as described above. The procedure is parameterized by a mutation rate. When a schema is chosen from an order-2 schema family, with a probability equal to the mutation rate, the schema frequencies are ignored and the schema is chosen from the set of compatible schemata using equal probabilities. A schema might be chosen which results in no compatible choices when processing the next schema family. In this case, the mutation procedure is forced when processing the next schema family. Note that, if mutation is applied, we are not sampling from the exact maxent distribution.

4 Empirical Results

The order-2 contiguous schemata algorithm for a binary alphabet was implemented in Java by Pularvarty and Wright. Individuals of populations were implemented as Java BitSets and schemata were represented as a pair of BitSets. For a schema, one BitSet represented the defined positions and the second represented the definitions of those defined positions. Then set operations (as implemented by computer bitwise operations) could be used to determine if an individual was an element of a schema. For example, if an individual is represented by the set A, the defined positions of schema Q by D, and the definitions by E, then the individual is in schema Q iff $D \cap A = E$.

For the order-2 contiguous algorithm we would expect that the algorithm would do well when there are building blocks that match the algorithm. The first class of test functions used was the concatenated trap function. Concatenated trap functions were designed with the building block hypothesis in mind [13,14].

A concatenated trap function of order k is a sum of trap subfunctions of order k. A trap function t of order k is defined by

$$t_k(x) = \begin{cases} |x| & \text{if } |x| \neq 0 \\ k+1 & \text{if } |x| = 0 \end{cases}$$

where $|x|$ is the number of ones in the binary string x. We used $k = 3$ and $k = 4$. The concatenated trap function is defined by dividing the string into

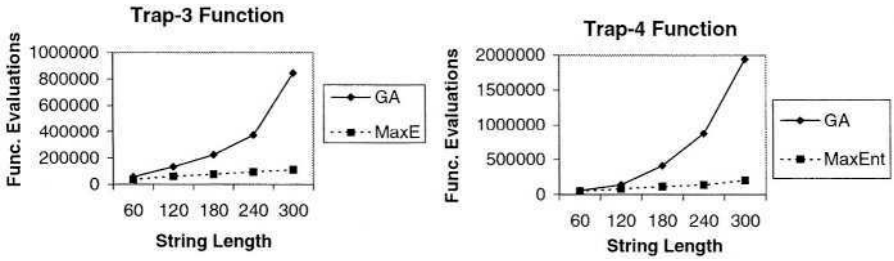


Fig. 1. On Trap benchmarks the GA scales much worse than maxent (order-2).

non-overlapping contiguous blocks of length k, and then defining an order-k trap function on each of these blocks. The value of the function is the sum of the trap functions.

Note that the optimum string is the all-zeros string, and that there are natural building blocks, namely the schemata with zeros for one block and all other positions variable. Since the building blocks are contiguous schemata, the structure of the problem matches the algorithm.

We compared the contiguous order-2 algorithm of this paper with a conventional genetic algorithm. For the experiments on the trap function and for both algorithms we used a mutation rate of 0.01, and truncation selection with a truncation parameter of 0.2. In other words, the best 1/5 of individuals are kept in the selection phase. The GA is sensitive to population size, and an attempt was made to choose a reasonably appropriate population for the GA. A population size of 5000 was used for the maximum entropy algorithm and for the GA, except when $k = 4$ and the string length was 240 or 300. In these two cases, population sizes of 10000 and 20000 were used. The genetic algorithm used one-point crossover with crossover rate 0.9. Algorithms were run until the optimum string was found. Figure 1 shows the average (over 20 runs) of number of fitness evaluations until the optimum was found. As the string length increases, the algorithm of this paper does much better than the one-point crossover GA.

The second class of fitness functions used are the NK-fitness functions [15]. These functions are also sums of simpler subfunctions. In this case, the simpler subfunctions are real-valued functions that depend on $K+1$ bits. Each subfunction is defined by a table mapping bit strings of length $K+1$ to real numbers, and the real numbers are chosen randomly using a uniform distribution from the interval [0,1]. There are two versions of NK-fitness functions. For the adjacent version, the string is divided into ℓ overlapping contiguous blocks of order-$K+1$ each, and the function is a sum of random subfunctions defined on each of these blocks. For the random version, the string is divided into ℓ overlapping non-contiguous blocks of $K+1$ bits each. The ith block is guaranteed to contain bit i. The positions of the remaining bits are chosen randomly. We used $K = 3$ so that the blocks contained 4 bits each.

For a given string length and K, there are many NK fitness functions depending on the choice of the random numbers defining the subfunctions, and in the case of the random version, on the choice of the bits that make up the

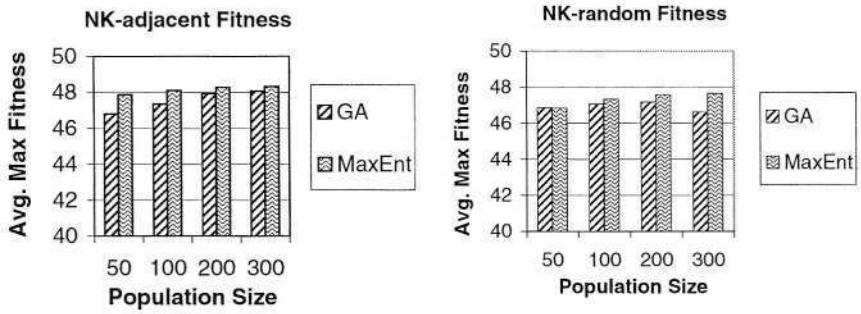


Fig. 2. Performance of GA v. maxent (order-2) for NK (N=64, K=3). Maxent is significantly better on both types of NK landscape.

block. The results given in Figure 2 are for string length 64, mutation rate 0.01, truncation selection with truncation parameter 0.2, and 5000 fitness evaluations. Each result is the average of 1000 runs.

The following procedure was used to statistically test whether the maxent algorithm was better than the GA. For each of 1000 trials, an NK fitness function was generated and then the GA and the Maxent algorithm were run with population sizes of 50, 100, 200. and 300. The fitness of the best individual over the 4 maxent runs was compared with the fitness of the best individual over the 4 GA runs. For NK-adjacent, the Maxent algorithm had better fitness 792 times, the GA had better fitness 198 times, and there was equal fitness 10 times. For NK-random, the Maxent algorithm had better fitness 646 times, the GA had better fitness 350 times, and there was equal fitness 4 times. Clearly these results are statistically significant.

5 Conclusion

A major advantage of the approach of this paper is the potential for modeling algorithms based on maximum entropy. The sampling step of the algorithm framework (step 5) depends only on the frequencies of the schemata in the schema collection (and on the outcome of random number generation). Thus, a model of the algorithm can be based on these frequencies rather than the frequencies of all strings. This means that the model is naturally coarse-grained.

This can be illustrated by existing models of algorithms based on gene pool recombination and linkage equilibrium, such as the UMDA algorithm. One analysis of the UMDA, based on quantitative genetics, appears in [11]. Fixed point analysis is used to explore the phenomenon of bistability in the UMDA with mutation in [16]. Both of these analyses use the fact that the population is in linkage equilibrium after gene pool recombination. When order-1 schema families are used, step 5 of our algorithm framework is gene pool recombination, and in this case, linkage equilibrium is equivalent to maximum entropy. In our algorithm that uses contiguous order-2 schema families, the state of the population

is reduced to the frequencies of the schemata in these families, and a model of the algorithm can be based on keeping track only of these frequencies.

In conclusion, we have presented a novel paradigm based on maximum entropy for constructing estimation of distribution algorithms. Preliminary experiments with an algorithm based on this idea using order-2 schemata on deceptive trap functions (Figure 1) and NK landscapes (Figure 2) are promising. Further work on using higher order schemata should be done.

Acknowledgements. CRS is grateful for support from DGAPA project IN100201 and Conacyt project 41221-F. AW, RP and WBL are grateful for hospitality of the Instituto de Ciencias Nucleares, where this work was initiated, and to the above grants for financial support.

References

1. Pedro Larrañaga. *A review on estimation of distribution algorithms,* chapter 3, pages 57–100. Kluwer Academic Publishers, 2002.
2. Jeremy S. de Bonet, Charles L. Isbell, Jr., and Paul Viola. MIMIC: Finding optima by estimating probability densities. In Michael C. Mozer et. al., editor, *Advances in Neural Information Processing Systems,* volume 9, page 424. MIT Press, 1997.
3. E. T. Jaynes. Information theory and statistical mechanics. *Physical Review,* 106:620–630, 1957.
4. John Holland. *Adapdation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, Michigan, 1975.
5. C. R. Stephens and H. Waelbroeck. Schemata evolution and building blocks. *Evolutionary Computation,* 7(2):109–124, 1999.
6. Ricardo Poli. Exact schema theory for genetic programming and variable-length genetic algorithms with one point crossover. *Genetic Programming and Evolvable Machines,* 2(2):123–163, 2001.
7. William B. Langdon and Riccardo Poli. *Foundations of Genetic Programming.* Springer-Verlag, 2002.
8. C. R. Stephens, H. Waelbroeck, and R. Aguirre. Schemata as building blocks: does size matter. In *Foundations of Genetic Algorithms 5,* pages 117–133. Morgan Kaufmann, 1997.
9. Heinz Mühlenbein and Thilo Mahnig. Convergence theory and application of the factorized distribution algorithm. *Journal of Computing and Information Technology,* 7(1):19–32, 1999.
10. Heinz Mühlenbein and Thilo Mahnig. FDA – a scalable evolutionary algorithm for the optimization of additively decomposed functions. *Evolutionary Computation,* 7(4):353–376, 1999.
11. Heinz Mühlenbein. The equation for the response to selection and its use for prediction. *Evolutionary Computation,* 5(3):303–346, 1997.
12. Robert E. Heckendorn and Alden H. Wright. Efficient linkage discovery by limited probing. In Erick Cantú Paz et al., editor, *Genetic and Evolutionary Computation – Gecco 2003,* LNCS 2724, pages 1003–10014. Springer Verlag, 2003.
13. Kalyanmoy Deb and David E. Goldberg. Analyzing deception in trap functions. In *Foundations of Genetic Algorithms - 2,* pages 93–108. Morgan Kaufmann, 1992.
14. Martin Pelikan, David E. Goldberg, and Erick Cantu-Paz. Linkage problem, distribution estimation, and bayesian networks. *Evolutionary Computation,* 8(3):311–340, 2000.

15. Stuart A. Kauffman. *The Origins of Order.* Oxford University Press, 1993.
16. A. H. Wright, J. E. Rowe, R. Poli, and C. R. Stephens. Bistability in a gene pool GA with mutation. In *Foundations of genetic algorithms-7,* San Mateo, 2003. Morgan Kaufmann.
17. Ameil Feinstein. *Foundations of Information Theory.* The Maple Press Company, York, PA, USA, 1959.
18. Silviu Guiasu and Abe Shenitzer. The principle of maximum entropy. *The Mathematical Intelligencer,* 7:42–48, 1985.

Appendix

In this section we give justification for the equal probability rule, the non-overlapping schemata rule, and the overlapping schemata rule.

Proof of the equal probability rule: Suppose that there are points $x_1, x_2 \in A$ such that $p(x_1) \neq p(x_2)$. Define the probability distribution q by $q(x_1) = q(x_2) = (p(x_1) + p(x_2))/2$ and $q(x) = p(x)$ for all $x \notin \{x_1, x_2\}$. Clearly, q satisfies all of the schema constraints. Since $-p \log p$ is a concave function of p, the entropy of q is greater than the entropy of p, contradicting the fact that p is the maximum entropy distribution. □

Following [17] can define the **conditional entropy** of a probability distribution $p(x, y)$ over $X \times Y$ by

$$S(X|Y) = \sum_Y p(y) S(X|y) = -\sum_Y \sum_X p(x, y) \log p(x|y)$$

The following is lemma 4 of chapter 2 of [17].

Lemma 2. *$S(X|Y) \leq S(X)$ with equality iff X and Y are probabilistically independent, i. e., if $p(x, y) = p(x)p(y)$.*

The following is taken from equation (5) of [18] and Lemma 3 of chapter 2 of [17].

Lemma 3. *$S(X, Y) = S(X|Y) + S(Y) \leq S(X) + S(Y)$, with equality if and only if X and Y are probabilistically independent.*

Proof of the non-overlapping schemata rule: We prove the rule for schema families $\mathcal{E}$ and $\mathcal{F}$ which correspond to the product decomposition $X \times Y$. The more general case follows by induction from this case.

Lemma 3 shows that $S(X, Y) \leq S(X) + S(Y)$ with equality iff $p(x, y) = p(x)p(y)$. It is sufficient to show that if $p(x, y)$ is defined to be $p(x)p(y)$, then $p(x, y)$ satisfies all of the schema constraints.

Let $Q \in \mathcal{T}_X$. Then

$$\begin{aligned}\sum_{(x,y)\in Q} p(x, y) &= \sum_{(x,y)\in \Pi_X^{-1}(\Pi_X(Q))} p(x)p(y) \\ &= \sum_{x\in \Pi_X(Q)} p(x) \sum_{y\in Y} p(y) \\ &= g(\Pi_X(Q)) \cdot 1 = g(Q)\end{aligned}$$

The proof for $Q \in \mathcal{T}_Y$ is similar. □

Lemma 4. $S(X, Z|Y) = S(X|Y, Z) + S(Z|Y)$

Proof.

$$
\begin{aligned}
S(X|Y,Z) &= -\sum_X \sum_Y \sum_Z p(x,y,z) \log p(x|y,z) \\
&= -\sum_Y p(y) \sum_X \sum_Z p(x,z|y) \log p(x|y,z) \\
&= -\sum_Y p(y) \sum_X \sum_Z p(x,z|y) \log \frac{p(x,z|y)}{p(z|y)} \qquad \text{since } p(x|y,z) = \frac{p(x,z|y)}{p(z|y)} \\
&= -\sum_Y p(y) \sum_X \sum_Z p(x,z|y) \big(\log p(x,z|y) - \log p(z|y) \big) \\
&= -\sum_Y p(y) \sum_X \sum_Z p(x,z|y) \log p(x,z|y) + \sum_Y p(y) \sum_X \sum_Z p(x,z|y) \log p(z|y) \\
&= -\sum_Y p(y) \sum_X \sum_Z p(x,z|y) \log(p(x,z|y) + \sum_Z \sum_Y p(y,z) \log p(z|y) \\
&= S(X,Z|Y) - S(Z|Y)
\end{aligned}
$$

The following lemma is a special case of lemma 6 of chapter 2 of [17].

Lemma 5. $S(X|Y, Z) \leq S(X|Y)$ *with equality iff*

$$p(x, z|y) = p(x|y)p(z|y),$$

or equivalently, if

$$p(x, z|y) = p(x)p(z)/p(y).$$

Proof of the overlapping schemata rule: In view of lemmas 4 and 5, it is sufficient to show that if $p(x, y, z)$ is defined by $p(x, y, z) = p(x, y)p(y, z)/p(y)$, then all of the constraints are satisfied.

First, let $Q \in \mathcal{T}_{X,Y}$. Recall that $Q = \Pi_{X \times Y}(Q) \times Z$. Then

$$
\begin{aligned}
\sum_{(x,y,z) \in Q} \frac{p(x,y)p(y,z)}{p(y)} &= \sum_{(x,y) \in \Pi_{X \times Y}(Q)} \frac{p(x,y)}{p(y)} \sum_{z \in Z} p(z,y) \\
&= \sum_{(x,y) \in \Pi_{X \times Y}(Q)} \frac{p(x,y)}{p(y)} p(y) \quad \text{since } p(y) \text{ is the marginal of } p(y,z) \\
&= \sum_{(x,y) \in \Pi_{X \times Y}(Q)} p(x,y) = g(Q)
\end{aligned}
$$

The cases when $Q \in \mathcal{T}_{Z,Y}$ and $Q \in \mathcal{T}_Y$ are similar. □

Dependency Structure Matrix Analysis: Offline Utility of the Dependency Structure Matrix Genetic Algorithm

Tian-Li Yu and David E. Goldberg

Illinois Genetic Algorithms Laboratory (IlliGAL)
Department of General Engineering
University of Illinois at Urbana-Champaign
104 S. Mathews Ave, Urbana, IL 61801
{tianliyu, deg}@illigal.ge.uiuc.edu

Abstract. This paper investigates the off-line use of the dependency structure matrix genetic algorithm (DSMGA). In particular, a problem-specific crossover operator is design by performing dependency structure matrix (DSM) analysis. The advantages and disadvantages of such an off-line use are discussed. Two schemes that helps the off-line usage are proposed. Finally, those off-line schemes are demonstrated by DSMGA on MaxTrap functions.

1 Introduction

Dependency structure matrix genetic algorithm (DSMGA) [1] was proposed under the inspirations of Holland's observation of the importance of linkage [2], Goldberg's genetic algorithm (GA) decomposition [3], and the use of dependency structure matrix (DSM) in organization theory [4,5]. In particular, the DSMGA adopts DSM analysis to identify linkage groups or building blocks (BBs) and then utilizes the BB information to perform a BB-wise crossover which tries to maximize BB mixing and minimize BB disruptions. The DSM analysis utilizes a DSM clustering algorithm [6] which turns pair-wise dependencies into linkage-group information, or BB information. The first proposal of the DSMGA [1] performs the DSM analysis every generation to retrieve BB information, like most probabilistic model building GAs [7] or estimation of distribution algorithms [8] do. However, the DSM analysis can also be performed off-line. In other words, one can simply apply the DSM analysis once to retrieve BB information and then use a simple GA (sGA) with a crossover operator designed for the problem according to the BB information.

The purpose of this paper is to investigate the off-line use of DSM analysis for GAs. The paper first gives an introduction to the DSMGA. The advantages and disadvantages of the off-line DSM analysis are then discussed, and two schemes for off-line usage are proposed. To demonstrate the off-line schemes of DSM analysis, a MaxTrap function is tested. Finally, the paper discusses some possible future work, followed by conclusions.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 355–366, 2004.

2 An Introduction to DSMGA

This section gives a brief introduction to the DSMGA. Readers who are interested in DSM clustering are referred to [6]. For more details about DSMGA, please refer to [1].

2.1 DSM and DSM Clustering

A DSM is a matrix where each entry d_{ij} represents the dependency between node i and node j [4,5]. Entries d_{ij} can be real numbers or integers. The larger the d_{ij} is, the higher the dependency is between node i and node j. The diagonal entries (d_{ii}) have no significance and are usually set to zero or blacked-out. Figure 1 gives an example of DSM.

Fig. 1. A DSM. "x" means that dependency exists; the blank squares means no dependency. This figure illustrates, for example, that A and B are independent, and that A and C are dependent. Clustering is not so obviously at the first glance.

The goal of DSM clustering is to find subsets of DSM elements (i.e., clusters) so that nodes within a cluster are maximally dependent and clusters are minimally interacting. DSM clustering is a sophisticated task which requires expertise [9]. For example, not too many people know how to cluster the DSM in Figure 1. However, after we reorder the nodes (Figure 2), it is easily seen that the DSM can be cleanly clustered into three parts: {B, D, G}, {A, C, E, H}, and {F}.

Yu, Yassine, and Goldberg [6] proposed the following objective function by using the minimal description length principle.

$$f_{DSM}(M) = (n_c \log(n_c) + \log(n_n) \Sigma_{i=1}^{n_c} cl_i) + (|S|(2\log(n_n) + 1)), \qquad (1)$$

where f measures the description length that a model M needs to describe a given DSM; n_c is the number of clusters in M; n_n is the number of nodes of the DSM; cl_i is the size of the i-th cluster in M; S is the set of mis-described data of M. The above objective function has shown capable to cluster a given DSM, and the clustering results competes with human experts.

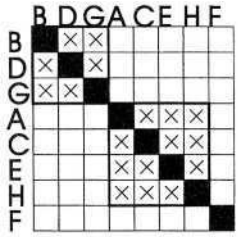


Fig. 2. The same DSM in Figure 1 but after reordered. The DSM can be cleanly clustered as ((B,D,G)(A,C,E,H),(F)).

2.2 Utilizing DSM Clustering to Identify BBs: The DSMGA

The DSM clustering algorithm can be thought as an linkage-identification algorithm which turns pair-wise linkage information into high-order linkage information. DSMGA [1] is proposed based on this thought.

DSMGA utilizes statistical analysis to estimate the fitness of order-2 schemata. Based on a nonlinearity-detection method similar to LINC [10], a DSM where each entry d_{ij} represents the linkage between gene i and gene j is created. By applying the DSM clustering algorithm, the pair-wise linkage information is turned into BB information, which is then utilized to achieve the BB-wise crossover [11]. The DSMGA has shown capable to correctly identify BBs and efficiently solve problems with uniformly-scaled BBs.

3 Why Go for Offline: The Pros and the Cons

There are mainly three types of linkage-identification methods: (1) Perturbation/statistical analysis (*e.g.* mGA [12], gemGA [13], LINC/LIMD GA [10], DSMGA [1]) (2) linkage adaptation (*e.g.* LEGO [14], LLGA [15]), and (3) probabilistic model building [7,8]. Among these three types, the first type of linkage-identification methods is most suitable for off-line usage because they do not require promising individuals; instead, those methods favor a uniformly distributed population over the solution space, or perturbations from several individuals. The remaining of this section discusses the pros and the cons of performing linkage-identification methods off-line.

Pros:

Computational Time Saving. If the linkage-identification algorithm is used off-line, because it is only performed once, obviously there is a speedup obtained from the linkage-identification procedure. The speedup should be in the order of the number of generations of the GA.

The off-line usage does not only save computational time on linkage-identification, it also possibly saves some number of function evaluations. By

considering genetic drifting and statistical error of decision-making, Pelikan [16] has shown that to guarantee a high confidence of obtaining correct dependencies, roughly a population size of $O(l)$ is needed, where l is the problem size. By assuming an infinite population size and perfect mixing, the time to convergence has shown to grow proportional to $O(\sqrt{l})$ [17,11] (for a finite population size, see [18]). The above models suggests a lower bound of the number of function evaluations $O(l^{1.5})$ for a GA with linkage-identification techniques.

If the linkage-identification techniques are performed off-line, and every BB is correctly identified, according to the gambler's ruin model [19], a population size $O(l^{0.5}\log l)$ is needed (the $\log l$ is so when the failure rate is set to $O(\frac{1}{l})$). Therefore, the lower bound of the number of function evaluations becomes $O(l)+O(l^{0.5}\log l)\times O(\sqrt{l}) = O(l\log l)$, where the first $O(l)$ comes from the off-line linkage identification, and $O(l^{0.5}\log l)\times O(\sqrt{l})$ comes from the production of the population size and the convergence time. This is surely just a loose lower bound because an infinite population size and perfect mixing is assumed in the time-to-convergence model.

The accuracy of the linkage model should be taken into consideration when we try to estimate a possible speedup. When the linkage-identification method is applied off-line, a fewer number of individuals are investigated and the BB information might be less accurate. As a result, the inaccuracy of the BB information causes a longer convergence time. This will be mentioned later when the disadvantages are discussed.

BB information reusability. Another reason to perform linkage identification off-line is that the BB information retrieved from one problem might be reusable for another problem. Take the car design as an example. The car design usually has similar multiple objectives but the design for different cars weight those objectives differently. For example, the design for one car might put importance on acceleration, while the design for another car might emphasize more on comfortability. However, no matter how different the design weights those objectives, some dependencies are invariant. For instance, one can imagine that the design of the brake system should depend more on the engine and depend less on the windscreen wiper. In such cases, the BBs for the series of the problems are similar, and hence the BB information are reusable. After one of those problems is optimized by the GA with linkage-identification techniques, other problems should be optimized efficiently by a sGA with the same BB information.

Population size estimation. Several population-sizing models have been proposed including the decision-making model [20] and the gambler's ruin model [19]. To use those population-sizing models, several parameters are needed including the number of BBs m, the order of BBs k, the variance of fitness contributed by BBs σ^2_{BB}, and the minimal signal magnitude between the correct BB and the most competing BB d_{min}. The first two parameters are straightforward once the BB information is obtained. However, it is not an easy task to estimate the latter two parameters, and we simply leave it as future work for now. As a

conclusion, the off-line usage of linkage-identification methods possibly enables us to estimate the population size needed for the GA.

Cons:

Less linkage model error tolerance. In a worst-case scenario, a misidentified BB in the off-line BB information makes the BB difficult to correctly converge for the GA because of BB disruptions. If the signals of BBs are not too small for the linkage-identification method to detect (otherwise, both on-line and off-line linkage identification would fail), when the linkage-identification method is performed every generation, the disruption would not be that severe. When a BB is misidentified in some generation because the current population does not reveal the signal of that BB, as long as the linkage-identification method is not too inaccurate, the BB would be correctly identified in the next generation with a high probability. Yu and Goldberg [21] indicates that as long as the number of misidentified BBs e is smaller than $O(\sqrt{m})$, where m is the number of BBs of the problem, the convergence of $(m-1)$ BBs is highly possible. However, if the linkage-identification method is performed off-line and it misidentifies e BBs, in the worst case, only $(m-e)$ BBs would correctly converge. Therefore, the quality of the retrieved BB information is more important in the off-line usage. In the next section, two schemes are proposed to alleviate the quality problem.

BB-invariance assumption. One of the assumptions of the off-line use of linkage-identification methods is that BBs are invariant. However, it is known that some optimization problems are hierarchical [22], and the dependencies varies with the degree of the GA convergence. In this case, probably the linkage-identification method needs to be performed every several generations, and the scheme becomes a evaluation relaxation version of the on-line usage which performs the linkage-identification algorithm every generation. If on-line estimation of the number of errors of BB information can be obtained, we can perform the linkage-identification algorithm only when the number of errors exceeds some predefined threshold.

4 Identifying Linkage Offline: Two Schemes

In this section, we propose two schemes that help for the off-line usage. Later, in section 5, the strength and weakness of these two schemes are shown empirically.

4.1 Sizing the Population Properly

One important issue when the linkage-identification algorithm is performed off-line is to determine the population size needed for the algorithm. Population sizing is especially important for the off-line usage. A non-sufficient population size will decrease the BB information quality and cause a great loss of solution quality.

Here we propose the following scheme, which is similar to the parameterless GA [23]. First, we choose a reasonable, but small population size N_1. For example, a population size of $N_1 = l$ is a good initial guess, where l is the problem size. In the next trial, N_1 more individuals are investigated, and the linkage-identification algorithm is performed on the doubled population size, $N_2 = 2N_1$. For the next trial, N_2 more individuals are investigated. This procedure is repeated until the BB information does not vary. If the population size actually needed to reveal correct linkages with a high probability is *N,* this scheme would at most costs $8N$ number of function evaluations (in which case we are unlucky at *N,* and get correct linkages on $2N$ and $4N$). Depending on the noise of linkage identification and the time constraint, practically one should stop the procedure when the BB information retrieved in the last two runs are similar enough. Note that this population size is only used when performing the linkage-identification algorithm. After that, the GA performs on a population size estimated by one of the population-sizing models [20,19], by some prior knowledge, or by empirical observations. Note that unlike the parameterless GA, the number of function evaluations consumed does not increase that fast, when the doubling-population-size procedure is stopped at a population size *N,* the number of function evaluations consumed is only *N.*

4.2 Combining BB Information

As indicated in the previous section, one of the disadvantages of the off-line usage of the linkage-identification method is that the off-line usage is less error-tolerant. If the BB information contains e errors, probably only $(m - e)$ BBs would correctly converge. Suppose that we stop the above doubling-population-size scheme at some point, and the BB information contains some errors. Now we are facing the following two questions: (1) Should we double the population size again to make the BB information more accurate or execute the linkage-identification algorithm on an independent population of the same size and try to combine the BB information, and (2) if the latter scheme is chosen, how to combine BB information? This subsection proposes a method to combine two BB information and leaves the first question to section 5.

As indicated in [21], there are two types of error that can happen for a linkage-identification algorithm. One is that the linkage-identification algorithm indicates that two genes are linked but in reality they are not; the other is that the linkage-identification algorithm indicates no linkage between two genes but in reality they are linked. The first type of error slows down the BB mixing [24] and causes a larger estimation of the population size (because of the higher-order linkage). The second type of error causes BB disruptions. When the linkage-identification method is performed off-line, the second type of error means a lower solution quality as mentioned in section 3. In other words, in the off-line scheme, the second type of error is more destructive than it is in the on-line scheme. Therefore, when combining two copies of BB information, the combined BB information indicates no linkage between two genes only when both copies of BB information indicate so.

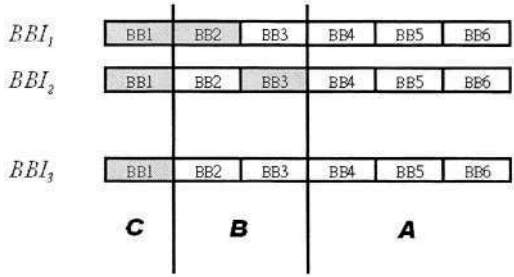


Fig. 3. Combining BB information. BBI_3 is the combination of BBI_1 and BBI_2. The shaded BBs are misidentified. Suppose a proportion $e1$ of BBs are misidentified in BBI_1, and $e2$ proportion in BBI_2. BBI_3 at most misidentifies a proportion e_1e_2 of BBs. The proportion of correctly identified BBs (regions A and B) in BBI_3 is $(1-e_1)(1-e_2)$.

Suppose that we have two independent copies of BB information, BBI_1 and BBI_2; let BBI_3 be the combined BB information according to the above scheme (Figure 3). Suppose that BBI_1 causes e_1 proportion of BB disruptions, and BBI_2 causes e_2 proportion of BB disruptions. On average, a proportion $(1-e_1)(1-e_2)$ of BBs are correctly identified in both BBI_1 and BBI_2, and they are also correctly identified in BBI_3 (region *A* in Figure 3). A proportion $(e_1)(1-e_2)+(e_2)(1-e_1)$ of BBs will be disrupted by one of BBI_1 and BBI_2, while BBI_3 will correctly identify those BBs according to the combining scheme (region *B* in Figure 3). A proportion e_1e_2 of BBs will be disrupted by both BBI_1 and BBI_2, and BB disruptions might also occur by using BBI_3 (region *C* in Figure 3). However, BBI_3 still have a chance not to disrupt them. For instance, if a BB {1, 2, 3, 4, 5} is misidentified by BBI_1 as {1, 2, 3} and {4, 5}, and misidentified by BBI_2 as {1, 2} and {3, 4, 5}, BBI_3 will still correctly identify that BB. To be conservative, we assume that the region *C* in Figure reffig:BBI will always cause BB disruptions. Then BBI_3 will only cause a proportion of e_1e_2 BB disruptions.

Note that the above calculations assume that BBI_1 and BBI_2 are independent.

4.3 A Little Complexity Analysis

This subsection shows that the above usage of multiple copies of BB information does not increase the lower bound of the number of function evaluations that GAs need.

Suppose that b different copies of BB information were retrieved by performing the linkage-identification methodoff-line b times, and on average the number of misidentified BBs is e. According to [21] (also see this for the meaning of the misidentified BB), $e < O(\sqrt{m})$ must be satisfied; otherwise, even the on-line version of the GA would not achieve a $(m-1)$-BB convergence. Assuming that the errors are uniformly distributed, the probability for a copies

of BB information misidentifying a specific BB is then $\frac{e}{m}$. The probability that all b copies of BB information contains wrong information for a specific BB is then $\left(\frac{e}{m}\right)^b$. The expected number of BBs that all b copies of BB information misidentify is $m\left(\frac{e}{m}\right)^b$. To achieve a $(m-1)$-BB convergence, $m\left(\frac{e}{m}\right)^b \leq 1$ must be satisfied. Given that $e < O(\sqrt{m})$, we can obtain that b is roughly bounded by $O(\log m)$. The estimation of number of function evaluations when we talked about the computational time saving of the off-line usage becomes $O(l \log l) + O(l^{0.5} \log l) \times O(\sqrt{l}) = O(l \log l)$, which means, the use of multiple different copies of BB information does not increase the lower bound of the number of function evaluations.

5 Empirical Results

This section demonstrates the off-line scheme by performing off-line DSM analysis on a 5 × 10 MaxTrap function (a concatenated trap function of 10 5-bit traps). A GA with $(\lambda + \mu)$ selection, where $\lambda = 100$ and $\mu = 1000$, and uniform crossover is used to cluster the DSM of genetic dependencies (for the linkage identification). All experiment results are averaged over 50 independent runs. The reason of choosing MaxTrap as the test base is that if a BB is misidentified, most likely the crossover operator would disrupt the BB. For more details about deceptions, see [25].

The doubling-population-size scheme on average needed 11712 function evaluations (6400 for 17 times; 12800 for 83 times), and correctly identifies 99.8% BBs (100% for 98 times; 90% for 2 times). As predicted, the number function evaluations is much larger than needed (Figure 4).

Figure 4 illustrates the use of the combining BB information scheme. The dashed line represents the combination of the BB information given by DSM analysis on two populations of the same size. For instance, for number of function evaluation 4000, the direct scheme (solid line) simply applies the DSM analysis on a population of size 4000. The combining scheme (dashed line) applies the DSM analysis on two independent populations of size 2000 and then uses the combined BB information. It can be seen that when the BB information is highly erroneous, combining BB information is not worthwhile, and simply doubling the population would be better. However, when the BB information is more accurate (roughly when the BB information is 92% accurate in experiments), the combining scheme outperforms the direct scheme.

The reason might be that when the population is too small, doubling the population size will greatly increase the accuracy of pair-wise dependency detection. However, when the population is large enough to reveal pair-wise dependencies, the errors mainly come from the the GA failure on DSM clustering. In the experiments, the proportion of correctly identified BBs roughly saturates at 0.98. The combining scheme outperforms the direct scheme since the combination of two 0.9 BB information will correctly identify more than $1 - 0.1 \times 0.1 = 0.99$ BBs.

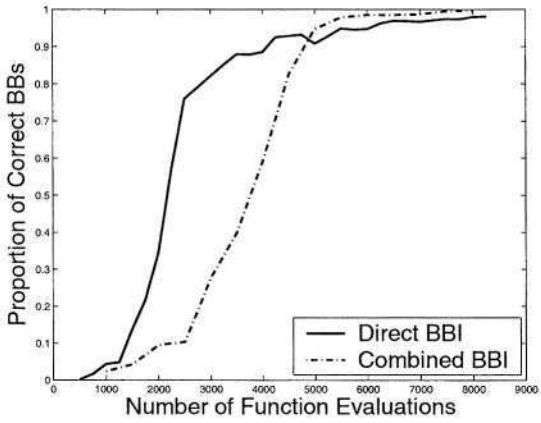


Fig. 4. The number of function evaluations versus the proportion of correctly identified BBs for two different schemes. The solid line represents directly applying the DSM analysis on individuals. The dashed line represents the combination of the BB information given by DSM analysis on two populations of the same size.

We can do a little modeling for the two schemes. Assume that the probability that one chromosome reveals a particular pair-wise dependency is p. The probability that all N independent chromosome fail to reveal a particular pair-wise dependency can be modeled as $q(N) = (1-p)^N$. Suppose that for a BB to be identified, s pair-wise dependencies need to be detected. Assuming pair-wise dependencies are uniformly important, the probability that the signal of a BB is too weak to identify can be approximated as $P_1(N) = 1 - (1 - q(N))^s$. In addition, even if the signal of a BB is strong enough for the linkage-identification method to identify, the linkage-identification method might still fail with a small probability P_2 because of stochastic behaviors. The probability that the linkage-identification method fails to identify a particular BB is then $error_{Scheme_1}(N) = 1 - (1 - P_1(N))(1 - P_2)$. The value of function $error_{Scheme_1}$ also varies with p and s, but here we emphasize that it is a function of N. If we combine two copies of BB information, the error of the combining scheme can be modeled as $error_{Scheme_2}(N) = error^2_{Scheme_1}(N/2)$. Figure 5 plots the models with $p = 0.1$, $s = 10$, and $P_2 = 0.05$. The models basically agree with the empirical observations in Figure 4.

If the proportion of correctly identified BBs can be estimated, it is possible to combine the strength of the two schemes. Suppose that the population size is N in the current iteration, the BB information is BBI_1, and the proportion of correctly identified BBs is $Q(BBI_1)$. In the next generation, when N_1 more individuals are being investigated, the DSM analysis is performed on both the new N individuals and the combined $2N$ individuals. Suppose the BB information are BBI_2 and BBI_3 respectively. Let BBI_4 be the combined BB information of BBI_1 and BBI_2. Then we switch to the combining-BB-information scheme

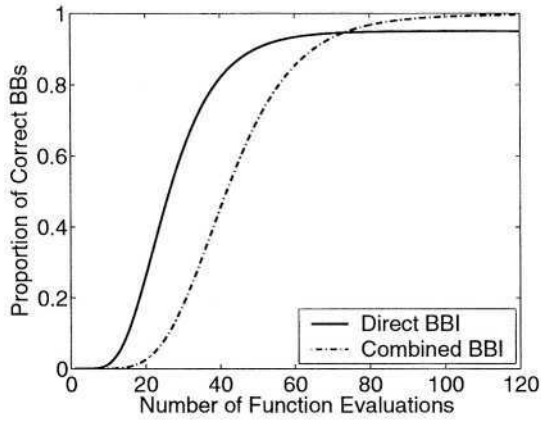


Fig. 5. Models for the two schemes in Figure 4.

when $Q(BBI_4) > Q(BBI_3)$; otherwise, we keep doubling the population size until the combined BB information is better.

6 Future Work and Conclusions

This paper leaves a number of possibilities for future work. The empirical results indicated that the combining-BB-information scheme works better when the BB information is accurate. We would like to see if it is possible to suggest an optimal switching time between the two schemes. The discussions in this paper are not limited to separable problems, but more experiments need to be done to verify that, and some modifications of the discussions can be expected. In section 3, we mentioned about utilizing the BB information to estimate the needed population size, and we are currently investigating the possibility of doing so.

To conclude, this paper investigated the off-line use of DSM analysis. The advantages and disadvantages of the off-line use of linkage-identification methods were discussed. Two schemes that helps the off-line usage were proposed. The number of function evaluations needed for the two schemes were analyzed. To demonstrate those schemes, experiments of a MaxTrap function were performed. The results showed that the off-line usage of DSMGA is applicable, and DSM analysis is suitable for designing a problem-specific crossover operator.

Acknowledgment. This work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant F49620-03-1-0129, and by the Technology Research, Education, and Commercialization Center (TRECC), at University of Illinois at Urbana-Champaign, administered by the National Center for Supercomputing Applications (NCSA) and funded by

the Office of Naval Research under grant N00014-01-1-0175. The U.S. Government is authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research, the Technology Research, Education, and Commercialization Center, the Office of Naval Research, or the U.S. Government.

References

1. Yu, T.-L., Goldberg, D.E., Yassine, A., Chen, Y.-p.: Genetic algorithm design inspired by organizational theory: Pilot study of a dependency structure matrix driven genetic algorithm. Proceedings of Artificial Neural Networks in Engineering 2003 (ANNIE 2003) (2003) 327–332 (Also IlliGAL Report No. 2003007).
2. Holland, J.H.: Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor, MI (1975)
3. Goldberg, D.E.: The design of innovation: Lessons from and for competent genetic algorithms. Kluwer Academic Publishers, Boston, MA (2002)
4. Steward, D.V.: The design structure system: A method for managing the design of complex systems. IEEE Transactions on Engineering Management **28** (1981) 77–74
5. Yassine, A., Falkenburg, D.R., Chelst, K.: Engineering design management: An information structure approach. International Journal of production research **37** (1999) 2957–2975
6. Yu, T.-L., Yassine, A., Goldberg, D.E.: A genetic algorithm for developing modular product architectures. Proceedings of the ASME 2003 International Design Engineering Technical Conferences, 15th International Conference on Design Theory and Methodology (DETC 2003) (2003) DTM–48657 (Also IlliGAL Report No. 2003024).
7. Pelikan, M., Goldberg, D.E., Lobo, F.G.: A survey of optimization by building and using probabilistic models. IlliGAL Report No. 99018, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL (1999)
8. Larrañaga, P., Lozano, J.A., eds.: Estimation of Distribution Algorithms. Kluwer Academic Publishers (2002)
9. Sharman, D., Yassine, A., Carlile, P.: Characterizing modular architectures. ASME 14th International Conference, DTM-34024 (2002)
10. Munetomo, M., Goldberg, D.E.: Identifying linkage groups by nonlinearity/non-monotonicity detection. Proceedings of the Genetic and Evolutionary Computation Conference 1999: Volume 1 (1999) 433–440
11. Thierens, D., Goldberg, D.E.: Convergence models of genetic algorithm selection schemes. In: Parallel Problem Solving from Nature, PPSN III. (1994) 119–129
12. Goldberg, D.E., Korb, B., Deb, K.: Messy genetic algorithms: Motivation, analysis, and first results. Complex Systems **3** (1989) 493–530
13. Kargupta, H.: The performance of the gene expression messy genetic algorithm on real test functions. Proceedings of 1996 IEEE International Conference on Evolutionary Computation (1996) 631–636

14. Smith, J., Fogarty, T.C.: Recombination strategy adaptation via evolution of gene linkage. Proceedings of the 1996 IEEE International Conference on Evolutionary Computation (1996) 826–831
15. Harik, G.R., Goldberg, D.E.: Learning linkage. Foundations of Genetic Algorithms **4** (1996) 247–262
16. Pelikan, M.: Bayesian optimization algorithm: From single level to hierarchy. Ph.d. dissertation, University of Illinois at Urbana-Champaign (2002) (Also IlliGAL Report No. 2002023).
17. Mühlenbein, H., Schlierkamp-Voosen, D.: Predictive models for the breeder genetic algorithm: I. Continuous parameter optimization. Evolutionary Computation **1** (1993) 25–49
18. Ceroni, A., Pelikan, M., Goldberg, D.E.: Convergence-time models for the simple genetic algorithm with finite population. IlliGAL Report No. 2001028, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL (2001)
19. Harik, G., Cantú-Paz, E., Goldberg, D.E., Miller, B.L.: The gambler's ruin problem, genetic algorithms, and the sizing of populations. Proceedings of 1997 IEEE International Conference on Evolutionary Computation (1997) 7–12
20. Goldberg, D.E., Deb, K., Clark, J.H.: Genetic algorithms, noise, and the sizing of populations. Complex Systems **6** (1992) 333–362
21. Yu, T.-L., Goldberg, D.E.: Toward an understanding of the quality and efficiency of model building for genetic algorithms. Proceedings of the Genetic and Evolutionary Computation Conference 2004 (2004) (To appear) (Also IlliGAL Report No. 2004004).
22. Simon, H.A.: The Sciences of the Artificial. The MIT Press, Cambridge, MA (1968)
23. Harik, G.R., Lobo, F.G.: A parameter-less genetic algorithm. Proceedings of the Genetic and Evolutionary Computation Conference 1999: Volume 1 (1999) 258–265
24. Sastry, K., Goldberg, D.E.: Analysis of mixing in genetic algorithms: A survey. IlliGAL Report No. 2002012, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL (2002)
25. Deb, K., Goldberg, D.E.: Analyzing deception in trap functions. Foundations of Genetic Algorithms 2 (1993) 93–108

Toward an Understanding of the Quality and Efficiency of Model Building for Genetic Algorithms

Tian-Li Yu and David E. Goldberg

Illinois Genetic Algorithms Laboratory (IlliGAL)
Department of General Engineering
University of Illinois at Urbana-Champaign
104 S. Mathews Ave, Urbana, IL 61801
{tianliyu, deg}@illigal.ge.uiuc.edu

Abstract. This paper investigates the linkage model building for genetic algorithms. By assuming a given quality of the linkage model, an analytical model of time to convergence is derived. Given the computational cost of building the linkage model, an estimated total computational time is obtained by using the derived time-to-convergence model. The models are empirically verified. The results can be potentially used to decide whether applying a linkage-identification technique is worthwhile and give a guideline to speed up the linkage model building.

1 Introduction

Holland [1] suggested that operators learning linkage information to recombine alleles might be necessary for genetic algorithm (GA) success. Many such methods [2] have been developed to solve the linkage problem. The linkage model can be implicit (*e.g.* LLGA [3]) or explicit (*e.g.* LINC [4]), probabilistic (probabilistic model building GAs [5], or estimation of distribution algorithms [6]) or deterministic (*e.g.* DSMGA [7]). Those methods have different strength in identifying linkage and consume different computational time. Because some of the linkage-identification methods are computationally expensive, recently there is a trend of applying speedup techniques on those linkage-identification methods including parallelism [8]. Another possible speedup technique is the evaluation relaxation scheme [9]. In a unified point of view, a linkage-identification method with the evaluation relaxation technique applied can be considered as another linkage-identification method which is more efficient and possibly less accurate. With the same idea, a simple GA (sGA) can be thought as a GA with a linkage-identification method which does not consume additional computations and always *reports* tight linkages.

Two questions might frequently come into a GA researcher's mind: (1) On which classes of problems does a specific GA design have strength and weakness, and (2) is a speedup technique worthwhile to use? Take the Bayesian optimization algorithm (BOA) as an example. BOA works well on problems with

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 367–378, 2004.

linkages; however, for problems where linkage is not important, like OneMax, a sGA would perform better since building a Bayesian network is computationally time-consuming [10]. A linkage-identification method with the evaluation relaxation techniques is more efficient but possibly less accurate. The lower accuracy might elongate the convergence time, and speeding up the model-building process might result in a slower GA convergence. Although the above two questions seem loosely related, they boil down to the following more basic question: Given two different model builders with different accuracies and different computational costs, should one use a more accurate, but more computationally expensive model builder or a less accurate, but less computationally expensive model builder?

The purpose of this paper is to answer the above question, and by doing this, the results lead us toward a better understanding of the relationship between building linkage models and GA convergence. The paper first defines the errors of a linkage model, and then time to convergence is derived for a given number of errors of the linkage model. The time-to-convergence model is then used to derive the total computational time. A number of experiments are done to verify the models derived in the paper. Finally, discussions of the contributions and possible future work conclude this paper.

2 Time to Convergence for Linkage Model Building

This section derive the time-to-convergence model of GA by assuming a given linkage model quality. All derivations done by assuming an infinite population size and prefect mixing by population-wise crossover. For simplicity, we further assume that the problem contains a unique global optimum.

2.1 The Errors of a Linkage Model

The term "linkage" is widely used in GA field, but defining linkage is not an easy task. In this paper, the linkage can be loosely defined as follows. *If linkage exists between two genes, recombination might result in lowly fit offspring with high probability if those two genes are not transferred together from parents to offspring.* A group of highly linked genes forms a linkage group, or a building block (BB) in [2].

A linkage model is a model telling which genes form linkage groups. For instance, the boolean flags in LEGO [11], the genetic ordering in LLGA [12], the clustering model in eCGA [13], and the DSM clustering in DSMDGA [7] are all linkage models. Two different types of errors can happen when a linkage model is adopted to describe the genetic linkage. One is that the linkage model links those gene which are not linked in reality. The other is that the linkage model does not link those gene which are linked in reality. The first type of error does not disrupt correct BBs but they slow down BB mixing. Since perfect mixing (in short, BBs are uniformly distributed over the population on each particular position) is one of the presumptions of the time-to-convergence model used later

in the paper, this paper only focuses on the second type of error and leaves the first type of error as future work.

The quality of a linkage model can be quantified by the number of errors it makes. For example, consider a problem with four BBs, where $\{BB_1, BB_2, BB_3, BB_4\}$={{1,2,3}, {4,5,6} , {7,8,9}, {10,11,12}}, and a linkage model $\{BB_1', BB_2', BB_3', BB_4', BB_5'\}$={{1,2}, {3,4,5,6}, {7,8}, {9,10,11}, {12}}. By ignoring the first type of errors, the linkage model can be re-expressed as {{1,2}, {3}, {4,5,6}, {7,8}, {9}, {10,11}, {12}}. As a result, the linkage model produces 3 errors and only BB_2 is correctly identified.

2.2 Building Block Disruptions

According to [2], effectively mixing BBs is critical for a GA success. In most traditional GAs, BB mixing is done by performing crossover. However, if BBs are not correctly identified, crossover will also disrupt BBs (addressed in Holland's [1] schema theory). This subsection derives the upper bound of the expected number of BB disruptions given the number of errors of the linkage model.

Before derivations, it is convenient to define two terms, a *correct BB* and an *incorrect BB.* If the genes in a BB have the same values as those genes at the same locations of the globally optimal solution, the BB is called a correct BB; otherwise, it is called an incorrect BB. A BB disruption occurs when a correct BB becomes an incorrect BB after crossover.

By the assumption, after crossover is performed, a misidentified BB is recombined by two portions which come from two different BBs. When one portion comes from a correct BB and the other portion comes from an incorrect BB, a BB disruption probably occurs. To be conservative, we assume that recombining a portion of a correct BB with a portion of an incorrect BB always results in an incorrect BB. That happens when the most competitive incorrect BB is exactly the compliment of the correct BB. Likewise, we assume that recombining incorrect BBs always produces incorrect BBs.

Assume that there is a proportion p of correct BBs in the current population. For a randomly chosen BB, a BB disruption occurs when (1) it is misidentified, (2) it is a correct BB, and (3) it is going to be recombined with an incorrect BB. Therefore, the probability of a BB disruption occurrence is given by

$$\left(1-(1-\frac{1}{m})^e\right)p(1-p), \tag{1}$$

where m is the number of BBs in a chromosome. When the number of errors e is much smaller than the number of BBs m, it is valid to assume that only one errors occurs for each misidentified BB. The above equation can be approximated as

$$\frac{e}{m}p(1-p). \tag{2}$$

In a population of size N, there are total Nm BBs. The expected number of BB disruptions is then $Nep(1-p)$.

2.3 Time-to-Convergence Model

Mühlenbein and Schlierkamp-Voosen [14] gave the following time-to-convergence model for OneMax problem by assuming a infinite population size and prefect mixing.

$$t_{conv} = \left(\frac{\pi}{2} - \arcsin(2p_0 - 1)\right)\frac{\sqrt{l}}{I}, \tag{3}$$

where p_0 is the initial proportion of ones, I is selection intensity, and l is the length of chromosome. The time-to-convergence model can be derived from the following equation [14,15].

$$p_{t+1} - p_t = \frac{I}{\sqrt{l}}\sqrt{p_t(1-p_t)}, \tag{4}$$

where p_t is the proportion of ones at generation t.

Miller [16] extended the time-to-convergence model to problems with uniformly scaled BBs. If the linkage model successfully identifies every BB, by treating correct BBs as 1's and incorrect BBs as 0's, the problem is then similar to the OneMax problem. The only difference is that the growth of correct BBs are slower. The reason is that a chromosome with more correct BBs does not always have a higher fitness value. For example, {0" 0" 1} might have a lower fitness value than {0' 0' 0'}, where 1 represents a correct BB, 0' represents an incorrect BB with a high fitness value, and 0" represents another incorrect BB with a low fitness value. The growth of correct BBs is then modelled as follows.

$$p_{t+1} - p_t = \frac{I'}{\sqrt{m}}\sqrt{p_t(1-p_t)}, \tag{5}$$

where $I' \leq I$, and p_t is the proportion of correct BBs at generation t.

When the linkage model has some errors, the growth of correct BBs is slowed down. After selection, the growth of correct BBs is still governed by Equation 5, $p_{t,selected} = p_t + \frac{I'}{\sqrt{m}}\sqrt{p_t(1-p_t)}$. The proportion of disrupted BBs is given by

$$\frac{e}{m}p_{t,selected}(1 - p_{t,selected}). \tag{6}$$

Hence, the proportion of correct BBs for the next generation is

$$p_{t+1} = p_{t,selected} - \frac{e}{m}p_{t,selected}(1 - p_{t,selected}). \tag{7}$$

By approximating the BB disruption in Equation 6 as $\frac{e}{m}p_t(1-p_t)$ (the proportion of disrupted BBs is calculated according to the proportion of correct BBs before selection) and adopting $p(1-p) \leq \frac{1}{2}\sqrt{p(1-p)}$ for $0 \leq p \leq 1$, the proportion of disrupted BBs can be approximated as:

$$\frac{e}{2m}\sqrt{p_t(1-p_t)}. \tag{8}$$

The proportion of correct BBs in the next generation is then given by

$$p_{t+1} - p_t = \left(\frac{I'}{\sqrt{m}} - \frac{e}{2m}\right)\sqrt{p_t(1-p_t)}. \tag{9}$$

Following a similar procedure in [14] and [15], one can derive the time-to-convergence model.

$$t_{conv} = \frac{\left(\frac{\pi}{2} - \arcsin(2p_0 - 1)\right)}{\frac{I'}{\sqrt{m}} - \frac{e}{2m}}. \tag{10}$$

The dimensionless model can be obtained as

$$\frac{t_{conv}(e=0)}{t_{conv}} = 1 - \frac{e}{2I'\sqrt{m}}. \tag{11}$$

The above equation also suggests that when $e \geq 2I'\sqrt{m}$, BB disruptions are severer than BB growths; the behavior of the GA is then like a random search and difficult to converge. Therefore, defining the *critical number of errors* $e_{critial}$ as the largest number of BBs that a linkage model could misidentify while a $(m-1)$-BB convergence is still possible, the following relation between $e_{critial}$ and m can be expressed as

$$e_{critial} = 2I'\sqrt{m}. \tag{12}$$

Since I' is only loosely related to the problem, the key idea of the above equation is that $e_{critial} = O(\sqrt{m})$.

3 Overall Computational Time

This section models the overall computational time using the time-to-convergence model in the previous section. The overall computational time is then used to derive an optimal decision of which linkage model should be used. A similar methods of modelling in this section can be found in [9].

Assuming the GA operators consume α computational time each generation, if a linkage model on average misidentifies e BBs and consumes β computational time, the overall computation time is then

$$T = t_{conv}(e)(\alpha + \beta). \tag{13}$$

Suppose we have two linkage-identification methods, M_1 and M_2, which misidentifies e_1 and e_2 BBs, and consumes α_1 and α_2 computational time, respectively. The ratio of the overall computational time of GAs which adopt those two linkage-identification methods is given by

$$\frac{T_{M_1}}{T_{M_2}} = \frac{2I'\sqrt{m} - e_2}{2I'\sqrt{m} - e_1} \cdot \frac{\alpha + \beta_1}{\alpha + \beta_2}. \tag{14}$$

If the ratio is smaller than 1, method M_1 should be used, and vice versa.

The above equation is difficult to use in practice mainly because the the number of BBs that a linkage model misidentifies is not easy to estimate. Nevertheless, the equation gives some insights and mathematical foundation to the following observations.

1. When the fitness function evaluation is computationally expensive ($\alpha >> \beta$), the first term ($\frac{2I'\sqrt{m}-e_2}{2I'\sqrt{m}-e_1}$) dominates the decision, and a time-consuming, but more accurate linkage-identification method is favored.

2. On the contrary, when the fitness function evaluation is relatively computationally inexpensive ($\alpha << \beta$), the second term ($\frac{\alpha+\beta_1}{\alpha+\beta_2}$) dominates the decision, and a less accurate, but computational efficient linkage-identification method is favored.

When errors are few ($e_1, e_2 << \sqrt{m}$) and the computational cost of the linkage model builder is relatively cheap compared to the GA operators ($\beta_1, \beta_2 << \alpha$), the above equation can be approximately simplified as

$$\frac{T_{M_1}}{T_{M_2}} = 1 + \frac{e_1 - e_2}{2I'\sqrt{m}} + \frac{\beta_1 - \beta_2}{\alpha}. \tag{15}$$

The above equation suggests the following definitions: The *quality* of a linkage model is $Q = 1 - \frac{e}{2I'\sqrt{m}}$ and the *relative cost* of a linkage model is $c = \frac{\beta}{\alpha}$. For any linkage model with $Q < 0$ or $e > 2I'\sqrt{m}$, the GA is difficult to converge. Given two linkage-identification methods M_1 and M_2 with qualities Q_1, Q_2 and their relative costs c_1, c_2 respectively, by defining $\triangle Q = Q_1 - Q_2$ and $\triangle c = c_1 - c_2$, the decision ratio becomes

$$\frac{T_{M_1}}{T_{M_2}} = 1 + (\triangle c - \triangle Q). \tag{16}$$

Therefore, if $\triangle c < \triangle Q$, M_1 is better; otherwise, M_2 is better. Consider M_2 as an evaluation relaxation version of M_1: M_2 is more computationally inexpensive but less accurate than M_1 ($\triangle c > 0$ and $\triangle Q > 0$). The evaluation relaxation is worthwhile only when $\triangle c > \triangle Q$, or in other words, when the save of relative cost is greater than the loss of quality.

4 Empirical Results

This section presents the experiments that empirically verify our models. This section first describes the design of the experiments, and then it shows the empirical results followed by discussions.

4.1 Experiment Design

The test function should be carefully chosen to fulfill the worst-case analysis: If a BB is misidentified, most likely it will be disrupted by crossover. Based on the reason, a MaxTrap function is used (for more details and analysis of a trap function, see [17]). A correct BB in a 5-bit trap is 11111, and its most competing BB is 00000. If the BB is misidentified, any crossover would result in incorrect BBs with some ones and zeros in them. By the nature of the trap function, those incorrect BBs would then tend to become the most competing BB, 00000.

In addition, two assumptions used in the time-to-convergence model derivations should be satisfied: (1) Infinite population-sizing and (2) perfect mixing. In implementation, a large-enough population size is used and uniform crossover is performed to ensure high mixing rate. In particular, a pair-wise BB-specific uniform crossover is used, the main difference between a population-wise BB-specific uniform crossover and the pair-wise BB-specific uniform crossover is that the amount of BB disruptions predicted in Equation 6 is reduced by a factor of 2, because the swap at any cross-site only is performed with a probability 0.5.

A 5 × 50 and a 5 × 100 MaxTrap are tested. The 5-bit trap is defined as:

$$trap_5(u) = \begin{cases} \frac{4-u}{5}, & u = 0, 1, 2, 3, 4 \\ 1, & u = 5 \end{cases}, \tag{17}$$

where u denotes the number of ones in the input 5-bit block.

A linkage model without any error ($e = 0$) gives the following BB information: $\{\{1, 2, 3, 4, 5\}, \{6, 7, 8, 9, 10\}, \cdots\}$. For $e > 0$, first, e BBs are randomly selected, their genes are randomly shuffled, and then those selected BBs are randomly split into two parts. For example, a randomly selection BB {6, 7, 8, 9, 10} might be shuffled as {6, 9, 10, 7, 8} and then split into {6, 9} and {10, 7, 8}. The processed BB information is then used to perform a pair-wise BB-specific uniform crossover. Tournament selection with tournament size $s = 2$ is used. According to Blickle and Thiele [18], the selection intensity $I \simeq 0.5763$. Assuming I' is a constant, I' can be then estimated by comparing the estimation of time to convergence given by Equation 10 and the empirical time to convergence for $e = 0$. As a result, $I' \simeq 0.752I \simeq 0.4334$. All experiments are averaged over 100 independent runs.

To approximate the asymptotic behavior of the time-to-convergence model, four times of the population size estimated by the gambler's ruin model [12] are used. For example, a population size of 1539 should supply enough BBs for GAs to process the 5 × 100 MaxTrap function. In the experiments, the population size is set to 6156.

4.2 Results and Discussions

The relationship between the number of BB disruptions and the correct BB proportion for different linkage model errors for the 5 × 100 MaxTrap is presented in Figure 1. The figure shows that the derived model (Equation 6) is more accurate when e is small. The reason is that Equation 6 is an overestimate which

ignores the possibility that correct BBs might be produced from the combination of two incorrect BBs. When e is larger, the crossover of 11111 and 00000 gives more incorrect BBs with some ones and zeros in them. The recombination of those BBs then has a higher probability to reproduce the correct BB, 11111. Note that the model indeed bounds the empirical data.

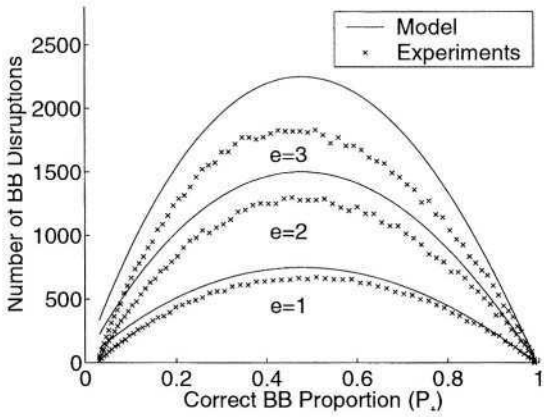


Fig. 1. Numbers of BB disruptions for the 5×100 MaxTrap. The BB disruption is severe in the middle of the GA run, and mild when the GA is merely or nearly converged.

It is easily seen that BB disruption is severe when roughly half of the BBs in the population are correct; Only few BB disruptions occur when the proportion of correct BBs is close to either 0 or 1. The observation suggests the following possible adaptive speedup scheme. Instead of recalculating the linkage model every generation, the linkage model is only updated every several generations at the beginning and the end of the GA run. Of course, it is non-trivial to estimate the degree of convergence of the GA for any given generation, and that leaves a room for the future work.

The time to convergence for different linkage model error e is shown in Figures 2 and 3. The convergence condition is that on average, there are $(m-1)$ correct BBs in each individual, where m is the number of BBs of the problem. As expected, since the number of BB disruptions is overestimated, the predicted time to convergence is also longer than the actual number of generations that the GA needs to converge. For a smaller e (compared with $\sqrt{m}$), the model predicts better. The empirical data agree better with the model for the 5×100 MaxTrap than the the 5×50 one, because for the same value of error, the error is relatively smaller compared to $\sqrt{m}$ for a larger m.

Finally, Equation 12 predicts that the GA could hardly converge for $e > 2I'\sqrt{m}$. In the experiments, because a pair-wise crossover is used and the swap is performed with a probability 0.5 on every cross-site, the number of BB dis-

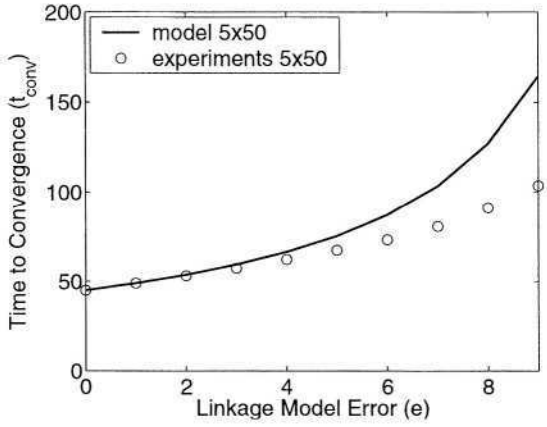


Fig. 2. Time to convergence for the 5x50 MaxTrap. The model overestimates the time to convergence. For a relatively smaller e, the model predicts better.

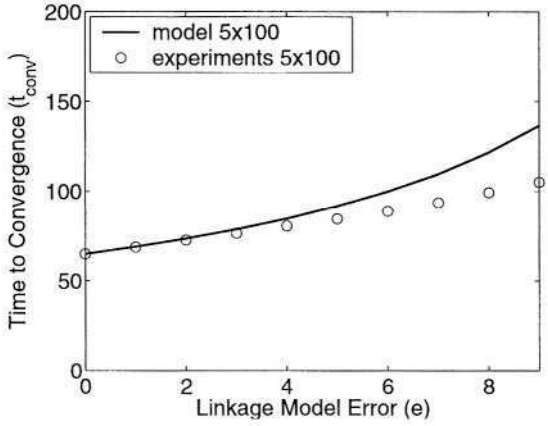


Fig. 3. Time-to-convergence for the 5x100 MaxTrap. The model predicts better than that for 5x50 because then number of errors e is relatively smaller compared with $\sqrt{m}$.

ruptions is only half as modelled in section 2, and hence $e_{critial} = 4I'\sqrt{m}$. If the linkage model contains more errors than $e_{critial}$, BB disruption rate is higher than BB growth rate, and the GA is difficult to converge. The critical number of errors versus the number of BBs is plotted in Figure 4. As shown in the figure, the results basically agree with the model: The critical number of error $e_{critical}$ grows proportionally to the square root of the number of BBs ($\sqrt{m}$). Since the number of BB disruptions is overestimated, the estimation of $e_{critical}$ should be a underestimation. However, due to the finite population size, the empirical $e_{critical}$ is smaller than predicted.

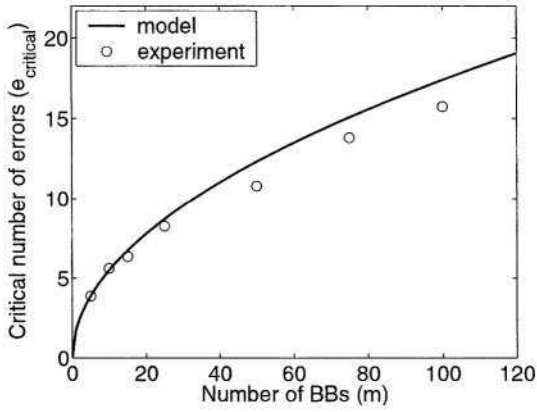


Fig. 4. The critical number of errors versus the number of BBs. Roughly $e_{critical} = O(\sqrt{m})$.

5 Conclusions and Future Work

In this paper, by assuming a infinite population size and perfect mixing, the time to convergence was derived for a given number of misidentified BB of the linkage model. The derivations gave several insights about the model-building for GAs:

1. The BB disruptions is severe when roughly half of the BBs are converged. The BB disruptions are mild when the GA is merely or fully converged.

2. Speedup might be achieved by applying evaluation relaxation techniques on the model-building techniques at the beginning and at the end of the GA run.

3. The critical number of errors (the largest number of BBs that a linkage model could misidentify while a $(m - 1)$-BB convergence is still possible) grows proportional to the square root of the number of BBs $(e_{critical} = O(\sqrt{m}))$.

As future work, we would like to integrate the the first type of errors discussed in section 2.1 into our models. Also, we are investigating how to estimate the number of errors of a linkage model. The number of errors might be estimated by observing the convergence behavior of the GA. If that is doable, we could perform the linkage-identification algorithm only when the number of errors exceeds some predefined threshold. By doing that, a speedup is obtained while the quality of solution is maintained.

Acknowledgment. This work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant F49620-

03-1-0129, and by the Technology Research, Education, and Commercialization Center (TRECC), at University of Illinois at Urbana-Champaign, administered by the National Center for Supercomputing Applications (NCSA) and funded by the Office of Naval Research under grant N00014-01-1-0175. The U.S. Government is authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research, the Technology Research, Education, and Commercialization Center, the Office of Naval Research, or the U.S. Government.

References

1. Holland, J.H.: Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor, MI (1975)
2. Goldberg, D.E.: The design of innovation: Lessons from and for competent genetic algorithms. Kluwer Academic Publishers, Boston, MA (2002)
3. Harik, G.R., Goldberg, D.E.: Learning linkage. Foundations of Genetic Algorithms 4 (1996) 247–262
4. Munetomo, M., Goldberg, D.E.: Identifying linkage groups by nonlinearity/non-monotonicity detection. Proceedings of the Genetic and Evolutionary Computation Conference 1999: Volume 1 (1999) 433–440
5. Pelikan, M., Goldberg, D.E., Lobo, F.G.: A survey of optimization by building and using probabilistic models. IlliGAL Report No. 99018, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL (1999)
6. Larrañaga, P., Lozano, J.A., eds.: Estimation of distribution algorithms: A new tool for evolutionary computation. Kluwer Academic Publishers, Boston, MA (2002)
7. Yu, T.-L., Goldberg, D.E., Yassine, A., Chen, Y.-p.: Genetic algorithm design inspired by organizational theory: Pilot study of a dependency structure matrix driven genetic algorithm. Proceedings of Artificial Neural Networks in Engineering 2003 (ANNIE 2003) (2003) 327–332 (Also IlliGAL Report No. 2003007).
8. Ocenasek, J., Schwarz, J., Pelikan, M.: Design of multithreaded estimation of distribution algorithms. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2003) (2003) 1247–1258
9. Sastry, K.: Evaluation-relaxation schemes for genetic and evolutionary algorithms. Master thesis, University of Illinois at Urbana-Champaign, Urbana, IL (2002)
10. Pelikan, M.: Bayesian optimization algorithm: From single level to hierarchy. Doctoral dissertation, University of Illinois at Urbana-Champaign (2002)
11. Smith, J., Fogarty, T.C.: Recombination strategy adaptation via evolution of gene linkage. Proceedings of the 1996 IEEE International Conference on Evolutionary Computation (1996) 826–831
12. Harik, G.R., Cantú-Paz, E., Goldberg, D.E., Miller, B.L.: The gambler's ruin problem, genetic algorithms, and the sizing of populations. Proceedings of 1997 IEEE International Conference on Evolutionary Computation (1997) 7–12
13. Harik, G.R.: Linkage learning via probabilistic modeling in the ECGA. IlliGAL Report No. 99010, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL (1999)

14. Mühlenbein, H., Schlierkamp-Voosen, D.: Predictive models for the breeder genetic algorithm: I. Continuous parameter optimization. Evolutionary Computation **1** (1993) 25–49
15. Thierens, D., Goldberg, D.E.: Convergence models of genetic algorithm selection schemes. In: Parallel Problem Solving fron Nature, PPSN III. (1994) 119–129
16. Miller, B.L.: Noise, sampling, and efficient genetic algorithms. Doctoral dissertation, University of Illinois at Urbana-Champaign, Urbana (1997)
17. Deb, K., Goldberg, D.E.: Analyzing deception in trap functions. Foundations of Genetic Algorithms 2 (1993) 93–108
18. Blickle, T., Thiele, L.: A mathematical analysis of tournament selection. Proceedings of the Sixth International Conference on Genetic Algorithms (1995) 9–16

Sexual and Asexual Paradigms in Evolution: The Implications for Genetic Algorithms

Mark W. Andrews[1] and Christopher Salzberg[2]

[1] Gatsby Computational Neuroscience Unit,
Alexandra House, 17 Queen's Square,
London WC1N 3AR, United Kingdom,
mark@gatsby.ucl.ac.uk,
[2] Sayama Lab, Department of Human Communication,
University of Electro-Communications,
Tokyo, Japan,
chris@cx.hc.uec.ac.jp

Abstract. In this paper, we generalize the models used by MacKay [1] in his analysis of evolutionary strategies that are based on sexual, rather than asexual, reproduction methods. This analysis can contribute to the understanding of the relative power of genetic algorithms over search methods based upon stochastic hill-climbing, e.g. [2], [3].

1 Introduction

The genome of individual i is denoted by $\mathbf{x}^{[i]}$, and each genome is of length G, i.e. $\|\mathbf{x}^{[i]}\| = G$. The set of all possible individuals in the species is denoted by $\mathcal{X}$ and $\|\mathcal{X}\| = 2^G$. We refer to the j^{th} bit on the genome of $\mathbf{x}^{[i]}$ by x^i_j, such that $\mathbf{x}^{[i]} \equiv [x^i_1, x^i_2, \ldots, x^i_G]$.

Fitness Function: The fitness of each genome $\mathbf{x}^{[i]}$ is assessed with a fitness function $\phi\colon \mathcal{X} \mapsto \mathbb{R}$,

$$\phi(\mathbf{x}^{[i]}) = \sum_{j=1}^{G/b} \Omega_j(\mathbf{x}^{[i]}), \tag{1}$$

where Ω_i is the parity of block i in the genome, and b is the number of variables per block.

Reproduction: Reproduction occurs by mutation alone, or by recombination and mutation. Mutation involves flipping each bit $x^{[i]}_j$ independently with a fixed probability m. In the case of sexual reproduction, an offspring $\mathbf{x}^{[k]}$ has parents $\mathbf{x}^{[i]}$ and $\mathbf{x}^{[j]}$. Each bit $x^{[k]}_i$ is chosen to be either $x^{[i]}_i$ or $x^{[j]}_i$, with equal probability.

Selection: At each generation, all individuals scoring above the 50^{th} percentile survive and reproduce to the extent that the population size stays constant at N.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 379–380, 2004.

2 Analysis

Using the parity fitness function described, the genome divides into G/b blocks. In the case where $b = 2$, we have $2^2 = 4$ possible block configurations: $B_0 = \{0,0\}, B_1 = \{0,1\}, B_2 = \{1,0\}, B_3 = \{1,1\}$. These configurations have different fitness values: $\Omega(B_0) = \Omega(B_3) = 1$, and $\Omega(B_1) = \Omega(B_2) = 0$. Let us denote $\mathrm{P}(B_0)$ by α and $\mathrm{P}(B_3)$ by β. We can assume that $\mathrm{P}(B_1) = \mathrm{P}(B_2)$, and denote both by γ. Given the initial probabilities of $\{\alpha, \gamma, \gamma, \beta\}$, the probabilities of the four possible configurations $\{B_0, B_1, B_2, B_3\}$ after sexual reproduction are given $\{\alpha', \gamma', \gamma', \beta'\}$ respectively, where

$$\alpha' = (\alpha + \gamma)^2,\ \beta' = (\beta + \gamma)^2,\ \gamma' = (\alpha + \gamma)(\beta + \gamma).$$

After reproduction, the probability of a good block is $\mathrm{p} = (\alpha' + \beta')$. The probability of k good blocks in the genome will be given by

$$\binom{\frac{G}{2}}{k} \mathrm{p}^k (1 - \mathrm{p})^{\frac{G}{2} - k}.$$

The mean and variance of this distribution are given by $\mu = \frac{G}{2}\mathrm{p}$ and $\sigma^2 = \frac{G}{2}\mathrm{p}(1-\mathrm{p})$, respectively. Selection chooses the upper-half of this binomial distribution, which will be positioned at approximately $\mu + \sqrt{\frac{2}{\pi}}\sigma$. Both α' and β' will be increased proportionally to the overall increase in fitness, i.e. by $\frac{1}{\mu}\left(\mu + \sqrt{\frac{2}{\pi}}\sigma\right)$. The proportions of B_0 and B_3 after selection, given by α'' and β'' respectively, will be as follows:

$$\alpha'' = \alpha' \left(\frac{\mu + \sqrt{\frac{2}{\pi}}\sigma}{\mu} \right) = (\alpha + \gamma)^2 \left(\frac{\mu + \sqrt{\frac{2}{\pi}}\sigma}{\mu} \right) \tag{2}$$

$$\beta'' = \beta' \left(\frac{\mu + \sqrt{\frac{2}{\pi}}\sigma}{\mu} \right) = (\beta + \gamma)^2 \left(\frac{\mu + \sqrt{\frac{2}{\pi}}\sigma}{\mu} \right). \tag{3}$$

As $\{\alpha'', \gamma'', \gamma'', \beta''\}$ must sum to 1,

$$\gamma'' = \frac{1 - (\alpha'' + \beta'')}{2}. \tag{4}$$

References

1. MacKay, D.J.C.: Information Theory, Inference and Learning Algorithms. Cambridge University Press (2003)
2. Mitchell, M., Holland, J.H., Forrest, S.: When will a genetic algorithm outperform hill climbing. In Cowan, J.D., Tesauro, G., Alspector, J., eds.: Advances in Neural Information Processing Systems. Volume 6., Morgan Kaufmann Publishers, Inc. (1994) 51–58
3. Baum, E.B., Boneh, D., Garrett, C.: Where genetic algorithms excel. In: Proceedings COLT 1995, Santa Cruz, California (1995)

Mutation Rates in the Context of Hybrid Genetic Algorithms

Seung-Hee Bae and Byung-Ro Moon

School of Computer Science & Engineering, Seoul National University
Shilim-dong, Kwanak-gu, Seoul, 151-742 Korea
{baesoong, moon}@soar.snu.ac.kr

Abstract. Traditionally, the mutation rates of genetic algorithms are fixed or decrease over the generations. Although it seems to be reasonable for classical genetic algorithms, it may not be good for hybrid genetic algorithms. We try, in this paper, the opposite. In the context of hybrid genetic algorithms, we raise the mutation rate over the generations. The rationale behind this strategy is as follows: i) The perturbation rate of crossover decreases over the generations as the chromosomes in the population become similar; ii) Local optimization algorithms can undo a considerable level of perturbation and return the offspring to one of the parents; iii) Thus, we rather need stronger mutation at a later stage of a hybrid genetic algorithm. Experimental results supported our strategy.

1 New Mutation Strategy

There have been a great number of studies on the desirable mutation rates in Genetic Algorithms (GAs). The previous studies, however, were mostly applied to simple GAs that do not use local optimization heuristics. This study starts from the fact that hybrid GAs have significantly different characteristics from simple GAs. Because GAs are weak in fine-tuning around local optima, it is less probable to improve, by mutation, the qualities of high-quality solutions than of low-quality solutions. Thus it is a reasonable approach, as in the non-uniform mutation, to lower the mutation rates in later generations of GAs. In hybrid GAs, however, the population consists of the same or mostly similar local optimum solutions at a later stage. In the case, the perturbation by crossover is lower than in populations with enough diversity. If we apply weak mutation in this situation, the local optimization engine would return the offspring to one of the parents (local optima) highly probably. Operators in a simple GA should sometimes be able to produce better offspring than parents, whereas operators in a hybrid GA do not have to produce better offspring than parents because they only have to provide initial solutions for a local optimization heuristic. Thus operators are free from fine-tuning in a hybrid GA and this allows more perturbation of solutions than in a simple GA.

This work was supported by Brain Korea 21 Project. The ICT at Seoul National University provided research facilities for this study

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 381–382, 2004.

From this viewpoint, we suggest a new mutation strategy that perturbs gradually more over time. We incorporate the new mutation method in hybrid GAs with LK local optimization heuristic for the traveling salesman problem (TSP) and compare the experimental results against fixed and non-uniform mutations.

We devised two methods that changes the mutation rates. The first method increases the mutation rate if 50% of the population convergences. After increasing mutation rates, we keep the increased mutation rate until the population convergence rate drops below 50%. When the population convergence rate drops below 50%, it returns to the initial mutation rate. In the second method, we increase the mutation rate at every increase of 10% starting at the convergence rate 50%. The mutation rate is determined as $r = (1+k) \cdot r_0, k = max\{0, \lfloor 1 + \frac{c-50}{10} \rfloor\}$ where r is current mutation rate, r_0 is the initial mutation rate, and c is the current rate of convergence. In this case, it returns to the initial mutation rate whenever the best individual of the population is updated. We call the first strategy "Binary Increase" and the second one "Gradual Increase."

2 Experimental Results

Table 1 shows the experimental results of the four different mutation strategies on four TSPLIB* benchmark problems with 1000 to 4461 cities using 5-point crossover. The column "Type" represents the mutation type. "Fixed", "Non-Uni", "Bin-Inc", and "Grd-Inc" mean Fixed Rate, Non-Uniform, Binary Increase, and Gradual Increase, respectively.

Table 1. The Experimental Results

Instance	Type	Best	Avg(%)	GSD	Time(Gen)
dsj1000 (18659688)	Fixed	**18659688**	18659957.45(0.001)	53.19	1358(8468)
	Non-Uni	**18659688**	18660082.53(0.002)	129.95	769(15000)
	Grd-Inc	**18659688**	**18659923.81(0.001)**	13.05	1519(8286)
	Bin-Inc	**18659688**	18659985.54(0.002)	96.99	1860(8230)
d2103 (80450)	Fixed	**80450**	80461.94(0.015)	7.53	666(5410)
	Non-Uni	**80450**	80463.96(0.017)	5.73	530(10000)
	Grd-Inc	**80450**	**80457.33(0.009)**	5.89	1294(6623)
	Bin-Inc	**80450**	80463.19(0.016)	5.12	1467(6880)
pcb3038 (137694)	Fixed	137698	137742.38(0.035)	5.06	1415(24873)
	Non-Uni	**137694**	137753.14(0.043)	5.70	658(30000)
	Grd-Inc	**137694**	137732.08(0.028)	2.63	1568(23658)
	Bin-Inc	**137694**	**137730.88(0.027)**	4.39	1755(22272)
fnl4461 (182566)	Fixed	**182584**	182684.67(0.065)	8.32	1386(38404)
	Non-Uni	182605	182713.13(0.081)	8.25	1112(50000)
	Grd-Inc	182601	**182667.37(0.056)**	6.18	1878(45939)
	Bin-Inc	182606	182682.97(0.064)	8.70	2896(49907)

The proposed mutation strategies overall showed improvement over the Fixed Rate strategy. For the pcb3038 and fnl4461, the improvements were within 5% statistical risk. For the dsj1000 and d2103, the improvements were not within the stable range. Non-uniform mutation was inferior to the others.

* http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/TSPLIB95/TSPLIB.html

Systematic Integration of Parameterized Local Search Techniques in Evolutionary Algorithms

Neal K. Bambha[1], Shuvra S. Bhattacharyya[1],
Jürgen Teich[2], and Eckart Zitzler[3]

[1] Department of Electrical and Computer Engineering, and
Institute for Advanced Computer Studies,
University of Maryland,
College Park, MD USA
{nbambha, ssb}@eng.umd.edu
[2] Computer Science Institute,
Friedrich-Alexander University,
Erlangen-Nuremburg
[3] Department of Computer Engineering
Swiss Federal Institute of Technology

Abstract. Application-specific, parameterized local search algorithms (PLSAs), in which optimization accuracy can be traded off with runtime, arise naturally in many optimization contexts. We introduce a novel approach, called simulated heating, for systematically integrating parameterized local search into evolutionary algorithms (EAs). Using the framework of simulated heating, we investigate both static and dynamic strategies for systematically managing the trade-off between PLSA accuracy and optimization effort. Our goal is to achieve maximum solution quality within a fixed optimization time budget. We show that the simulated heating technique better utilizes the given optimization time resources than standard hybrid methods that employ fixed parameters, and that the technique is less sensitive to these parameter settings. We demonstrate our techniques on the well-known binary knapsack problem and two problems in electronic design automation. We compare our results to the standard hybrid methods, and show quantitatively that careful management of this trade-off is necessary to achieve the full potential of an EA/PLSA combination.

1 Introduction

For many optimization problems, efficient algorithms exist for refining arbitrary points in the search space into better solutions. Such algorithms are called *local search algorithms* because they define neighborhoods, typically based on initial "coarse" solutions, in which to search for optima. Many of these algorithms are parameterizable in nature. Based on the values of one or more algorithm parameters, such a *parameterized local search algorithm (PLSA)* can trade off time or space complexity for optimization accuracy.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 383–384, 2004.

In this poster, we describe a technique called *simulated heating* [1], which systematically incorporates parameterized local search into the framework of global search. The idea is to increase the time allotted to each PLSA invocation during the optimization process—low accuracy of the PLSA at the beginning and high accuracy at the end. This is in contrast to most existing hybrid techniques, which consider a fixed local search function, usually operating at the highest accuracy.

We demonstrate our techniques on three applications—the well-known binary knapsack problem, a memory cost optimization problem in embedded systems, and a voltage scaling optimization problem for multiprocessor systems.

A simulated heating optimization begins with a low cost and accuracy for the PLSA. At certain points in time during the optimization the cost and accuracy are increased. The goal is to focus on the global search at the beginning and to find promising regions of the search space first; for this phase, the PLSA runs with low accuracy, which in turn allows a greater number of optimization steps of the global search. Afterward, more time is spent by the PLSA in order to improve the solutions found or to assess them more accurately. As a consequence, fewer global search operations are possible during this phase of optimization. Since the accuracy is systematically increased during the process, we use the term *simulated heating* for this approach by analogy to simulated annealing where the 'temperature' is continuously decreased according to a given cooling scheme.

Within the context of simulated heating optimization, we consider both static and dynamic strategies for systematically increasing the PLSA accuracy and the corresponding optimization effort. In the static optimization, the PLSA is varied at regular intervals. In the dynamic optimization, the PLSA is varied according to the overall progress of the optimization and the time remaining.

In most heuristic optimization techniques, there are some parameters that must be set by the user. In many cases, there are no clear guidelines on how to set these parameters. Moreover, the optimal parameters are often dependent on the exact problem specification. We show that the simulated heating technique, while still requiring parameters to be set by the user, is less sensitive to the parameter settings.

Our results for the three different applications show that, in the context of a fixed optimization time budget, simulated heating better utilizes the time resources and outperforms the standard fixed parameter hybrid methods. We show that careful management of this trade-off is necessary to achieve the full potential of an EA/PLSA combination, and to develop an efficient strategy for achieving this trade-off management.

References

1. N. K. Bambha, S. S. Bhattacharyya, J. Teich, and E. Zitzler, "Systematic integration of parameterized local search into evolutionary algorithms," *IEEE Transactions on Evolutionary Algorithms,* April 2004, to appear.

Comparative Molecular Binding Energy Analysis of HIV-1 Protease Inhibitors Using Genetic Algorithm-Based Partial Least Squares Method

Yen-Chih Chen[1], Jinn-Moon Yang[2], Chi-Hung Tsai[1], and Cheng-Yan Kao[1]

[1] Department of Computer Science and Information Engineering,
National Taiwan University, Taipei, Taiwan
{r91051, d90008, cykao}@csie.ntu.edu.tw
[2] Department of Biological Science and Technology & Institute of Bioinformatics,
National Chiao Tung University, Hsinchu, Taiwan
moon@cc.nctu.edu.tw

1 Introduction

Comparative molecular binding energy analysis (COMBINE) [1] is a helpful approach for estimation of binding affinity of congeneric ligands that bind to a common receptor. The essence of COMBINE is that the ligand-receptor interaction energies are decomposed into residue-based energy contributions, and then the partial least squares (PLS) analysis is applied to correlate energy features with biological activity. However, the predictive performance of PLS model drops with the increase of number of noisy variables. With regard to this problem genetic algorithm (GA) combined with PLS approach (GAPLS) [2] for feature selection has demonstrated the improvement on the prediction and interpretation of model. Therefore, the purpose of this paper is to derive a more accurate and more efficient GAPLS in COMBINE by introducing a number of successive refinements.

2 Methodology

The structure-activity data of forty-eight inhibitors of human immunodeficiency virus type I (HIV-1) protease analyze by Perez *et al.* [3] was used as a new data set for our methodology. According to COMBINE, the calculated ligand-receptor interaction energies in the refined complexes were partitioned on a per residue basis. The 48-by-396 data matrix was built with columns representing each of the interaction energy features and rows representing each inhibitor in the data set, and then only the 50 features with highest Mahalanobis distance are treated as significant features and retained to the further GAPLS analysis.

GAPLS is a sophisticated hybrid approach that combines GA as an efficient feature selection method with PLS as a robust statistical method. In accordance with the framework of GAPLS, a number of successive refinements which can be summarized as follows:

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 385–386, 2004.

(i) Weighted standard deviation error of predictions (*WSDEP*) is used as objective function to measure the internal predictability with respect to the selected features.

$$WSDEP = \sqrt{\frac{\sum \left(y_i - y_{pred,i} \right)^2}{N - lv - 1}} \left(\frac{100}{95} \right)^{lv} , \quad (1)$$

where y_i and $y_{pred,i}$ are the observed and predicted inhibitory activities belong to inhibitor i, N is the total number of samples and lv is the number of latent variables.

(ii) An extra bit lv, representing the number of latent variables, is appended to the original chromosome and expect GAPLS model to efficiently solve the problem of the optimum number of latent variables in PLS.

(iii) Post-ranking function is added to identify the most suitable chromosome with the least number of features and the best objective in the final population.

3 Results and Discussion

In order to demonstrate the performances of our model GAPLS and $\mathbf{GAPLS_{exp}}$, using 32 and 48 inhibitors in the training set, we quoted the models $\mathbf{C_{delphi}}$ and $\mathbf{C_{expanded}}$ from Perez *et al.* [3]. All the statistics of the models with respect to the predictability are listed in Table 1. GAPLS and $\mathbf{GAPLS_{exp}}$ provide not only a number of the most significant energy features but also the excellent predictability. Moreover, the selected amino acid residues are in a good agreement with the important binding sites in HIV-1 protease, and pinpoint the regions of significance in three-dimensional space where the actual ligand-receptor interactions involved.

Table 1. Comparison of the different regression models

Model	*Samples*	*Features*	*lv*	r^2	q^2	$SDEP_c$	$SDEP_{cv}$
C_{delphi}	32	47	2	0.90	0.73	0.69	0.59
$C_{expanded}$	48	54	2	0.91	0.81	0.66	-
GAPLS	32	16	1	0.87	0.87	0.48	0.49
$GAPLS_{exp}$	48	15	2	0.92	0.91	0.46	-

References

1. Ortiz, A.R., Pisabarro, M.T., Gago, F., Wade, R.C.: Prediction of Drug Binding Affinities by Comparative Binding Energy Analysis. J. Med. Chem. 38 (1995) 2681-2691
2. Hasegawa, K., Kimura, T., Funatsu, K.: GA Strategy for Variable Selection in QSAR Studies: Enhancement of Comparative Molecular Binding Energy Analysis by GA-Based PLS Method. Quant. Struct.-Act. Relat. 18 (1999) 262-272
3. Perez, C., Pastor, M., Ortiz, A.R., Gago, F.: Comparative Binding Energy Analysis of HIV-1 Protease Inhibitors: Incorporation of Solvent Effects and Validation as a Powerful Tool in Receptor-Based Drug Design. J. Med. Chem. 41 (1998) 836-852

Controlled Content Crossover: A New Crossover Scheme and Its Application to Optical Network Component Allocation Problem

Mohammad Amin Dallaali and Malin Premaratne

Advanced Computing and Simulation Laboratory (AXL)
Department of Electrical and Computer System Engineering
P.O. Box: 35 Monash University, Clayton Victoria 3800 Australia
Amin.Dallaali@eng.monash.edu.au

Abstract. In this paper, a new genetic mating scheme called Controlled Content Crossover (CCC) is proposed and applied to solve the optical network component allocation problem. In order to solve the constrained optimization problem, CCC finds the selected set of the components with the minimum cost while it keeps the total dispersion value of the answer within a double-sided limit. The simulation results show that CCC finds the optimum answer matching closely with CPLEX solution.

1 Introduction

Increasing the number of search branches in the search and optimization problems, conventional methods fail to find the answer in finite time. On the other hand, evolutionary computation schemes such as Genetic Algorithms (GAs) show promising performances. Genetic algorithm which was initially introduces by J. H. Holland in 1975 is adopted from the natural evolution of the best-fitted offspring when it is selected over the consequent generations [1]. Real world projects are mostly limited to the boundaries and therefore, constraints optimization problems are one of the major challenges that genetic algorithms are dealing with. There are several constraints handling methods discussed in the literature. For example a group of methods tries to code the search points so that they reserve the feasibility of the strings during the iterations [2]. In this paper the genetic algorithm is applied to solve the commercially important problem on optical network component allocation. The aim is to find the optimum selection of optical components with the minimum cost subject to the constraint that the total dispersion values of the selected components do not violate a double-sided limit. [3]

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 387–389, 2004.

2 The Algorithm

The main body of the algorithm consists of iterative genetic processors. The randomly generated initial population is sent into the iteration part. The final stable result is obtained after the sufficient turns of iterations. Mating, mutation and selection are the three main genetic operators. Mating or crossover is totally redesigned. The new proposed method is a bit-wise exchange phase that distributes the bits of the parents' string over the offspring. In the simulation, two parents are mated with each other and two offspring are obtained. The logic behind the distribution is that it tries to allocate the bits of parents to the offspring so that the total dispersion values of the obtained offspring maintain to be as equal as possible. In order to perform the process, there is an ordering phase before the crossover. The bits of strings are ordered based on the dispersion value that each of them contributes to the total dispersion value of the string. Therefore, a bit that represents a larger dispersion value is ordered with a higher rank. Afterward, the exchange phase starts and the first bit of the parents are randomly allocated to the offspring. For the next exchange turn, the total accumulated dispersion value of the offspring is calculated and the bit of the parents with the less dispersion value is allocated to the offspring that has more total dispersion value up to the previous exchange turn. This is of course intended in order to fairly distribute the dispersion values of the parents over the offspring.

3 Simulation and Results

A sample test data set consisting of 80 component types was used for the simulation. Figure 1 shows the convergence of the cost value over 200 iterations. It also compares the final stable value with the answer obtained from CPLEX software. The total dispersion values of the strings are shown in figure 2. For the last turns of iterations where the algorithm is in the steady state phase, the total dispersion values of the offspring are between the dispersion boundaries set up in the simulation.

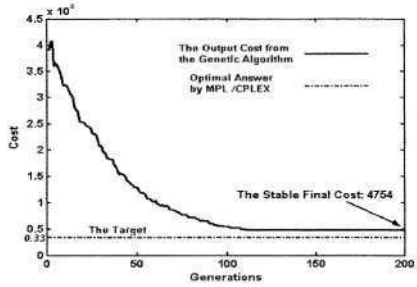


Fig. 1. The cost

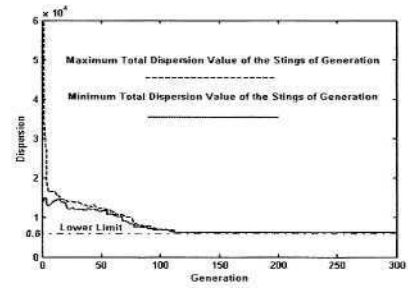


Fig. 2. The dispersion value

4 Conclusions

A new crossover scheme called Control Content Crossover (CCC) is proposed to solve the optical network component allocation problem. CCC is designed to perform the genetic crossover so that the total dispersion values of the obtained offspring fall into the constraint band. At the same time the algorithm finds the optimal solution that is the set of components with the minimal cost. The results showed that the performance of CCC is comparable with the result obtained by CPLEX and it fully satisfies the constraints.

References

[1] Goldberg, D. E.: Genetic algorithms in search, optimization, and machine learning. Addison-Wesley Pub. Co. (1989)

[2] Chu, P.C. and Beasley, J.E.: A Genetic Algorithm for Multidimentional Knapsack Problem, Journal of Heuristics, Vol. 4. Kluwer Academic Publisher (1998) 63-86

[3] Dallaali, M.A. and Premaratne, M.: Configuration of optical network using Genetic Algorithm, Proceeding of ONDM2004, (2004) 279-292

Efficient and Reliable Evolutionary Multiobjective Optimization Using ε-Dominance Archiving and Adaptive Population Sizing

Venkat Devireddy and Patrick Reed

Department of Civil and Environmental Engineering,
212 Sackett Building, University Park, PA-16802
vkd106@psu.edu, preed@engr.psu.edu

1 Introduction

This paper introduces a new algorithm, termed as the ε-NSGA-II that enables the user to specify the precision with which they want to quantify the Pareto optimal set and all other parameters are automatically specified within the algorithm. The development of the ε-NSGA-II was motivated by the steady state ε-MOEA developed by Deb et al [3]. The next section briefly describes the ε-NSGA-II.

2 Overview of the ε-NSGA-II Approach

The ε-NSGA-II is implemented using three primary steps. In the first step, the problem is solved using the original NSGA-II and parameters for crossover and mutation suggested by Deb [1]. The initial population size is set arbitrarily small (i.e., 5 in this paper) to ensure the algorithm's initial search is done using a minimum number of function evaluations. Subsequent increases in the population size adjust to the population size commensurate with problem difficulty. In the second step, the ε-NSGA-II uses a fixed sized archive to store the nondominated solutions generated in every generation of the NSGA-II runs. The archive is updated using the concept of ε-dominance, which has the benefit of ensuring that the archive maintains a diverse set of solutions. ε-dominance requires the user to define the precision with which they want to evaluate each objective by specifying an appropriate ε value for each objective. The third step checks if the user-specified performance and termination criteria are satisfied and the Pareto optimal set has been sufficiently quantified. If the criteria are not satisfied, the population size is doubled and the search is continued. When increasing the population, the initial population of the new run has solutions injected from the archive at the end of the previous run. The algorithm terminates if either a maximum user time is reached or if doubling the population size fails to significantly increase the number of nondominated solutions found across two runs.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 390–391, 2004.

3 Results and Conclusions

The efficiency of the proposed algorithm was tested of a suite of two objective test problems that are popular in literature [2]. The parameter settings are set exactly the same as specified by Deb [3], with the exception of the population size. To highlight the algorithms ability to reduce random seed effects, the problems were solved for 50 different random seeds using the same values of ε_i as specified by Deb et al. [3], and the results are presented in Table 1. The performance of the algorithm is measured using the convergence metric described in [3]. All the two objective problems solved by Deb et al. [3] required 20,000 function evaluations to obtain a set of 100 nondominated solutions.

Table 1. The comparison of the results between ε-NSGA-II and ε-MOEA[1] across various performance measures. Reliability is shown in terms of the number of runs that satisfied convergence criteria (α).

Problem	Nfe	No. of Solutions	ε-NSGA-II Convergence	ε-MOEA [see 1] Convergence	Successful runs(α)
ZDT1	5213	95	0.000384836	0.00039545	50(0.01)
ZDT2	5255	96	0.000407489	0.00046448	50(0.01)
ZDT3	5922	97	0.00180699	0.00175135	50(0.01)
ZDT4	8238	93	0.00048264	0.00259063	50(0.01)
ZDT6	8458	93	0.02618	0.067928	50(0.1)
DTLZ2	13842	86	0.045569	0.01080443	48(0.1)

It can be seen from the results that the algorithm required at least 60% fewer function evaluations than prior methods ([2], [3]). Also the algorithm has shown that it has been able to reliably converge to the true Pareto front in more than 96% of the runs.

References

1. Deb, K., *Multi-Objective Optimization using Evolutionary Algorithms.* 2001, New York, NY: John Wiley & Sons LTD.
2. Zitzler, E., K. Deb, and L. Thiele, *Comparison of multiobjective evolutionary algorithms: Empirical results.* Evolutionary Computation, 2000. **8**(2): p. 125-148.
3. Deb, K., M. Mohan, and S. Mishra, *A Fast Multi-objective Evolutionary Algorithm for Finding Well-Spread Pareto-Optimal Solutions.* 2003, Indian Institute of Technology: Kanpur, India.

[1] These results are taken from Deb et al. [3].

Heuristic Methods for Solving Euclidean Non-uniform Steiner Tree Problems

Ian Frommer[1]*, Bruce Golden[2], and Guruprasad Pundoor[2]

[1] Department of Mathematics, University of Maryland, College Park, MD 20742
[2] R.H. Smith School of Business, University of Maryland, College Park, MD 20742

Abstract. In this paper, we consider a variation of the Euclidean Steiner Tree Problem in which the space underlying the set of nodes has a specified non-uniform cost structure. This problem is significant in many practical situations, such as laying cable networks, where the cost for laying a cable can be highly dependent on the location and the nature of the area through which it is to be laid. We empirically test the performance of a genetic-algorithm-based procedure on a variety of test cases of this problem. We also consider the impact on solutions of charging an additional fee per Steiner node. This can be important when Steiner nodes represent junctions that require the installation of additional hardware. In addition, we present novel ways of visualizing the performance and robustness of the genetic algorithm.

The minimal spanning tree (MST) problem deals with connecting a given set of nodes in a graph in a minimal cost way. In a Steiner tree, we are allowed to use additional nodes if doing so reduces the total cost. Steiner tree problems find applications in various areas such as the design of communication networks, printed circuits, and the routing of transmission lines. There are many versions of the Steiner tree problem. The most common is the Euclidean Steiner tree problem, in which a given set of nodes is connected in Euclidean space. Another version involves rectilinear configurations, in which the arcs have to be either North-South or East-West. Modified versions of this include the hexagonal and octagonal grids [1,2].

We consider the problem of finding near-optimal, non-directed Steiner trees on a non-uniform grid. Each location in the grid has an associated cost and a given set of nodes has to be connected in the form of a spanning tree. This kind of problem is of relevance to many practical situations, such as laying cable networks. There may be regions where laying cable is prohibitively expensive, such as a prime location in a metropolitan area, or other regions where it is cheaper. So the objective is to find a set of additional nodes in the grid that can be used to connect all the given nodes at a minimum cost. The performance of our genetic algorithm (GA) is illustrated in Fig. 1

Our GA uses a fairly simple encoding (lists of Steiner node coordinates) and simple operators: spatial crossover, queen bee selection, and random mutation. It

* Corresponding author. Email: orbit@glue.umd.edu

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 392–393, 2004.

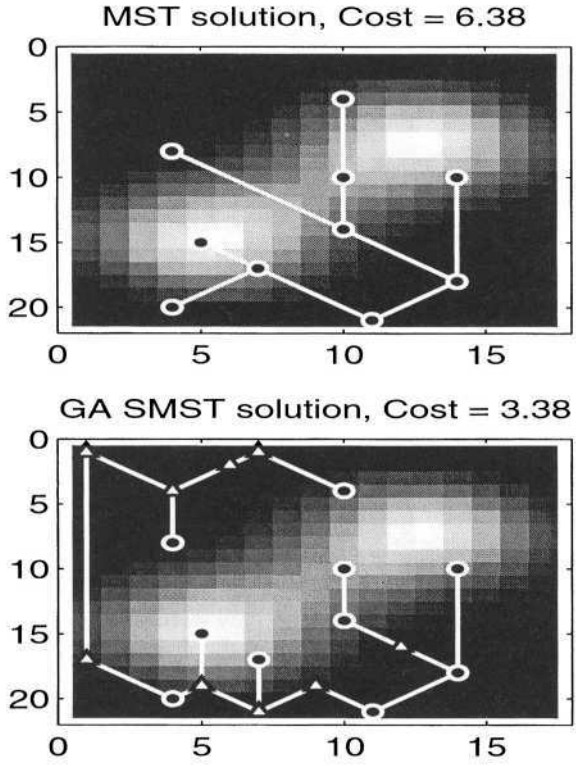


Fig. 1. MST (top) and Steiner MST (SMST) (bottom) for a sample problem. Black circles indicate terminal nodes. White triangles indicate Steiner nodes. The lighter the shading, the more costly the cell.

finds solutions significantly better than those of the minimal spanning tree and comparable to those of an enumerative algorithm we designed called progressive addition (PA). We point out, however, that the GA scales better with increasing problem size than the PA. The solution trees are able to correctly avoid high cost areas while finding low cost regions.

References

1. Coulston, C.: Constructing exact octagonal Steiner minimal trees. In: Proceedings of the 13th ACM Great Lakes Symposium on VLSI 2003, Washington, DC, USA, ACM (2003) 1–6
2. Coulston, C., Smith, M., Werger, K.: Ants and evolution: The non-uniform Steiner minimal tree problem. In: Artificial Neural Networks in Engineering 2002, St. Louis, Missouri, USA, ASME (2002) 213–218

Automating Evolutionary Art in the Style of Mondrian

Andrés Gómez de Silva Garza and Aram Zamora Lores

Departamento Académico de Computación, Instituto Tecnológico Autónomo de México (ITAM), Río Hondo #1, Colonia Tizapán-San Ángel, 01000—México, D.F., Mexico
agomez@itam.mx, aram_n1@hotmail.com

1 Introduction

Many recent systems for computer artwork generation with evolutionary algorithms have been interactive, relegating the task of evaluating new genotypes to their users. In contrast, we are interested in fully automating the entire evolutionary cycle applied to art work generation. We set out to produce a program that would generate artwork in the style of the Dutch painter Piet Mondrian, who was active mainly in the first half of the 20th century. He developed his own distinctive and abstract style (called simply *de stijl,* which is Dutch for "the style"). Paintings in Mondrian's style typically include vertical and horizontal black lines over a white background, with some or all of the primary colors (blue, red, and yellow), plus black, filling in some of the square or rectangular regions (or parts of the regions) separated out from the background by the black lines. It is this style that our system tries to emulate. Fig. 1 shows a typical Mondrian painting in his distinctive style.

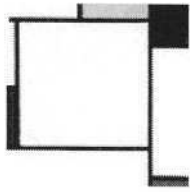


Fig. 1. A typical Mondrian painting

We decided to use an evolutionary algorithm in order to generate different paintings. After being generated by the evolutionary operators of crossover and mutation, paintings are evaluated to determine how much they "make sense." In the context of our research, "making sense" would imply being as close as possible to the style of Mondrian. The evaluation subroutine of the evolutionary algorithm, therefore, is of critical importance to the success of the approach. It consists of the following eight rules (each of which has the same weight), which capture and recognize the stylistic characteristics of Mondrian's paintings:

1. EvaluateColor: Each colored region that is contained in a genotype must have one of the five valid colors.
2. EvaluateCoordinates: The height, width, x-coordinate, and y-coordinate of each colored region in a genotype must all fall between 0 and 3.9999.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 394–395, 2004.

3. EvaluateLineThickness: Up to two black colored regions are allowed in a genotype that are not thin, but all other black regions must be either vertically or horizontally thin (and thus represent a line rather than a rectangular region).
4. EvaluateNumberOfVerticalLines: A minimum of two and a maximum of ten vertical lines must be present in a genotype.
5. EvaluateNumberOfHorizontalLines: A minimum of two and a maximum of ten horizontal lines must be present in a genotype.
6. EvaluateLimits: Each colored region in a genotype must be adjacent either vertically (both above and below) or horizontally (both to the left and to the right), or both, to another colored region or to the edge of the "frame" (with some small tolerance).
7. EvaluateFrame: All other colored regions in a genotype must fall within the coordinates of the "frame," whose color is white by definition and whose coordinates are represented just as any other colored region's are.
8. EvaluateNumberOfColoredRegions: There must be at least one colored region represented in a genotype, and at most 13, not counting lines. At most one of them can be white (and represents the "frame").

2 Discussion and Results

The implementation of this work has resulted in a system named MONICA (MONdrian-Imitating Computer Artist). Neither the relationship between evolutionary algorithms and Mondrian, nor the attempt to capture and automate the generation of new creations in the style of given artists or designers, is new. However, the approach we followed in implementing MONICA is different from those that have been used in other projects that have explored these issues. Fig. 2 shows three Mondrian-like paintings which were created by our system (which assigned a fitness value of 1 to them) at different times. The fact that MONICA has shown the capacity to create Mondrian-like paintings on several occasions validates the set of evaluation rules we programmed into the system and also confirms the capacity to create artwork by computer autonomously, without the need for user feedback or intervention.

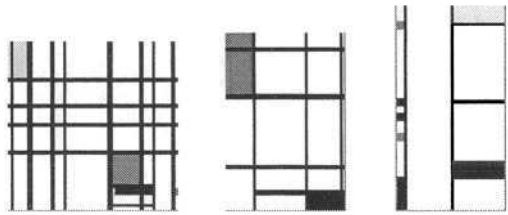


Fig. 2. Three Mondrian-like paintings generated by MONICA

Mutation Can Improve the Search Capability of Estimation of Distribution Algorithms

Hisashi Handa

Okayama University, Tsushima-Naka 3-1-1,
Okayama 700-8530, JAPAN,
handa@sdc.it.okayama-u.ac.jp,
http://www.sdc.it.okayama-u.ac.jp/~handa/index-e.html

Abstract. The Estimation of Distribution Algorithms are a class of evolutionary algorithms which adopt probabilistic models to reproduce the genetic information of the next generation, instead of conventional crossover and mutation operations. In this paper, we propose new EDAs which incorporate mutation operator to conventional EDAs in order to keep the diversities in EDA populations. Experiments results shown in this paper confirm us the effectiveness of the proposed methods.

1 Introduction

Recently, Estimation of Distribution Algorithms (EDAs) have been attracted much attention in genetic and evolutionary computation community due to their search abilities [1]. Genetic operators such like crossover and mutation are not adopted in the EDAs. In the EDAs, a new population is generated from the probabilistic model constituted by a database containing the genetic information of the selected individuals in the current generation. Such reproduction procedure by using the probabilistic model makes EDAs more powerful search algorithms. However, it significantly decreases the diversity of the genetic information in the generated population when the population size is not large enough. Hence, EDAs with small population size are often trapped into premature convergences. In this paper, we discuss on the effectiveness of mutation in the case of EDAs. We propose new EDAs which incorporate mutation operator to conventional EDAs in order to keep the diversities in EDA populations.

2 Estimation of Distribution Algorithms with Mutation

In this paper, we incorporate a mutation operator into EDAs in a simple way: For all M sampled individuals, the mutation operation is performed. We adopt the bitwise mutation operator, which is the same as mutation operation in SGA, as the mutation operator of the proposed method. That is, at all loci for all M sampled individuals, whether bit-flip will be held or not is probabilistically decided by referring to the mutation probability. The reason why we adopt such simple way is that we can apply the notion of the proposed method into any kinds of

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 396–397, 2004.

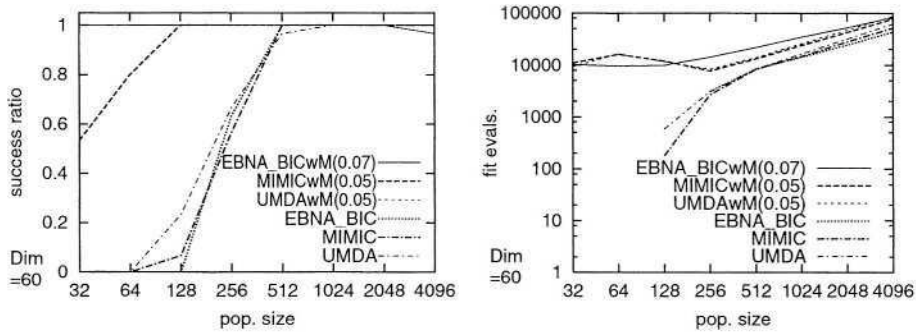


Fig. 1. Experimental results of Fc_4 function: SR (LEFT), NOE (RIGHT); dim = 60

EDAs. In this paper, the proposed methods for UMDA, MIMIC, and EBNA_{BIC} are referred as UMDAwM, MIMICwM, and $\mathrm{EBNA}_{BIC}\mathrm{wM}$, respectively.

3 Experimental Results

We examine Fc_4 function for comparing the proposed methods with corresponding conventional methods [2]. The number of individuals *M* is set to be one of 32, 64, 128, 256, 512, 1024, 2048, and 4096, for each problem instance. The number of selected individuals is set to be half of the number of individuals *M*. We use the truncation selection method to constitute the selected individuals. In this paper, we adopt two indices to evaluate the effectiveness of algorithms: success ratio (SR) and the number of fitness evaluations until finding optimal solutions (NOE). The NOE in this paper is averaged value over "success" runs.

Fig. 1 shows the experimental results of the Fc_4 function. The conventional methods whose population size is less than 512 could not solve the Fc_4 function well. On the other hand, our proposed methods, i.e., UMDAwM and $\mathrm{EBNA}_{BIC}\mathrm{wM}$, could solved that function for all population sizes. The NOE of our proposed methods is grater than the corresponding conventional methods if the population size of both methods is the same. However, the lowest NOE in the all experiments whose SR = 1 is shown by the MIMICwM with population size = 256.

References

1. P. Larrañaga and J. A. Lozano Editors: *Estimation of Distribution Algorithms,* Kluwer Academic Publishers (2002)
2. J. S. De Bonet *et al.:* MIMIC: Finding optima by estimating probability densities, *Advances in Neural Information Processing Systems* **9** (1996)

Neural Network Normalization for Genetic Search*

Jung-Hwan Kim, Sung-Soon Choi, and Byung-Ro Moon

School of Computer Science and Engineering
Seoul National University
Shilim-dong, Kwanak-gu, Seoul, 151-742 Korea
{aram,irranum,moon}@soar.snu.ac.kr

An arbitrary neural network has a number of functionally equivalent other networks. This causes redundancy in genetic representation of neural networks, which considerably undermines the merit of crossover in GAs [1]. This problem has received considerable attention in the past and has also been called the "competing conventions" problem [2].

We transform each neural network to an isomorphic neural network to maximize the genotypic consistency of two parents. We aim to develop a better genetic algorithm for neural network optimization by helping crossover preserve common functional characteristics of the two parents. This is achieved by protecting "phenotypic" consistency and, consequently, preserving building blocks with promising schemata.

Given a neural network, let N_i, N_h, and N_o be the numbers of input neurons, hidden neurons, and output neurons in the network, respectively. We denote a neural network $\mathfrak{N}$ by $\mathfrak{N} = \{\mathbf{h}_1, \ldots, \mathbf{h}_J\}$, where $\mathbf{h}_j = [w^i_{j1}, \ldots, w^i_{jN_i}, w^h_{1j}, \ldots, w^h_{N_h j}, w^o_{1j}, \ldots, w^o_{N_o j}]^T$ and w^i_{jk}, w^h_{kj}, and w^o_{kj} are the synaptic weights from input neuron k to hidden neuron j, from hidden neuron j to hidden neuron k, and from hidden neuron j to output neuron k, respectively. Let S_J be the set of all the permutations of the set {1,..., J} where J is the number of hidden neurons. We define the neural network isomorphism as follows:

Definition 1. *For two neural networks* $\mathfrak{N} = \{\mathbf{h}_1, \ldots, \mathbf{h}_J\}$ *and* $\mathfrak{N}' = \{\mathbf{h}'_1, \ldots, \mathbf{h}'_J\}$, $\mathfrak{N}$ *is* isomorphic *to* $\mathfrak{N}'$ ($\mathfrak{N} \simeq \mathfrak{N}'$) *if and only if there exists a permutation* $p \in S_J$ *such that* $\mathfrak{h}_{p(j)} = \mathfrak{h}'_j \ \forall j = 1, \ldots, J$.

From the definition, two isomorphic neural networks are constructed essentially in the same way. In other words, a neural network can be transformed into another isomorphic neural network by appropriate permutation of the hidden neurons.

We transform one of the parents in relation to the other so that high-quality schemata are well preserved and combined. We call such a transformation *normalization*. More formally, let $\mathcal{N}$ be the set of the neural networks and $\mathcal{N}_{\mathfrak{N}}$ be the set of the networks that are isomorphic to a network $\mathfrak{N} \in \mathcal{N}$. Suppose a distance measure $\mathfrak{d} : \mathcal{N} \times \mathcal{N} \to \mathbb{R}$ defined on a pair of networks that measures the genotypic distance of the two networks. Given parents $\mathfrak{N}, \mathfrak{M} \in \mathcal{N}$, the normalization

* This work was supported by Brain Korea 21 Project. The ICT at Seoul National University provided research facilities for this study.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 398–399, 2004.

Table 1. Comparison with Several Normalization

	WBCD	CHDD	TDD
Normal	96.47 (96.91)	80.61 (82.74)	94.54 (94.72)
$\mathfrak{d}_1$_2Opt	96.78 (97.42)	82.03 (85.79)	95.19 (96.44)
$\mathfrak{d}_1$_KL	96.72 (97.42)	82.16 (84.26)	95.16 (96.47)
$\mathfrak{d}_2$_HM	96.89 (97.42)	82.07 (83.82)	94.83 (94.97)
$\mathfrak{d}_3$_2Opt	97.03 (97.43)	83.36 (85.15)	95.65 (96.53)
$\mathfrak{d}_3$_KL	97.21 (97.60)	84.26 (85.83)	95.77 (96.85)

operator transforms $\mathfrak{M}$ to $\mathfrak{M}' \in \mathfrak{N}_{\mathfrak{M}}$ such that $\mathfrak{d}(\mathfrak{N}, \mathfrak{M}')$ is minimal among all the networks in $\mathfrak{N}_{\mathfrak{M}}$.

For two neural networks $\mathfrak{N}$ and $\mathfrak{N}'$, we propose three distance measures. Among these, the first measure ($\mathfrak{d}_1$) uses Euclidean distance based on the weights of the connections. The other two measures are based on the degrees of learning of the networks. These measures are related to our assumption that sufficiently learned neurons constitute high-quality schemata. $\mathfrak{d}_2$ uses the degrees of learning of the individual hidden neurons and $\mathfrak{d}_3$ uses the degrees of learning of the individual connections.

We devised a Kernighan-Lin-style heuristic for the normalization which is fast but suboptimal to some degree. The heuristic iteratively improves the permutation in terms of the distance measure by exchanging pairs of the hidden neuron indices in the manner of sequential 2-Opt.

We selected three well known datasets from UCI repository: Thyroid Disease Database (TDD), Wisconsin Breast Cancer Database (WBCD), and Cleveland Heart Disease Database (CHDD). The effects of the proposed normalization approaches are examined with the above databases.

Table 1 shows the classification results. In the table, "Normal" represents the results of the neuro-genetic hybrid without normalization. $\mathfrak{d}_i$_B denotes the neuro-genetic hybrid with normalization by method B on distance measure $\mathfrak{d}_i$. "2Opt" and "KL" denotes the heuristics of 2-Opt and Kernighan-Lin style, respectively. "HM" indicates the Hungarian method and uses the learning degree of the hidden neurons.

The two values in each experiment show the mean and the best classification results, respectively, from 50 trials. The normalization methods for the neural network overall showed improvement over the one without normalization. Among the five methods of normalization, the method $\mathfrak{d}_3$_KL showed the best results. This tendency was consistent in all of the three test problems.

References

1. J. D. Schaffer, D. Whitley, and L. J. Eshelman. Combinations of genetic algorithms and neural networks: A survey of the state of the art. In *International Workshop on Combinations of Genetic Algorithms and Neural Networks,* pages 1–37, 1992.
2. D. Thierens. Non-redundant genetic coding of neural networks. In *IEEE International Conference on Evolutionary Computation,* pages 571–575, 1996.

Distance Measures in Genetic Algorithms*

Yong-Hyuk Kim and Byung-Ro Moon

School of Computer Science & Engineering, Seoul National University
Shillim-dong, Kwanak-gu, Seoul, 151-742 Korea
{yhdfly, moon}@soar.snu.ac.kr

Metric is one of the fundamental tools for understanding space. It gives induced topology to the space and it is the most basic way to provide the space with topology. Different metrics make different topologies. The shape of the space largely depends on its metric. In understanding genetic algorithms, metric is also basic and important. In genetic algorithms, a good distance measure not only helps to analyze their search spaces, but can also improve their search capability. Hamming distance has been popular in most researches for genetic algorithms that deal with discrete spaces. It has also been widely adopted in studies about the analysis of the problem space. In this paper, we propose more reasonable distance measures depending on situations in the process of genetic algorithms and show that they are actually metrics. We propose three distance measures: one for the population-based search, another for the solution space based on K-ary encoding, and the third as an approximate measure of performance improvement of linkage-based genetic algorithms.

Since the genetic algorithm is a population-based search, the distance measure between populations is useful for understanding the behavior of genetic algorithms. We propose an intuitive and reasonable metric.

Definition 1 *Let K be the population size. Let population $\mathfrak{p} = \{\mathfrak{c}_1, \mathfrak{c}_2, \ldots, \mathfrak{c}_K\}$ and population $\mathfrak{p}' = \{\mathfrak{c}'_1, \mathfrak{c}'_2, \ldots, \mathfrak{c}'_K\}$. Given a metric $\mathfrak{d}$ in solution space, we define the distance D_K of the two populations as follows:*
$D_K(\mathfrak{p}, \mathfrak{p}') := min_{\sigma \in \Sigma_K} \left(\sum_{i=1}^{K} \mathfrak{d}(\mathfrak{c}_i, \mathfrak{c}'_{\sigma(i)}) \right)$ *where σ denotes a permutation.*

Theorem 1 *D_K is a metric in the population space.*

Then, it can be computed by the Hungarian method. In the assignment weight matrix $M = (m_{ij})$ between two populations $\mathfrak{p}$ and $\mathfrak{p}'$, each element m_{ij} represents $\mathfrak{d}(\mathfrak{c}_i, \mathfrak{c}'_j)$. The problem of computing D_K is exactly the problem of finding an assignment (permutation) with minimum summation.

We propose a distance measure for disjoint K-grouping problems. In these problems, since each group is not distinguishable, each solution has $K!$ representations. This makes the Hamming distance between two solutions unrealistic and undermines the effectiveness of crossover operators.

Definition 2 *Let the universal set U be $\{1, 2, \ldots, K\}^N$, where N is the problem size. Given two K-ary encodings $\mathfrak{a}, \mathfrak{b} \in U$ and a metric $\mathfrak{d}$ in U, we define the distance measure d_K for K-grouping problem as follows: $d_K(\mathfrak{a}, \mathfrak{b}) :=$*

* This work was supported by Brain Korea 21 Project. The ICT at Seoul National University provided research facilities for this study.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 400–401, 2004.

$min_{\sigma \in \Sigma_K}(\mathfrak{d}(\mathfrak{a}, \mathfrak{b}_\sigma))$ where σ is a permutation and $\mathfrak{b}_\sigma$ is a permuted encoding of $\mathfrak{b}$ by σ; i.e., i^{th} element e_i of $\mathfrak{b}$ is transformed into $\sigma(e_i)$.

Theorem 2 *d_K is a pseudo-metric in U. Moreover, suppose that Q is the quotient set of U by relation $\sim$ [1] (i.e., $Q = U/\sim$). Then, (Q, d_K) is a metric space; i.e., d_K is a metric in Q.*

When the metric $\mathfrak{d}$ is the Hamming distance $\mathfrak{H}$, the problem of computing d_K is also formulated as the optimal assignment problem. Hence, it can be computed by the Hungarian method. In the assignment weight matrix $M = (m_{ij})$ between two chromosomes X and Y, each element m_{ij} means $\sum_{k=1}^{N} I(X_k = i, Y_k \neq j)$, where N is the length of chromosome and $I(\cdot)$ is the indicator function. The problem of computing d_K is exactly the problem of finding an assignment (permutation) with minimum summation. Since the normalizations of chromosomes pursue the minimization of genotype inconsistency among chromosomes, the proposed metric is ideal in this line of work.

Theorem 3 *If the metric $\mathfrak{d}$ is the Hamming distance $\mathfrak{H}$, then $d_K(X, Y) = min_{\sigma \in \Sigma_K} \left(\sum_{i=1}^{K} \sum_{k=1}^{N} I(X_k = i, Y_k \neq \sigma(i)) \right)$.*

The first level distance measure is commonly the Hamming distance. Other distance measures can also be used as the first level distance (e.g., normalized Hamming distance). We define the second level distance measure. It is defined from the first level distance. Given the problem instance p, consider the graph G_p representing the first order gene interaction; i.e., representing only gene interactions between a pair of genes. Let A_p be the adjacency matrix of G_p.

Definition 3 *Suppose that there exists the inverse of $A_p \oplus I$. We define the second level distance measure D_p as follows: $D_p(\mathfrak{a}, \mathfrak{b}) := \|(A_p \oplus I)^{-1}(\mathfrak{a} \oplus \mathfrak{b})\|$ where $\oplus$ is XOR operator and $\|\cdot\|$ is a norm derived from the first level distance $\mathfrak{d}$ (i.e., $\|\cdot\| = \mathfrak{d}(\cdot, 0)$).*

Theorem 4 *D_p is a metric.*

The second level distance and its extension are efficiently computed in $O(N^3)$ by a variant of Gauss-Jordan elimination method. In genetic algorithms for graph partitioning, gene rearrangement shows dramatic performance improvement on some graphs. The fitness-distance correlation using the proposed second level distance identified the graphs that benefited most by gene rearrangement in genetic algorithms.

Most previous studies needing distances among chromosomes in genetic algorithms used the Hamming distance. The purpose of this paper is to develop more meaningful distance measures for genetic algorithms. We hope that the proposed metrics are useful for improving GA's search capability and understanding GA's working mechanism.

[1] Given an element $\mathfrak{a} \in U$, since $\mathfrak{d}$ is a metric, there are only $K!$ elements such that the distance d_K to $\mathfrak{a}$ is zero. If the distance d_K between two elements is equal to zero, we define them to be *in relation* $\sim$. Then, the relation $\sim$ is an equivalence relation.

Analysis of a Parallel MOEA Solving the Multi-objective Quadratic Assignment Problem

Mark P. Kleeman, Richard O. Day, and Gary B. Lamont

Air Force Institute of Technology, Dept of Electrical and Computer Engineering,
Graduate School of Engineering & Management,
Wright-Patterson AFB (Dayton) OH, 45433, USA
{Mark.Kleeman, Richard.Day, Gary.Lamont}@afit.edu
http://www.afit.edu

The Quadratic Assignment Problem (QAP) is an NP-Complete problem [1]. The multiobjective Quadratic Assignment Problem (mQAP) is the multiobjective version of the QAP and was formalized in 2002 [2]. The QAP has had extensive research, but mQAP research is still in its infancy. The mQAP has been been used to optimize communication for formations of heterogenous unmanned aerial vehicles (UAVs) through the use of the Multi-Objective Messy Genetic Algorithm - II (MOMGA-II) [3]. This research extends that research by using a parallelized version of MOMGA-II and comparing the speedup and efficiency of the parallel results to the serial results.

The mQAP is defined in mathematical terms in equations 1 and 2

$$minimize\{C(\pi)\} = \{C^1(\pi), C^2(\pi), \ldots, C^m(\pi)\} \tag{1}$$

where

$$C^k(\pi) = \underset{\pi \in P(n)}{min} \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} b^k_{\pi_i \pi_j}, k \in 1..m \tag{2}$$

and where n is the number of objects/locations, a_{ij} is the distance between location i and location j, b^k_{ij} is the kth flow from object i to object j, π_i gives the location of object i in permutation $\pi \in P(n)$, and *minimize* means to find all non-dominated points [2].

These experiments compared the parallel results with the serial results. The parallel model used in this research is the island model. See [3] for the environment the experiments were run.

The metrics used for the parallel experiments are speedup and efficiency. Speedup is chosen for its commonality throughout the literature and its ability to show how much faster or slower the parallel processing is compared to serial processing. Efficiency is used to show the amount of time the processing element is used vs. its idle time. Table 1 lists the results of the speedup and efficiency analysis. By using more processors, the speedup is increased enabling the runs to be done in a more efficient manner. The efficiency for each processor decreases as more processors are added. This means the processors are in idle mode more often and that scalability problems can occur as more processors are added.

Figure 1 shows a graph of the mean time to finish an experiment with set population sizes. This indicates that as more processors are added to search for

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 402–403, 2004.

Table 1. Speedup and Efficiency Results

Number of Processors	Speedup	Efficiency
2 Processors	1.842	0.9210
4 Processors	2.376	0.5941
8 Processors	3.431	0.4289

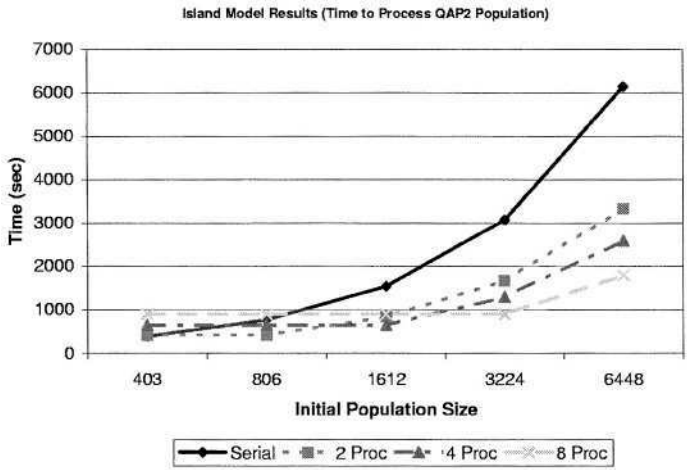


Fig. 1. Speedup results from running MOMGA-II on data set KC10-2fl-1uni.

solutions there is almost a linear speedup when compared to running the same number of searches in serial.

References

1. Çela, E.: The Quadratic Assignment Problem - Theory and Algorithms. Kluwer Academic Publishers, Boston, MA (1998)
2. Knowles, J., Corne, D.: Instance generators and test suites for the multiobjective quadratic assignment problem. In Fonseca, C., Fleming, P., Zitzler, E., Deb, K., Thiele, L., eds.: Evolutionary Multi-Criterion Optimization, Second International Conference, EMO 2003, Faro, Portugal, April 2003, Proceedings. Number 2632 in LNCS, Springer (2003) 295–310
3. Day, R.O., Kleeman, M.P., Lamont, G.B.: Solving the Multi-objective Quadratic Assignment Problem Using a fast messy Genetic Algorithm. In: Congress on Evolutionary Computation (CEC'2003). Volume 4., Piscataway, New Jersey, IEEE Service Center (2003) 2277–2283

Evolving Features in Neural Networks for System Identification

Yung-Keun Kwon and Byung-Ro Moon

School of Computer Science & Engineering, Seoul National University
Shilim-dong, Kwanak-gu, Seoul, 151-742 Korea
{kwon, moon}@soar.snu.ac.kr

Given N data pairs $\{X_i, y_i\}$, $i = 1, 2, ..., N$, where each X_i is an n-dimensional vector of independent variables ($X_i =< x_{i_1}, x_{i_2}, \ldots, x_{i_n} >$) and y_i is a dependent variable, the function approximation problem (FAP) is finding a function that best explains the N pairs of X_i and y_i. From the universal approximation theorem and inherent approximation capabilities proved by various researches, artificial neural networks (ANNs) are considered as powerful function approximators. There are two main issues on the feedforward neural networks' performance. One is to determine its structure. The other issue is to specify the weights of a network that minimizes its error. Genetic algorithm (GA) is a global search technique and is useful for complex optimization problems. So, it has been considered to have potential to reinforce the performance of neural networks. Many researchers tried to optimize the weights of networks using genetic algorithms alone or combined with the backpropagation algorithm. Others also tried to find a good topology that is even more difficult and called a "black art."

In this paper, instead, we use a genetic algorithm to evolve the input space. That is, a chromosome represents the meanings of input nodes. It generates a new input space using hidden neurons that play a critical role in the learning because they act as *feature detectors.* They gradually discover the salient features that characterize the training data. We try to find useful input features using a genetic algorithm. A chromosome represents the meaning of the input nodes. A chromosome consists of an array of I functions.

Roulette-wheel selection is used for parent selection. A crossover operator creates a new offspring chromosome by choosing some features from two parent chromosomes. Since each neural network has I input nodes and H hidden nodes, there are totally $2(I+H)$ candidate features to be chosen for offspring. Crossover randomly chooses I features from the set of candidates. This process is shown in Figure 1. The neural network then takes the features in the offspring as the input nodes. The offspring first attempts to replace the inferior out of the two parents. If it fails, it attempts to replace the most inferior member of the population. It stops when there is no improvement during a given number of generations.

We attack a critical heat flux (CHF) function approximation problem which is important for the safe and economic design of many heat transfer units including nuclear reactors, fossil-fuel boilers, fusion reactors, electronic chips, etc. Each data set consists of eight independent variables and one dependent variable. We were given 1607 sets of observed data from KAERI (Korea Atomic Energy

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 404–405, 2004.

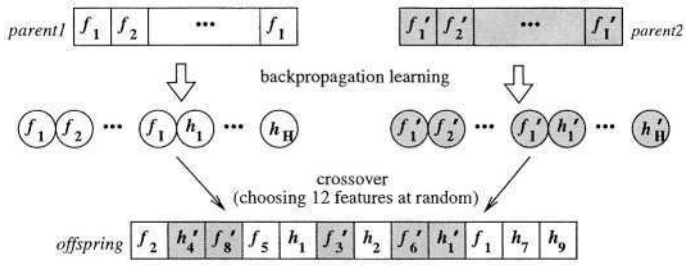


Fig. 1. Crossover process

Table 1. Performance Comparison in Terms of *LRE*

	KAERI	PLS	WGA	FEGA
$E(y/\hat{f}(X))$	1.002671	1.002562	1.006147	1.010703
$\sigma(y/\hat{f}(X))$	0.107265	0.121299	0.098247	0.094624
LRE	0.106979	0.120989	0.097647	0.093622

Research Institute). To evaluate the performance, we use *LRE* (Least Ratio Error) following the convention of the CHF studies: $LRE(\hat{f}) = \sigma(\frac{y}{\hat{f}(X)})/E(\frac{y}{\hat{f}(X)})$, where y and $\hat{f}$ are the observed output and the estimated output, respectively.

The 10 fold cross-validation estimate of the average and the standard deviations of the observed CHF value over the predicted value are shown in Table 1. Our approach, feature-evolving genetic algorithm (FEGA), is compared with four other results. In the table, KAERI means the function which has been used at KAERI for years; PLS means a partial least squares that is a representative statistical method for soft modeling. WGA is a hybrid genetic algorithm combined with feedforward neural networks to optimize the weights. The results of WGA and FEGA are the average from 20 trials and they took almost the same running time. WGA and FEGA have the same network architecture: 12 input nodes, 6 hidden nodes, and one output node. WGA showed better performance than the existing approaches. FEGA outperformed WGA by about 4.3%. FEGA found a CHF function better than that of KAERI by about 14.3%.

Acknowledgment. This work was supported by grant No. (R01-2003-000-10879-0) from the Basic Research Program of the Korea Science and Engineering Foundation. This work was also supported by Brain Korea 21 Project. The ICT at Seoul National University provided research facilities for this study.

A Bio-inspired Genetic Algorithm with a Self-Organizing Genome: The RBF-Gene Model

Virginie Lefort, Carole Knibbe, Guillaume Beslon, and Joël Favrel

INSA-IF/PRISMa, 69621 Villeurbanne CEDEX, France
{vlefort, cknibbe, gbeslon}@prisma.insa-lyon.fr, joel.favrel@insa-lyon.fr

1 Introduction

Although Genetic Algorithms (GAs) are directly inspired by Darwinian principles, they use an over-simplistic model of chromosome and genotype to phenotype mapping. This simplification leads to a lack of performance, mainly because the chromosome structure directly constrains the evolution process.

In biology, the structure of the chromosome is free to evolve. The main feature permitting it is the presence of an intermediate layer (the proteins) between genotype and phenotype: Whatever the size and the locus of a gene, it is translated into a protein and all the proteins are combined to "produce" the phenotype.

Some authors, like Goldberg [1], have tried to introduce some independence between the genotype and the phenotype in GAs but none have really introduced the "protein level". Thus, they do not really part the two levels. We propose a new model of GA introducing such an intermediate level in order to permit evolvability *during* and *by* the evolutionary process to improve convergence.

2 The RBF-Gene Algorithm

Inspired by neural networks, we propose to discover the shape of the problem function by combining an unspecified number of basic functions, each of them having a fixed number of parameters. Each "gene" (i.e. each coding sequence) will encode one of these basic functions (called "kernels", e.g. Gaussian functions) exactly as, in biology, the genes encode proteins. The phenotype is then obtained by the interactions of the different kernels in a linear combination. So, the algorithm can be used to approach any bounded $\mathbb{R}^n$ to $\mathbb{R}$ function.

Coding and non-coding sequences are simply distinguished thanks to two genetic sequences ("start" and "stop" sequences). In between, the sequence will be analyzed thanks to a "genetic code" in order to compute the kernel parameters: Each base is associated to one parameter and to one value (0 or 1). Our kernels have three parameters: The mean vector (in $\mathbb{R}^n$), the standard deviation and the kernel weight in the linear combination. To compute the value of a kernel parameter, we extract the subsequence of all the values associated with it, which is then translated thanks to a variable-size Gray code.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 406–407, 2004.

Since there is no global rule to analyze the sequence, it can be modified by any biologically-inspired operator: Switches, local insertions and deletions, large-scale operators, crossover. . . The chromosome size, the number of kernels, the locus of the genes and the size of the coding sequences are then free to evolve.

3 Experiments: The Abalone Regression Task

We tested the algorithm on a regression benchmark: The abalone data set [2]. The goal is to find the age of an abalone given eight biological inputs. The data set contains 4177 experimental points and the fitness function is the root mean squared error. We have done 10 runs of 5000 generations each (each with 10 fold cross-validation). The results show that the RBF-Gene algorithm gives good results, similar or better than those given in [3].

In the very first generations, the algorithm adds a lot of new kernels. Then, it improves the existing kernels, modifying/enlarging the coding sequences to be more precise while the number of kernels remains steady.

The important aspect of the algorithm is the possibility to reorganize its genome while evolving functionally. We could think that the genome size will grow more and more. In practice, merely all chromosomes stabilize their size at around 3000 bases (the initial individuals have 200 bases) representing 37 kernels. We can see here "second order evolution" since long genomes are less *evolvable.*

4 Conclusions and Future Work

The RBF-Gene algorithm introduces a real independence between the genotype and the phenotype: All the genes encode proteins and the phenotype is always computable, whatever the number of genes. Moreover, the chromosome structure is dynamically modified during the evolution process to improve future evolution. Although it is still under study, our model gives promising results and proves its ability to add new genes during the evolutionary process.

Future work will focus on biologically inspired operators: By now, we used random operators while biological large-scale ones are based on sequence similarity to enhance their possibilities on reorganizing the structure of the genome.

References

1. Goldberg, D.E., Deb, K., Kargupta, H., Harik, G.: Rapid accurate optimization of difficult problems using fast messy genetic algorithms. In Forrest, S., ed.: Proceedings of the Fifth International Conference on Genetic Algorithms, San Mateo, CA, Morgan Kaufmann (1993) 56–64
2. UCI Machine Learning Website (http://www.ics.uci.edu/~mlearn/MLRepository.html): Abalone data set (consulted in 2003)
3. Automatic Knowledge Miner (AKM) Server: Data mining analysis (request abalone). Technical report, AKM (WEKA), University of Waikato, Hamilton, New Zealand (2003)

Evolving Spike-Train Processors

Juan Liu and Andrzej Buller

ATR Human Information Science Laboratories
2-2-2 Hikaridai, Keihanna Science City, Kyoto 619-0288 Japan
{juanliu; buller}@atr.jp

The research described in this paper was motivated by the idea to process purposefully given spike-trains using a cellular automaton (CA). CAs have three attractive features, namely massive parallelism, locality of cellular interactions, and simplicity of basic components (cells). However, the difficulty of designing a CA for a specific behavior causes limited interest in this computational paradigm. Automating the design process would substantially enhance the viability of CAs. Evolving CAs for purposeful computation is a scientific challenge undertaken to date by, among others, Mitchell *et al*. [1], Sipper *et al* [2] and de Garis *et al* [3].

In previous work [4], we designed a special 2-dimensional cellular automaton, called qCA, which can be used for purposeful manipulations on binary time-series (spike-trains). The proposed procedure of qCA synthesis is to decompose the problem into logic synthesis and routing. As for logic synthesis, we evolve a graph as a half-product for the qCA synthesis, called Pulsed Para-Neural Networks (PPNN). PPNN consists of functional nodes and directed edges representing pure delays, in which each node returns a pulse at clock t if it received one and only one pulse at clock t-1. The routing is a conversion of the obtained PPNN into an initial state of a qCA using a relatively trivial heuristics.

The desired PPNN is not assumed to be evolved from scratch. In this paper, we described an evolutionary method to find unknown delays of a given scheme so that the network could produce desired spike-trains. We investigated the scheme shown in Fig.1(a) as a test-bed of our method. The chromosomes of GA are represented as $V_i = (D_1,...,D_n) \in R^n$, where n is the number of delays to be found. The vectors (genotype) are converted into binary spike-trains (phenotype) via PPNN and then evaluated by a fitness function that compares the similarity of obtained and desired spike-trains. However, the fitness landscape is flat and punctuated with high spikes. And the small peaks around the global optimal point make the problem even harder. The idea to fuzzify and shift output signals resulted in a breakthrough. We converted the binary spike-trains produced by the evolved PPNNs into fuzzified time-series to differentiate spike-trains of the same Hamming distance. Then, we shifted the obtained spike-trains and took the best-fit value as the fitness of the chromosome so that the fitness value could be more consistent with the similarity of two spike-trains. The modified genotype-to-phenotype mapping procedure provided a smoother and more informative fitness landscape. Similar to memetic algorithms [5], our method is based on Lamarckian evolutionary process combining GA with a hill climbing technique to accelerate the convergence of the evolution. The experiment result is presented in Fig.1 (b), which confirms that the proposed approaches could improve the performance of the algorithm. It remains unclear whether additional fuzzification of signals circulating inside PPNN, or the fuzzification of node function, would result in higher evolvabil-

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 408–409, 2004.

ity. The same applies to other, smarter ways of calculating similarity of two time-series that is still under investigation.

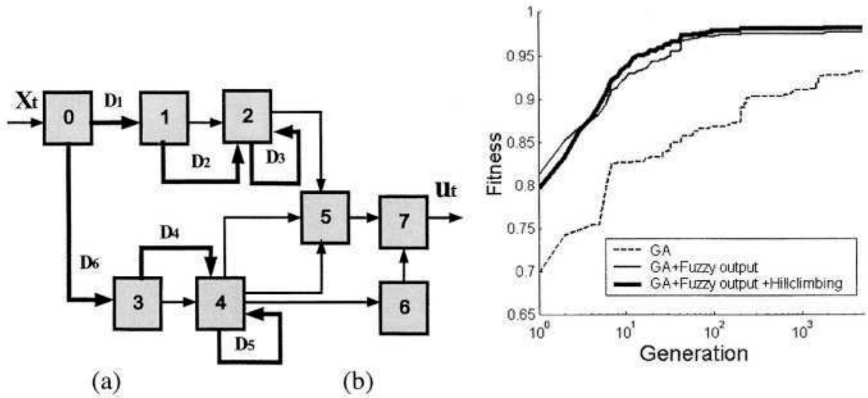


Fig. 1. Evolving PPNN. **(a)** Every non-labeled arrow means a connection of no delay. The presented scheme contains 6 unknown delays built on two parallel timers. For input sequence Xt=100000, the desired output spike-train is **U**t=00000000000000000000000000000000000 000000000000000000100000001000000010000000100000101010001000100000000000... where D_1, D_2, D_3, D_4, D_5, D_6 equal to 54, 40, 7, 16, 3, 84, respectively. Evolved PPNN can be easily converted into equivalent qCellular Automata, **(b)** The best fitness values of different methods averaged over 50 runs for evolving 4 unknown delays (D2, D3, D4, D5). As it can be noted, fuzzifying and shifting output signals as well as combining GA with a hill-climbing strategy dramatically accelerate the evolution of PPNN.

Acknowledgement. This research was conducted as a part of the Research on Human Communication supported by the Telecommunication Advancement Organization of Japan (TAO).

References

1. Mitchell, M., Crutchfield, J. P., and Hraber, P. T.: Evolving cellular automata to perform computations: Mechanisms and impediments. Physica D, 75, (1994) 361- 391
2. Sipper, M. and Ruppin, E.: Co-evolving cellular architectures by cellular programming. In Proceedings of IEEE Third International Conference on Evolutionary Computation (ICEC'96), (1996) 306-311
3. de Garis, H. et al: CAM-BRAIN—The evolutionary engineering of a billion neuron artificial brain by 2001 which grows/evolves at electronic speeds inside a cellular automata machine (CAM). In Proc. Int. Conf. Artif. Neural Nets Genetic Algorithms, Alés, D. W. Pearson, N. C. Steele, and R. F. Albrecht, Eds. France, Springer-Verlag, (1995) 84–87
4. Buller, A et al: ATR artificial brain project: key assumptions and current state. In: 9th International Symposium on Artificial Life and Robotics (AROB 9th '04), January 28-30, Beppu, Oita, Japan (2004)
5. Gen, M. and Cheng, R. W.: Genetic Algorithms and Engineering Design. John Wiley & Sons, INC., New York (1997) 33-34

A Philosophical Essay on Life and Its Connections with Genetic Algorithms

Fernando G. Lobo

ADEEC-FCT, Universidade do Algarve
Campus de Gambelas, 8000 Faro, Portugal.
flobo@ualg.pt

Abstract. This paper makes a number of connections between life and various facets of genetic and evolutionary algorithms research. Specifically, it addresses the topics of adaptation, multiobjective optimization, decision making, deception, and search operators. It argues that human life, from birth to death, is an adaptive or dynamic optimization problem where people are continuously searching for happiness.

This paper speculates that a person's life has many similarities with the way genetic algorithms (GAs) work, and argues that many problems encountered in day-to-day life have a direct counterpart in some aspect of GA research. In particular, the following topics are explored:

- Life has a large search space
- Happiness is life's best objective
- Happiness involves multiple objectives
- Life occurs in a non-stationary environment
- Life can have traps
- GA operators in real life

In the rest of the paper each of the above topics is briefly addressed (a longer version of this essay is available elsewhere [1]).

Human beings make decisions and take actions continuously. There are many different ways in which we can live our lives. The number is so large that one could say that the search space of life is unlimited. When we are born we are faced with a whole world to explore, just like an artificial individual at generation zero of a randomly initialized population.

Most people would agree that happiness is the main ingredient that drives our lives. We all want to maximize happiness, but happiness means different things to different people. In some sense, happiness is something that involves multiple objectives. People have different objectives or goals that wish to be fulfilled in life. The same thing holds for many real world optimization problems, and genetic algorithms have shown to be useful for that purpose.

What makes us happy today might not make us happy tomorrow. Our goal is to continuously seek happiness, day after day, year after year. In order to do that, global convergence should be avoided by all means. Life has no optimal

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 410–411, 2004.

solution. The important thing is to seek happiness and be prepared to react to changes in a timely manner. The same thing happens in many real world problems where the objective function changes through time.

Deception is one of the things that cause difficulties to GAs. An example of a deceptive problem is the so-called trap function. The problem is misleading because small incremental changes on a solution are rewarded, leading the search to a deceptive attractor. I firmly believe that deception occurs in life as well. It typically happens in those situations when one is not capable of looking far ahead enough, and simply follows the easy path. Herein I argue that the important thing is that one is able to recognize a trap, and once that happens, avoid by all means continuing doing small incremental changes. Instead, use GAs as a source of inspiration, and try escaping traps in more efficient ways. For example, by enlarging the population of our past decisions and finding the necessary linkages to allow escaping the trap in a quick way.

In genetic algorithms, selection is the operator that distinguishes good from bad solutions. Humans seem to have a natural way of applying selection in real life. A quotation from the Dalai Lama reveals the similarities.

> Generally speaking, one begin by identifying those factors which lead to happiness and those which lead to suffering. Having done this, one then sets about gradually eliminating those factors which lead to suffering and cultivating those which lead to happiness. (Dalai Lama)

If one observes the way most people live, one has to conclude that, with some exceptions once in a while, what people usually do in life is mostly small variations of what they have done in the past. In many ways, what we do today is not much different from what we did yesterday, and that sounds a lot like a mutation kind of exploration way of life. Likewise, one could speculate that crossover corresponds to the big decisions than one makes in life. Tracing back my own life, I can clearly recognize some landmarks that were very important for my development. In many ways, those were decisions that required a bit more thought, were more unusual, more challenging, more risky, but were also the ones that yielded the highest payoff on the long run.

As a final remark, it is interesting to observe that the pioneers of the field of genetic and evolutionary algorithms have used nature and life as an inspiration for designing computer algorithms. This paper speculates that the reverse is also true; that genetic algorithms can be used as a source of inspiration for life itself.

Acknowledgements. This work was sponsored by FCT/MCES under grants POSI/SRI/42065/2001 and POCTI/MGS/37970/2001.

References

1. Lobo, F.G.: A philosophical essay on life and its connections with genetic algorithms. arXiv Report cs.NE/0402050 (2004) (Available at http://arxiv.org/abs/cs.NE/0402050).

An Architecture for Massive Parallelization of the Compact Genetic Algorithm

Fernando G. Lobo, Cláudio F. Lima, and Hugo Mártires

ADEEC-FCT, Universidade do Algarve
Campus de Gambelas, 8000 Faro, Portugal.
{flobo,clima}@ualg.pt, hmartires@myrealbox.com

Abstract. This paper presents an architecture which is suitable for a massive parallelization of the compact genetic algorithm. The approach is scalable, has low synchronization costs, and is fault tolerant. The paper argues that the benefits that can be obtained with the proposed methodology is potentially higher than those obtained with traditional parallel genetic algorithms.

1 Introduction

With a traditional parallel genetic algorithm (GA) implementation, population members need to be sent over a computer network, and that imposes a limit on how fast they can be [1]. In this paper, we investigate the parallelization of the compact genetic algorithm (cGA) [2], and take advantage of its compact representation of the population do develop a parallelization scheme which significantly reduces the communication overhead.

The cGA uses a probability vector as a model to represent the population. The vector can be stored with $\ell \times \log_2(N+1)$ bits (ℓ is the chromosome length, N is the population size), a different order of magnitude than the $\ell \times N$ bits needed to represent a population in a regular GA. Since communication costs can be drastically reduced, it makes sense to clone the probability vector to several computers, and let each computer work independently on solving the problem by running a separate cGA. Then, the different probability vectors need to be consolidated (or mixed) once in a while.

2 Massive Parallelization of the Compact GA

We have developed an asynchronous parallelization scheme which consists of a manager processor, and an arbitrary number of worker processors. Initially, the manager starts with a probability vector with 0.5 in all positions, just like in a regular cGA. After that, it sends the vector to all workers who are willing to contribute with CPU time.

Each worker processor runs a cGA on its own based on a local copy of the probability vector. Workers do their job independently and only interrupt the

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 412–413, 2004.

manager once in a while, after a predefined number of m fitness function evaluations have elapsed.

During the interruption period, a worker sends the accumulated results of the last m function evaluations as a vector of probability fluxes with respect to the original probability vector. Subsequently, the manager adds the probability fluxes to its own probability vector, and resends the resulting vector back to the worker. Meanwhile, other worker processors can continue doing their job nonstop, even though some of them are working with a slightly outdated vector.

Each worker processor can start and finish at any given point in time making the whole system fault tolerant. When a worker starts, it receives a copy of the manager's probability vector, which already contains the accumulated results of the other cGA workers. On the other hand, when a worker quits, we simply loose a maximum of m function evaluations, which is not a big problem.

We have conducted computer simulations to validate the proposed approach and observed a linear speedup with a growing number of processors (see Figure 1). Additional details of the simulations and a longer description of this work can be found elsewhere [3].

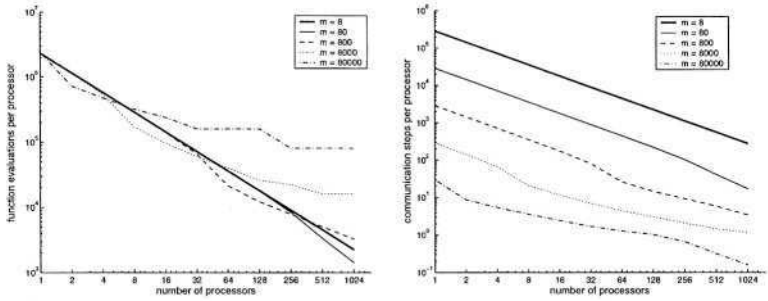


Fig. 1. On the left, we see the average number of function evaluations per processor. On the right, we see the average number of communication steps per processor.

Acknowledgements. This work was sponsored by FCT/MCES under grants POSI/SRI/42065/2001 and POCTI/MGS/37970/2001.

References

1. Cantú-Paz, E.: Efficient and accurate parallel genetic algorithms. Kluwer Academic Publishers, Boston, MA (2000)
2. Harik, G.R., Lobo, F.G., Goldberg, D.E.: The compact genetic algorithm. IEEE Transactions on Evolutionary Computation **3** (1999) 287–297
3. Lobo, F.G., Lima, C.F., Mártires, H.: An architecture for massive parallelization of the compact genetic algorithm. arXiv Report cs.NE/0402049 (2004) (Available at http://arxiv.org/abs/cs.NE/0402049).

An Evolutionary Technique for Multicriterial Optimization Based on Endocrine Paradigm

Corina Rotar

Computer Science Department
"1 Decembrie 1918" University, Nicolae Iorga 11-13
Alba Iulia, 2500, Romania
crotar@uab.ro

1 MENDA Technique

We propose a new evolutionary algorithm (*M*ultiobjective *End*ocrine *A*lgorithm) for multiobjective optimization, which applies several principles inspired of endocrine system. Even if hormonal system has a particular dependence of nervous system, due to the intrinsic mechanisms of control and its functions, it represents by itself a suitable paradigm in evolutionary computation's landscape.

The principle of the proposed method relies on keeping two populations: an active population of hormones, H_t, and a passive population of non-dominated solutions, A_t. The members of the passive population behave as a population of elite and also have a supplementary function: to lead the hormones toward the Pareto front, keeping them as much as possible well distributed among the search space. These two populations correspond to the two classes of hormones in endocrine paradigm: specific hormones, which are released by different glands of the body and the hormones of control (tropes), which are produced by the control level of the endocrine system in order to supervise the density of each type of specific hormones.

Population of control, A_t, is modified at each generation, gathering all non-dominated solutions from the population U_t, which has resulted from merging current population of hormones H_t and the previous population A_t. The passive population, A_t, doesn't suffer modification at the individual's level. It behaves as an elite population of non-dominated solutions from the current generation, which is only rebuild at each generation. Finally, population A_t contains a predetermined number of non-dominated vectors and provides a good approximation of Pareto front.

At each generation t, the members of H_t are classified. A corresponding controller from A_t supervises a particular class. The idea is that each hormone from H_t is supervised by the nearest controller from A_t. A member of the population A_t has a similar control function as a trop from endocrine paradigm.

We use two fitness functions. The first fitness function measures the crowding degree of the class of the individual. The second fitness of each individual calculates the number of dominated solutions from the current generation.

Another specific issue of the proposed algorithm is the manner of selecting and recombining the hormones in order to generate descendants. First parent is selected from entire population H_t, proportionally with the crowding degree of the classes. By this manner of selecting first parent, the less crowded hormones are preferred and those zones from the search space, which are apparently disappearing, would be re-

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 414–415, 2004.

vived. The second parent is selected only from the class of the first parent, accordingly to the value of the second fitness function. By selecting the second parent accordingly to the values of the second fitness function, the method assures a faster convergence of the population toward Pareto front. Crossover operator recombines two parents and produces a single descendant, which is accepted into the next population.

2 Results and Conclusions

Nature offers complex phenomena, processes and systems, which can inspire us in many ways. An example of this kind is the natural endocrine system. Our work essentially attempts to mimic hormones' behavior in order to solve difficult problems like multiobjective optimization. Consequently, we propose a new technique based on endocrine paradigm, called *MENDA* (*M*ultiobjective *End*ocrine *A*lgorithm). It provides satisfactory solutions in our tests.

We used in our tests many bi-objective test functions found in the literature. The results were satisfactory. Among these, the popular test suit of 6 bi-objective functions proposed by Deb [1], [3] was a real challenge. We had to modify the crossover operator in order to overcome the initial fails. The results were evaluated using *Spacing* metric and *Error Ratio* metric [2]. The values of those metrics were encouraging.

Table 1. The results obtained for the first four test functions proposed by Deb[3]. The algorithm run for 20 times and the average values for the considered metrics are shown.

Test Function	Number of Generations	Non-dominated Solutions	Spacing Metric	Error Ratio Metric
F1	60	83	0.0056	0
F2	65	83	0.0058	0
F3	121	97	0.0061	0
F4	37	82	0.0059	0

References

1. Deb, K., Multi-objective genetic algorithms: Problem difficulties and construction of test functions, Evolutionary Computation, 7(3), 1999.
2. Van Veldhuizen, D., Lamont, G., Multiobjective Evolutionary Algorithm Research: A History and Analysis, Technical Report: TR-98-03, 1998.
3. Zitzler, E., Deb, K., Thiele, L, Comparison of Multiobjective Evolutionary Algorithms: Empirical Results, Evolutionary Computation, vol. 8, no. 2, 2000.

Evolving Golomb Rulers

Jorge Tavares[1], Francisco B. Pereira[1,2], and Ernesto Costa[1]

[1] Centre for Informatics and Systems of the University of Coimbra,
Polo II - Pinhal de Marrocos, 3030 Coimbra, Portugal
[2] Instituto Superior de Engenharia de Coimbra,
Quinta da Nora, 3030 Coimbra, Portugal
{jast, xico, ernesto}@dei.uc.pt

1 Introduction

A Golomb Ruler is defined as a ruler that has marks unevenly spaced at integer locations in such a way that the distance between any two marks is unique. Unlike usual rulers, they have the ability to measure more discrete measures than the number of marks they possess. Although the definition of a Golomb Ruler does not place any restriction on the length of the ruler, researchers are usually interested in rulers with minimum length. An Optimal Golomb Ruler (OGR) is defined as the shortest length ruler for a given number of marks [1].

2 Evolution with Marks

We propose a straightforward chromosome representation that allow us to evolve both the number and position of marks for a given ruler with a predetermined maximum length *L*. An individual is codified as a binary string with length *L*, where each bit is associated with an integer position in the ruler. If a given gene has the value 1 then it indicates that there is a mark in the corresponding position. To generate descendants we selected variants of standard genetic operators usually applied to binary representations: two-point crossover and shift mutation. Shift mutation acts in the following way: it randomly selects a mark and shifts it left or right with equal probability. To determine how many marks are shifted in a given individual, a value is uniformly selected from the integer interval $[1, S]$, where S is a small constant. To evaluate an individual we consider two criteria: ruler length and legality of the solution. The equation used to assign fitness to an individual x is the following:

$$fitness(x) = \begin{cases} \text{- number of repeated measurements} & \text{, if } x \text{ is illegal} \\ \text{number of marks} \times \text{L} + (\text{L - M}) & \text{, if } x \text{ is legal} \end{cases} \quad (1)$$

Where L is the maximum length of the ruler and M is the length of the ruler encoded in the individual.A correction and insertion procedure is performed during the evaluation of an individual. The aim is twofold: to fix invalid rulers and to see if it is possible to add marks to legal solutions. The line of action is straightforward: a mark is randomly removed if, when checking the segments

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 416–417, 2004.

measured by the ruler, a duplicate measurement is found. A constant C gives the maximum number of rectifications allowed for a given solution. If a valid ruler is obtained at the end of the correction, then the second stage can proceed: an insertion operator tries to add marks in every possible position ensuring that the obtained ruler is still valid.

3 Experimental Results

To evaluate our approach (Marks-EC), we used it to seek for good rulers with lengths between 90 and 200 (given the known OGRs we will be looking for rulers between 12 and 16 marks). In table 1 we summarize the results achieved by Marks-EC and compare them with previous evolutionary approaches: Solidays EC [2] and RK, RK heuristic [3]. We also present the optimal length for each one of these rulers.The overall results attained by Marks-EC clearly outperform previous approaches. It was able to find the OCR for all tested instances with the single exception for 15 marks. Nevertheless, for OGR-15 the solution discovered is just 2:65% higher than the optimal value, whilst the previous best value found by an evolutionary approach is 7:28% higher. A detailed analysis of the results can be found at [4].

Table 1. Overview of the results.

Instances	Optimal	Soliday's EC	RK	RK Heuristic	Marks-EC
OGR-12	85	103	91	85	85
OGR-13	106	124	111	106	106
OGR-14	127	168	134	131	127
OGR-15	151	206	160	162	155
OGR-16	177	238	193	180	177

References

1. Golomb, S.: How to Number a Graph. In: Graph Theory and Computing. Academic Press (1972) 23–37
2. Soliday, S., Homaifar, A., G., L.: Genetic algorithm approach to the search for golomb rulers. In: Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA-95), Morgan Kaufmann (1995) 528–535
3. Pereira, F.B., Tavares, J., Costa, E.: Golomb rulers: The advantage of evolution. In: Proceedings of the 11th Portuguese Conference on Artificial Intelligence, Workshop on Artificial Life and Evolutionary Algorithms (ALEA), EPIA'03. (2003) 27–33
4. Tavares, J., Pereira, F.B., Costa, E.: Understanding the role of insertion and correction in the evolution of golomb rulers. In: Proceedings of the Congress on Evolutionary Computation (CEC 2004). (2004)

Populating Genomes in a Dynamic Grid

Han Yu, Ning Jiang, and Annie S. Wu

School of Computer Science, University of Central Florida, Orlando, FL 32816-2362
{hyu, njiang, aswu}@cs.ucf.edu

1 A Dynamic Grid Environment

We study the behavior of a spatially distributed GA (Collins and Jefferson 1991) that has several unique features. First, individuals are allowed to move within a two dimensional grid over time. Such movement results in a continual change in the local environment in which each individual interacts. Second, selection is directly based on an individual's energy value and indirectly based on fitness. The fitness of an individual determines its initial energy level, but an individual's energy level at any point during a run reflects a combination of its fitness and its interactions in the environment. Third, the population size can change dynamically throughout a run.

We study the mating radius of the individuals, and the energy-based selection which results in noisy selection. We investigate the impact of these two aspects on the selection pressure among individuals.

2 Experiments and Analysis of Results

We use the Hierarchical-if-and-only-if(H-IFF) function as the test bed to study the behavior of this grid environment. We test different mating radii ranging between one and nine grids. Results indicate that the mating radius has significant impact on the performance of this GA. A GA with a mating radius of one has the highest probability of finding solutions. GAs with larger mating radii achieve competitive results in the 16-bit function, but performance degrades quickly as problem size increases. Experiments also show that, in cases where the GA finds an optimal solution, a large mating radius improves the search efficiency. The larger the mating radius, the fewer fitness evaluations necessary to find a solution.

Further analysis, with the results shown in Figure 1, reveals that although a large mating radius gives individuals the chance to find more candidate mates, the probability of a successful mating is very low due to high competition among the individuals within a neighborhood. As a result, fewer offspring are created in each time unit than when a small mating radius is used. A large mating radius also encourages the GA to converge quickly to a single solution, which is not desirable for domains with large or rugged search spaces. Slower convergence allows a GA to have more complete coverage of search space. This explanation is consistent to the experimental results on the H-IFF function, where a GA

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 418–419, 2004.

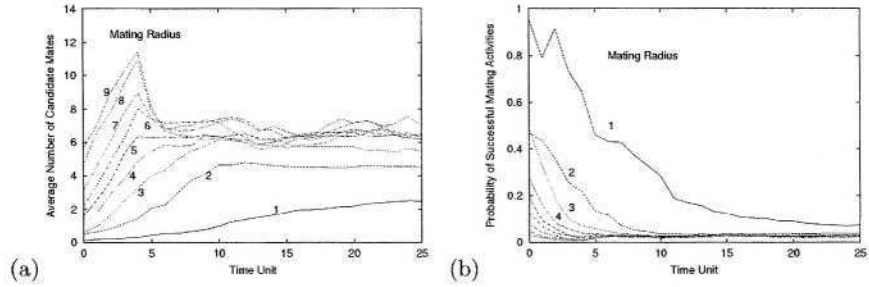


Fig. 1. (a) Average number of candidate mates in the early period of typical runs using different mating radii (b) Probability of successful mating activities in the early period of the same runs.

with a mating radius of nine is the fastest to solve the smaller 16-bit function but performs much worse on larger functions.

A distinguishing feature of our GA is that selection among individuals is based on their energy values. The energy of an individual is initialized as its fitness and fluctuates throughout the individual's lifetime. As a result, the energy-based selection introduces "noise" in the selection scheme. Figure 2 shows that noise in our selection method helps to reduce the selection pressure as compared to traditional fitness-based selection. As the energy does not truly reflect the fitness of an individual, the noise provides more chance for low fitness individuals to succeed in mating. This phenomenon is more noticeable in a highly converged population as both good and bad individuals have nearly equal chance of mating.

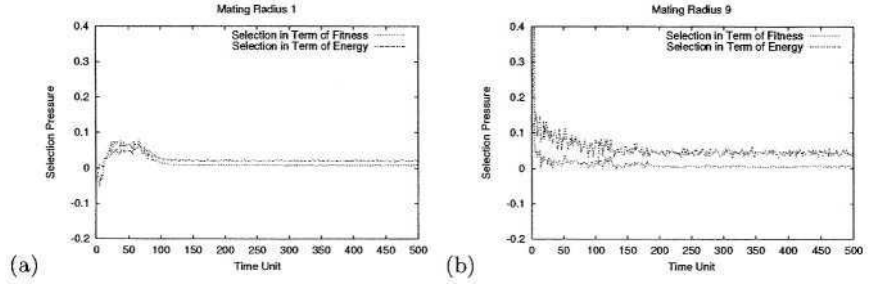


Fig. 2. The selection pressure for mating among individuals with (a) mating radius of 1 (b) mating radius of 9.

References

1. Collins, R. J., Jefferson, D. R.: Selection in massively parallel genetic algorithms. In: Proc. of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann (1991) 249-256

Empirical Study of Population Diversity in Permutation-Based Genetic Algorithm

Kenny Q. Zhu and Ziwei Liu

Department of Computer Science, National University of Singapore, Singapore 119260
kzhu@comp.nus.edu.sg, lziwei@hotmail.com

Abstract. This paper presents an empirical study of population diversity measure and adaptive control of diversity in the context of a permutation-based algorithm for Traveling Salesman Problems and Vehicle Routing Problems.

Maintenance of diversity adaptive control is one of the most fundamental issues of Genetic Algorithm (GA). We are concerned in this paper with GA in which individual chromosomes are integer encoded, and the crossover operations are *permutation-based* [3]. Here, all individuals have the same set of distinct alleles (integers) in all generations. The different permutations of the individuals decode into different fitness values. Permutation-based GA is used to solve Traveling Salesman Problem (TSP) [2], Vehicle Routing Problem (VRP) [1], and many other problems. Permutation-based crossover operator *Partially Matched Crossover(PMX)* [3] is used in this research, with an application rate of p_c. We use a *sequence insertion* mutation, which is defined as relocating a subsequence from one position of the chromosome to another position. Probability of mutation is p_m.

We define the *phenotypes (ptype)* diversity measure as the number of unique fitness values in the population, divided by the size of the population.

With a fixed, 100% random initial population, and PMX crossover only, the basic algorithm was run 10 times with $p_c = 0.7$ and $p_m = 0$. In each generation, ptype is recorded and plotted in Fig. 1, which demonstrates the natural convergence of the ptype diversity without any mutation. We further ran the algorithm to 201 generations for 50 times, and plot the rankings of accumulated diversity over 201 generations against the rankings of the mean fitness at the 201st generation, and this is shown in Fig. 2. One can observe that the more diverse the populations are in a run, the better the search quality. In other words, there is some positive correlation between diversity and search quality.

Fig. 3 and Fig. 4 illustrate the effect of genetic operations on the ptype diversity. One can observe from these plots that both crossover and mutation promote diversity, with increasing application rate. The correlation between diversity and search quality motivates us to maintain diversity at some desirable level so that more promising regimes in the search space can be explored. And the effect of crossover and mutation suggests that one can control the diversity by adaptively calibrating the crossover/mutation rates against the changing population

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 420–421, 2004.

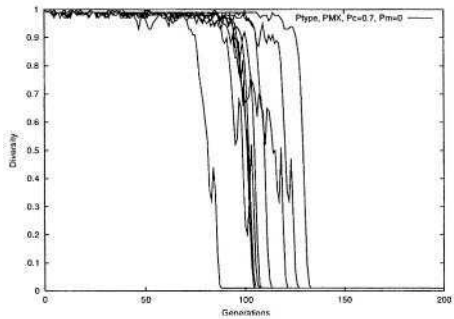


Fig. 1. Ptype vs generations

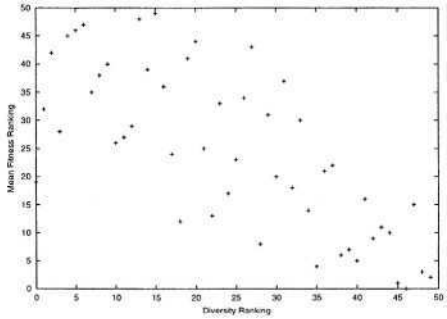


Fig. 2. Ptype rankings vs mean fitness rankings

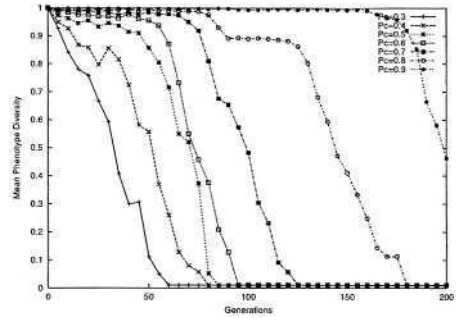


Fig. 3. Effect of PMX on Ptype

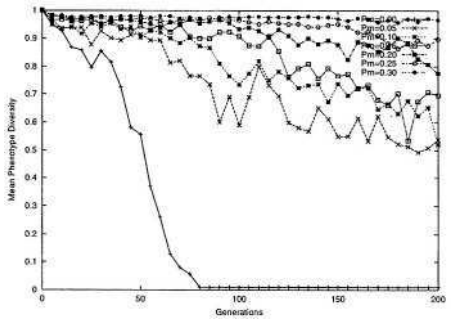


Fig. 4. Effect of mutation on Ptype

diversity. We apply the following simple adaptive function.

$$p' = max(p_{min},\ min(p_{max},\ p(1 + \frac{\xi(d_t - d)}{d}))), \tag{1}$$

where p is the current rate p_c or p_m, p' is the new rate in the next generation, d is the diversity of current population, d_t is the target diversity, ξ is the control *sensitivity,* and p_{min}, p_{max} are constants. Our preliminary experiments on various VRP benchmark problems indicated that adaptive control outperforms the search quality of fixed parameter GA by 1-5%.

References

1. L. Bodin, A. Assad, and M. Ball. *Routing and Scheduling of Vehicles and Crews - the State of the Art.* Pergamon Press, 1983.
2. G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Operations Research,* 2, 393–410.
3. D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Pub. Co., 1989.

A Demonstration of Neural Programming Applied to Non-Markovian Problems

Gabriel Catalin Balan and Sean Luke

George Mason University, Fairfax, VA 22030
gbalan@cs.gmu.edu, sean@cs.gmu.edu

Abstract. Genetic programming may be seen as a recent incarnation of a long-held goal in evolutionary computation: to develop actual computational devices through evolutionary search. Genetic programming is particularly attractive because of the generality of its application, but it has rarely been used in environments requiring iteration, recursion, or internal state. In this paper we investigate a version of genetic programming developed originally by Astro Teller called *neural programming*. Neural programming has a cyclic graph representation which lends itself naturally to implicit internal state and recurrence, but previously has been used primarily for problems which do not need these features. In this paper we show a successful application of neural programming to various partially observable Markov decision processes, originally developed for the learning classifier system community, and which require the use of internal state and iteration.

1 Introduction

Early evolutionary computation work (famously [1]) centered not on optimizing the settings for multidimensional parameter spaces but on using simulated evolution to develop *mechanisms* which perform computational functions. This goal has resurfaced in different forms over time, including the evolution of classifier systems, neural networks and automata, and most recently, genetic programming. The impetus for GP has from the beginning been "computer programs" which ideally can "perform iteration and recursion", "define intermediate values and subprograms", etc. ([2], p. 2). In keeping with this motivation, many GP methodologies have encorporated forms of iteration, recursion, and internal state. This notwithstanding, nearly all of genetic programming concerns itself with relatively simple feed-forward functions, and given the lack of even rudimentary iteration or internal state in common GP problems it seems strange to refer to GP as genetic *programming* at all.

As discussed later, much of the work in computational mechanisms involving iteration and internal state, and many illustrative problem domains, have recently been in learning classifier systems. Extensions to classifier systems such as XCS have of late added basic facilities to learn agent policies for partially-observable Markov decision processes, whose non-Markovian features require internal state. But classifier systems are of limited generality. Our interest is in

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 422–433, 2004.

evolving programs as graph structures with at least basic iteration and internal state, in forms sufficiently general to be used for a wide variety of tasks.

Most work in recurrent GP has so far centered around methods where a single node in a graph is currently in control at any one time, and control passes forward to a single successor node. An exception to this pattern is Astro Teller's *neural programming* (NP) architecture [3] which activates all nodes in the graph simultaneously, letting data flow synchronously in parallel to successor nodes. Teller's model has appealing features which we believe could provide interesting advantages over a single state model: data parallelism, more natural use of functions rather than state transitions, and the promise of more graceful degradation under modification. However Teller did not use his model in non-Markovian environments requiring internal state and iteration, choosing instead to demonstrate its functionality with problems to which a traditional GP representation may be easily applied. In this paper we will demonstrate, as a proof of concept *only,* that the NP model may be used in non-Markovian environments.

The test problems we chose in this paper are "Woods" problems common to learning classifier systems. We chose these problems for our test cases specifically because they are *not* problems for which neural programming is likely to be well-designed, particularly because our neural programming representations can be of any size. These problems are custom-tailored for discrete, policy-oriented single-state systems such as XCS: but we will demonstrate neural programming's efficacy on these problems. In future work we hope to test neural programming on continuous environments for which we think it is likely to be a much better match, such as two-pole balancing with no provided velocity information.

The rest of the paper is organized as follows. First we discuss relevant previous work in GP and learning classifier systems, including the Woods problems. We then describe our version of neural programming and how we apply it to these problems. Last, we present the experimental work and results.

1.1 Genetic Programming and Neural Programming

Though some early work attempted to add mapping, iteration, or recursive operators to genetic programming [2], these aims have never caught on in the community, with a few important exceptions ([4,5]). Iteration operators and internal memory are generally rare in GP. Recursion is easily achievable with *automatically defined functions* (ADFs), but ADFs are primarily used to add modularity rather than recursive function calls.

Angeline [6] proposed a variation on the ADF theme, called *multiple interacting programs* (MIPs) nets, which replaced recursion with forward data flow: trees generated values which were then fed into "parent" trees: cycles in parenthood were permitted. Like neural programming, MIPs nets allowed for parallelism.

The alternative method, using a single active state at a time, is presented in [7]. Here, nodes in a cyclic call graph contain both actions (side effects) and conditional branches. The graph structure was later replaced with a sequence of linear genetic programming instructions [8]. Similarly [9] allowed GP programs with single active states to be augmented with forward and backward edges,

Table 1. Genetic operators, with probabilities of occurrence and descriptions.

Mutate-ERC (20%) changes all ephemeral random constants.
Mutate-Nodes (20%) replaces a random number (from 1 to 3) of nodes with ones of identical arity, so that the graph's topology is not affected.
Mutate-Edges (25%) changes the source nodes of a random number (from 1 to 3) of edges.
Mutate-Result-Node (20%) randomly designates a new result node.
Crossover (10%) swaps fragments between two graphs. First, each graph randomly chooses a number N, where N is the number of nodes to be kept and the remaining M are to be crossed over. N may differ for the two graphs. The values of N are chosen with the constraints that both graphs may not crossover all, nor zero, of their nodes, and that no crossover may result in a graph of less than 2 or more than 40 nodes. A BFS traversal is then performed on each graph, starting with the result node, until N nodes have been traversed. The subgraph consisting of the M nodes not in the traversal is swapped with its counterpart in the other individual. Nodes with broken inputs have new input edges attached from randomly-chosen sources.
Randomize (5%) replaces the individual with a brand-new individual. First, between 2 and 40 nodes are generated and placed in an array: terminals (zero-arity nodes) are selected 1/3 of the time, and nonterminals 2/3 of the time. Nodes needing inputs are connected to neighboring sources chosen at random from up to 20% of the array length in either direction.

forming cycles. [10] incorporated time delays to prevent infinite execution loops in the graph, and also used separate action and branch nodes. This work also showed the effectiveness of the method using non-Markovian multiagent problems.

In Teller's neural programming (NP) [3], the representation is a graph of nodes similar to tree-based GP nodes. Each node takes input from other nodes, applies a function to its inputs, and outputs the result. Cycles are permitted and are common. In NP, all the nodes execute their functions simultaneously and synchronously based on the data each received in the previous time period. Since there is no single state, there is also no "start node", and so nodes must all have initial default values they output until they begin receiving data from other nodes. Just as in tree-based GP, NP nodes may also have zero inputs, and ephemeral random constants are permitted.

Our implementation of NP differs from the original in a few important ways. First, NP had an ADF mechanism, but in our experiments we will not use it. Second, NP's nodes were normally not restricted in arity: as many or as few nodes as necessary could feed them data. In our implementation this is not the case — we will enforce a fixed arity in a fashion similar to tree-based GP nodes. Third, NP originally featured a credit assignment mechanism used to determine which edges and nodes should be more susceptible to genetic modification. This mechanism was relatively easy to use for Teller's original signal-classification applications, but in the Woods problem it is somewhat problematic and we have discarded it. Last, neural programming graphs were augmented with one or more fixed "output nodes" whose values indicated the output of the program. We do not use a fixed output node, but instead allow the system to designate any node in the graph to also be the "result node" whose output is the output of the program. Each time step, the current value of the result node may change as data continues to flow through the graph.

Table 2. The characteristics of the Woods mazes. Shown are the number of ambiguous states and the number of equivalence classes they fall into; and the maximum and average path lengths of perfect solutions.

	Ambiguous		Perfect solution	
Environment	classes	states	max length	average length
Woods101	1	2	4	2.9
Maze7	1	2	7	$4.\overline{3}$
MazeF4	1	2	7	4.5
E1	9	20	4	$2.\overline{81}$
E2	5	36	5	$2.979196\overline{6}$
Maze10	3	7	8	$5.0\overline{5}$

Graph-structured representations inevitably require exotic genetic operators. Table 1 contains the description of the operators we used. We iteratively picked one operator by its probability, then applied it once to selected individuals, resulting in one or (for Crossover) two children. Individuals were selected using tournament selection of size 5. Population initialization used the Randomize operator described in Table 1.

2 The Woods Environments

In a simple version of the Woods problem [11,12] an agent navigates a grid-based maze trying to find food. The grid cells are either obstacles ("woods"), food (the goal), or are empty. An agent can only sense the contents of cells in its immediate 8-neighborhood, and may only transition to one of the eight neighbors in that timestep. If an agent attempts to transition to a woods cell, it fails to do so and remains at its current location. The goal is to guide the agent to the food in the minimal number of steps *regardless of initial position.*

Many woods problems are Markovian: a set of behaviors may be learned for the agent which guide it to the food each time based solely on the current sensor values. In other problems, different cells yield identical sensor values but require different actions, and so the agent must rely to some degree on internal memory. This situation is referred to as the *perception aliasing problem,* and the associated environments are said to be non-Markovian.

Non-Markovian environments have long been studied by the learning classifier system literature. Early work [14] allowed rules to write to a global internal state. Wilson [15] more formally suggested adding internal memory condition and action segments to rules, allowing the storage of information in bit-registers. Bit registers have since been used to extend other models ([16,17]). Relatively few such papers allow more than a fixed, and usually very small, number of registers. An alternative approach is to allow the action part of a rule be a sequence of actions rather than a single action [18].

Work in these problems has also been done using Pitt-approach rule systems [13], extending the mechanism to finite-state automata. Some work has also been

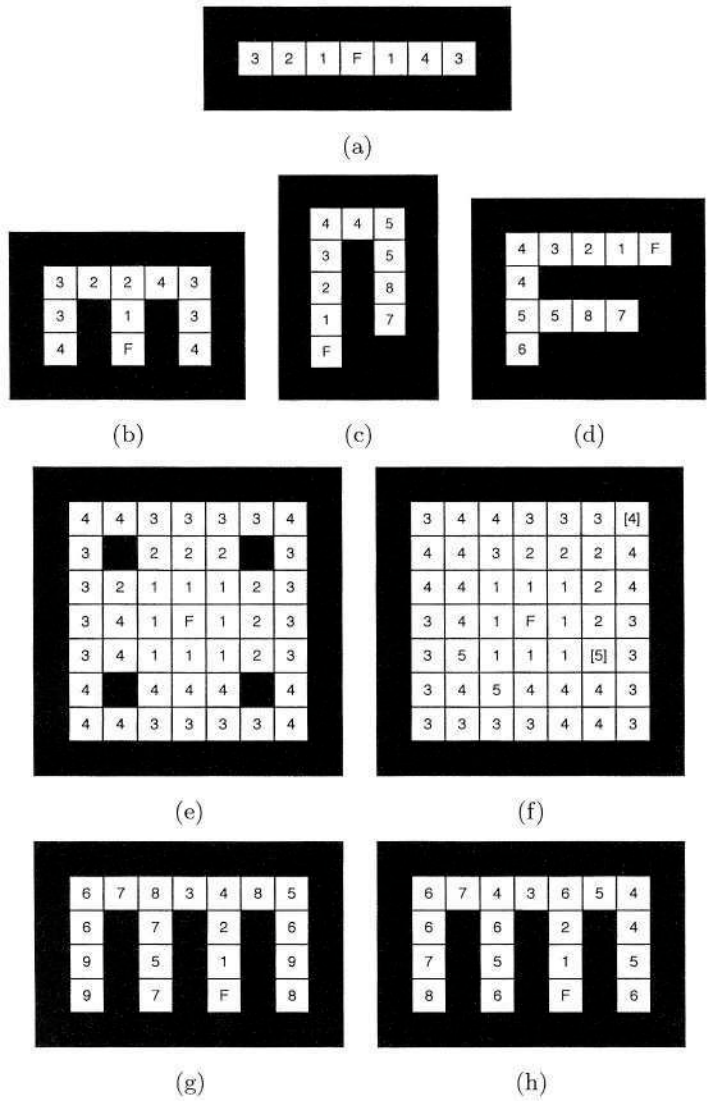


Fig. 1. The non-Markovian Woods environments used in this paper. The notation is consistent with [13]: black cells are obstacles and F is the goal state. Each figure also shows one (possibly suboptimal) solution to the problem: a cell is numbered with the steps necessary to reach the goal from that cell using this solution. (a) Woods100 shows the solution represented by the neural program in Figure 2. (b) Woods101, (c) Maze7, (d) MazeF4, (e) E1, (f) E2, and (g) Maze10 show solutions discovered by the NP system in the paper; cells marked [ ] are one step longer than an optimal solution. (g) shows an optimal solution for Maze10 suggested by Samuel Landau.

Table 3. The Function Set for the Woods Problem. Each function is followed by the inputs to the function, and a function description. All functions operate on 2D real-valued vectors. If a node has not yet received input, its output is the vector {0,0}.

Delay (v_1) Outputs its input.
To8 (v_1) Rounds its input to point to the nearest of the eight canonical directions.
Wall Filter / Empty Filter (v_1) Returns its input if the input points to a wall / empty cell. Otherwise, returns the vector {0,0}.
Wall Radar / Empty Radar () Cycles through vectors pointing towards the neighboring walls / empty cells in trigonometric order, outputting each in turn. After every complete cycle, outputs one {0,0} vector.
Wall Sum / Empty Sum () Returns the sum of those vectors from the eight directions that point to walls / empty spaces.
Add (v_1, v_2) Adds two vectors together.
Sub (v_1, v_2) Subtracts two vectors.
Accumulator (v_1) Accumulates the vectors inputted over time. Equivalent to an Add node with one input connected to its own output.
Time Weighted Accumulator (v_1) Accumulates the vectors inputted over time, each multiplied by the time step. The output at time t is $\sum_{i=1}^{t-1} i \times input_i / \sum_{i=1}^{t-1} i$.
Vector ERC () An ephemeral random 2D constant.
Rotation ERC (v_1) Rotates its input vector clockwise by an angle equal to $k\pi/8$, where k is an integer ephemeral random constant in the range $1 \ldots 15$.
If-Equal (v_1, v_2, v_3, v_4) Returns v_3 if $v_1 = v_2$, else v_4
If-Greater (v_1, v_2, v_3, v_4) Returns v_3 if $|v_1| > |v_2|$, else v_4
If-Dot (v_1, v_2, v_3, v_4) Returns v_3 if $v_1 \cdot v_2 > 0$, i.e. $\sphericalangle(v_1, v_2) \in [-\pi/2, \pi/2]$, else v_4

done in performing pareto optimization to minimize the amount of internal state necessary [19].

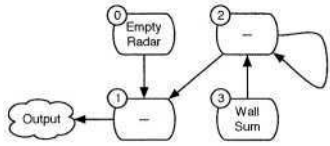
The Woods problems studied here are shown in Figure 1. Characteristics of these environments (number of ambiguous states, number of classes those states fall into, and maximum and average path lengths of perfect solutions) are shown in Table 2.

2.1 Application of Neural Programming to the Woods Problem

To move the agent around a Woods maze, we chose to have the neural program output a vector rather than move the agent via function side effects. All functions in the function set (shown in Table 3) operated on 2D continuous vectors and returned vectors.

Each time we needed to move the agent one square, we first determined if the agent was right next to food. If so, we hard-coded its movement directly to the food. If not, we evaluated the neural program graph to determine where the agent should go. First, the graph was stepped m times to "prime" it. Then the graph continued to be stepped until the result node outputted a vector in a valid direction (that is, one not pointing into woods and not {0,0}), and moved the agent to the eight-neighbored square in the direction of that vector. If after $M \geq m$ steps the result node still did not yield a valid direction, the agent was not moved. The particular sizes of M and m chosen are problem-specific, and are indicated as (m, M) in the discussion for each experiment.

Fitness assessment may be done in a variety of ways. We chose to run the neural program with the agent placed in each empty square, then averaged the



0	Empty Radar
1*	Sub (2, 0)
2	Sub (2, 3)
3	Wall Sum

Fig. 2. Two views of a neural program solving the Woods100 maze. The table at right lists the four program nodes (0–3), and indicates the nodes provided as input. The result node is marked with a $*$. The figure shows the equivalent graph structure.

Table 4. Value traces for all nodes of the graph in Figure 2 operating in the Woods100 maze starting at position $\langle 1, 1\rangle$. The agent is primed six times prior to making an action. The action is read from the result node shown in light gray. Cells contain the output vectors for each node in polar coordinates: size and heading.

		$\langle 1,1\rangle$						$\langle 2,1\rangle$						$\langle 3,1\rangle$						position
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	time step
0 Empty Radar	0	1 →	0	1 →	0	1 →	0	1 →	1 ←	0	1 →	1 ←	0	1 ←	0	1 ←	0	1 ←	0	
1 Sub(2,0)	0	1 ←	1 →	1 →	1 →	3 →	3 →	5 →	5 →	7 →	6 →	5 →	7 →	6 →	7 →	6 →	7 →	6 →	7 →	
2 Sub(2,3)	0	0	1 →	2 →	3 →	4 →	5 →	6 →	6 →	6 →	6 →	6 →	6 →	6 →	6 →	6 →	6 →	6 →	6 →	
3 Wall Sum	0	1 ←	1 ←	1 ←	1 ←	1 ←	1 ←	0	0	0	0	0	0	0	0	0	0	0	0	

lengths of the paths it took to find the food. An agent might never reach the food, so path length is limited to a problem dependent cutoff threshold *max-walk*.

An Example Consider the neural program shown in Figure 2. The table to the right shows the four nodes of the program, each labelled with an index 0–4. Node 1 is labelled with a $*$, indicating that it is the result node. The figure to the left shows the recurrent call-graph of the neural program.

This neural program will be run with an (m, M) value of (6,12). Table 4 shows its trace moving from position $\langle 1, 1\rangle$ to $\langle 3, 1\rangle$ in the Woods100 maze. In this example, before each action the neural program is primed six times, after which it happens to be outputting a valid vector and needs no further stepping. Figure 1(a) shows the path lengths of the solution followed by this program were it to be started in any particular empty location.

3 Experimental Results

We have tested the described system on six non-Markovian environments: Woods101, Maze7, MazeF4, E1, E2 and Maze10 (the Woods100 maze is trivial). These environments vary widely in difficulty.

Table 5. NP vs. Random search.

Problem	Priming (m, M)	Evals	Random Search			Evolutionary Search		
			Mean	Stdev	#Ideals	Mean	Stdev	#Ideals
Woods101	(6,6)	500K	3.47999	0.18735	0	3.42399	0.35945	2
Woods101	(6,12)	100K	4.08200	0.44524	0	3.40599	0.36500	2
Maze7	(17,34)	100K	7.67333	0.83944	0	5.97777	1.04424	3
MazeF4	(11,22)	100K	8.21	1.17649	0	6.97600	1.45301	0
E1	(11,22)	100K	3.22914	0.47751	0	2.92454	0.11119	12
E2	(11,22)	500K	3.17666	0.03885	0	3.08416	0.03668	0
Maze10	(12,24)	100K	19.8344	2.42619	0	13.0622	5.05325	0

We implemented NP using ECJ ([20]). Unless specified otherwise, all experiments evolved populations of 1000 individuals for 100 generations each. Each individual was one neural program containing between 2 and 40 nodes. For all environments but the last one, *max-walk* was set to 10.

This is the first time a generalized graph-based GP method has been tried on these problems. Even if it were not, few examples from the GP literature discussed earlier provide useful statistics to compare against. The method described here, as we imagined, is generally far outperformed by the learning classifier literature, though it fares well against the Pitt-approach rule system results. This is to be expected though: these problems are custom-tailored for LCS systems. Our method is hobbled by searching a wide range of internal states and a more general-purpose architecture. This reduces us to demonstrating proof of concept. We do so by comparing the search method against random search for the same representation.

Thus we compare the best-of-run fitness values of 50 runs with the best values found in 50 runs of random search with an equal budget of evaluations, using an ANOVA at 95% confidence. Note that by random search we mean generating random individuals every generation, and not just performing uniform selection. Figure 5 shows the results. In every case we outperform random search, except for the easy 500K Woods101 problem, for which the difference is statistically insignificant. We also find ideal solutions in several cases, whereas random search finds none.

3.1 Woods101

The Woods101 environment, studied in [16,21], is a very simple maze with only one pair of perception aliased states — see Figure 1(b). This means that learning classifiers and finite-state automata only require two internal states to solve the problem. We found optimal programs in two runs set at (5,10). The most compact was:

0*	If-Equal (1, 0, 2, 3)	3	Sub (4, 5)
1	Wall Filter (0)	4	If-Dot (4, 3, 0, 1)
2	Empty Radar	5	Wall Sum

3.2 Maze7

The Maze7 environment (Figure 1.(c)) was discussed in [22,21] and again contains only one pair of aliased positions. However, one of the paths must pass through both ambiguous states. While the longest solution path for the Woods101 environment is 4, the longest solution path in Maze7 is 7, and the average is $4.\overline{3}$. Thus Maze7 is considered more challenging than Woods101. One discovered perfect solution, set at (17,34), was:

0*	Space Filter (1)	4	Delay (6)
1	If-Dot (2, 3, 2, 1)	5	Empty Radar
2	Empty Radar	6	Time Weighted Accumulator (7)
3	Rotation ERC [$3\pi/8$] (4)	7	If-Greater (6, 5, 7, 5)

3.3 MazeF4

The MazeF4 environment (Figure 1(d)) is a rotation of Maze7 plus an extra non-ambiguous empty space. The optimal solution has an average path length of 4.5 and a maximum of 7 steps. In the fifty runs of our experiment we did not discover an optimal solution; but in other experiments we discovered several, including the following (8,16) program:

0*	If-Dot (1, 2, 3, 4)	9	Wall Radar
1	Space Filter (5)	10	Vector ERC [-0.9425225, 0.3341425]
2	Space Filter (6)	11	Sub (12, 13)
3	Empty Radar	12	If-Equal (14, 3, 15, 0)
4	If-Dot (7, 8, 8, 4)	13	Vector ERC [-0.6436193, 0.7653458]
5	Add (9, 9)	14	If-Dot (12, 13, 16, 15)
6	Rotation ERC [$7\pi/8$] (10)	15	Rotation ERC [$-\pi/8$] (3)
7	Delay (11)	16	Time Weighted Accumulator (16)
8	Empty Radar		

3.4 E1 and E2

The E1 and E2 environments were introduced in [18]. While the other mazes studied in this work consist of narrow corridors only, the E1 and E2 environments feature wide-open areas. E2 is an empty environment with a food cell in the center. There are 36 perceptually aliased positions belonging to five different perception classes. The optimal solution in E2 is $2.9791\overline{6}$ steps on average and 5 steps maximum. We discovered no optimal solutions for E2. The closest, an (11,22) program, is only suboptimal by two steps. E2 is shown in Figure 1(f) marked with the suboptimal steps. The program is below:

0*	Time Weighted Accumulator (1)	9	If-Greater (3, 45, 13, 14)
1	If-Greater (2, 3, 2, 4)	10	Delay (15)
2	To8 (5)	11	Wall Radar
3	Time Weighted Accumulator (6)	13	Space Filter (1)
4	Empty Radar	14	Accumulator (18)
5	Empty Radar	15	Empty Sum
6	Sub (7, 8)	18	Sub (19, 15)
7	Add (9, 10)	19	Vector ERC [0.0, -1.0]
8	Rotation ERC [$\pi/2$] (11)	45	Vector ERC [1.0, -1.0]

E1(Figure 1(e)) is a significantly easier version of E2, with only 20 aliased positions belonging to nine classes. For a more intuitive depiction of the perception classes, see [13]. The average optimal path length is $2.8\overline{1}$, and the maximum is 4. We found twelve optimal solutions for E1, including the following (11,22) program:

0*	Accumulator (1)	4	Empty Radar
1	If-Greater (2, 3, 4, 1)	5	Add (5, 4)
2	Add (5, 0)	6	Wall Filter (5)
3	If-Greater (2, 0, 6, 3)		

3.5 Maze10

This environment was introduced in [17]. There are seven perceptually aliased positions split into three classes, not counting ambiguous cells that require the same action. Although only one bit of memory is theoretically enough to solve the problem, it has proven prohibitively difficult [23]. The average optimal path length is 5, and the maximum is 8. Because the optimal solution contains an eight-step path, our customary *max-walk* threshold of 10 steps used so far is too small, and so we increased it to 25.

We did not find an optimal program for this problem. Our best solution, with an average optimal path length of $6.1\overline{6}$ steps, was the following (13,26) program:

0*	If-Equal (1, 2, 3, 1)	6	Sub (3, 12)
1	To8 (3)	7	If-Greater (3, 8, 12, 8)
2	To8 (4)	8	Empty Radar
3	If-Dot (5, 6, 7, 8)	9	Wall Sum
4	If-Dot (9, 9, 10, 1)	10	Empty Radar
5	Rotation ERC [$5\pi/8$] (12)	12	Vector ERC [0.99563, 0.09339]

Figure 1(g) shows the path lengths for this solution, while Figure 1(h) shows the path lengths for an optimal solution.

4 Conclusions and Future Work

One of neural programming's most interesting and attractive features is the cyclic nature of its representation. Surprisingly, previous work in NP [3] does not rely on this feature at all; indeed recurrence in the graph is more or less relegated to providing modularity and iterative improvement of an existing solution. In this paper our goal was to demonstrate that NP could also make use of its recurrence to solve problems requiring internal state.

In order to demonstrate the generality of the mechanism, we chose a problem which we knew beforehand that NP was not suited to. We tested NP on various non-Markovian problems more commonly used for policy-learning methods such as learning classifier systems. These problems ranged widely in difficulty. Accordingly, as expected, our success rates are lower than those reported in the

LCS community and the number of evaluations required are higher. Still, in most of the problems we were able to find optimal solutions; in the remainder we found near-optimal solutions.

In future work, our next goal is to demonstrate the efficacy of NP in a very different environment for which LCS or FSAs are *not* suited but NP is: one requiring recurrent mathematical calculation and internal state: for example, a problem solvable only by partial differential equations. One such problem domain that we are planning to tackle is a two-pole balancing problem with only positional information provided.

Another area of future work we hope to pursue is how to add recursion and expandable internal state to the neural programming paradigm in a way which is tractable. While we may argue for the generality of the neural programming method in terms of application (similar to the variety of problems to which GP is applied), and while NP can perform iteration and use finite amounts of internal state, nonetheless we have not investigated how to get NP to solve problems requiring a stack. Last, we hope to investigate the application of more traditional GP methods (ADFs with indexed memory, etc.) against such problem domains as the one described. Genetic programming has always held an implied promise of discovery of arbitrary computer programs to solve problems and we hope this work might further this effort.

Acknowledgments. We thank Keith Sullivan for his help in the paper preparation, and the reviewers for their helpful suggestions.

References

1. Fogel, L.: Intelligence Through Simulated Evolution: Fourty Years of Evolutionary Programming. Wiley Series on Intelligent Systems (1999)
2. Koza, J.R.: Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge, MA (1992)
3. Teller, A.: Algorithm Evolution with Internal Reinforcement for Signal Understanding. PhD thesis, Carnegie Mellon University (1998)
4. Koza, J.R., Bennett III, F.H., Andre, D., Keane, M.A.: Genetic Programming III - Darwinian Invention and Problem Solving. Morgan Kaufmann (1999)
5. O'Neill, M., Ryan, C.: Under the hood of grammatical evolution. In Banzhaf, W., Daida, J., Eiben, A.E., Garzon, M.H., Honavar, V., Jakiela, M., Smith, R.E., eds.: Proceedings of the Genetic and Evolutionary Computation Conference. Volume 2., Orlando, Florida, USA, Morgan Kaufmann (1999) 1143–1148
6. Angeline, P.J.: Multiple interacting programs: A representation for evolving complex behaviors. Cybernetics and Systems **29** (1998) 779–806
7. Kantschik, W., Dittrich, P., Brameier, M., Banzhaf, W.: Meta-evolution in graph GP. In Poli, R., Nordin, P., Langdon, W.B., Fogarty, T.C., eds.: Genetic Programming, Proceedings of EuroGP'99. Volume 1598., Springer-Verlag (1999) 15–28
8. Kantschik, W., Banzhaf, W.: Linear-graph GP – a new GP structure. In Foster, J.A., Lutton, E., Miller, J., Ryan, C., Tettamanzi, A.G.B., eds.: Proceedings of the Fifth European Conference on Genetic Programming (EuroGP-2002). Volume 2278 of LNCS., Kinsale, Ireland, Springer Verlag (2002) 83–92

9. Green, F.B.: Performance of diploid dominance with genetically synthesized signal processing networks. In Bäck, T., ed.: Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA97), San Francisco, CA, Morgan Kaufmann (1997) 615–622
10. Katagiri, H., Hirasawa, K., Hu, J., Murata, J.: Network structure oriented evolutionary model-genetic network programming-and its comparison with genetic programming. In Goodman, E.D., ed.: 2001 Genetic and Evolutionary Computation Conference Late Breaking Papers, San Francisco, California, USA (2001) 219–226
11. Wilson, S.W.: Knowledge growth in an artificial animal. In: Proceedings of the International Conference on Genetic Algorithms and Their Applications, Pittsburgh, PA (1985) 16–23
12. Wilson, S.W.: The animat path to AI. In Meyer, J.A., Wilson, S.W., eds.: Proceedings of the First International Conference on Simulation of Adaptive Behavior (From animals to animats). (1991) 15–21
13. Landau, S., Sigaud, O., Picault, S., Gérard, P.: An experimental comparison between ATNoSFERES and ACS. In Stolzmann, W., Lanzi, P.L., Wilson, S.W., eds.: IWLCS-03. Proceedings of the Sixth International Workshop on Learning Classifier Systems. LNAI, Chicago, Springer (2003)
14. Holland, J.H.: Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In Michalski, R.S., Carbonell, J.G., Mitchell, T.M., eds.: Machine Learning: An Artificial Intelligence Approach: Volume II. Kaufmann, Los Altos, CA (1986) 593–623
15. Wilson, S.W.: ZCS: A zeroth level classifier system. Evolutionary Computation **2** (1994) 1–18
16. Cliff, D., Ross, S.: Adding Temporary Memory to ZCS. Adaptive Behavior **3** (1995) 101–150
17. Lanzi, P.L.: An analysis of the memory mechanism of XCSM. In Koza, J.R., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo, M., Fogel, D.B., Garzon, M.H., Goldberg, D.E., Iba, H., Riolo, R., eds.: Genetic Programming 1998: Proceedings of the Third Annual Conference, University of Wisconsin, Madison, Wisconsin, USA, Morgan Kaufmann (1998) 643–651
18. Metivier, M., Lattaud, C.: Further comparison between ATNoSFERES and XCSM. In Stolzmann, W., Lanzi, P.L., Wilson, S.W., eds.: Proceedings of the Fifth International Workshop on Learning Classifier Systems, Springer (2002) 143–163
19. Kim, D., Hallam, J.C.T.: An evolutionary approach to quantify internal states needed for the woods problem. In Hallam, B., Floreano, D., Hallam, J.C.T., Hayes, G., Meyer, J.A., eds.: From Animals to Animats, MIT Press (2002) 312–322
20. Luke, S. ECJ 11: A Java EC research system. http://cs.gmu.edu/~eclab/projects/ecj/ (2004)
21. Lanzi, P.L., Wilson, S.W.: Toward optimal classifier system performance in non-markov environments. In: Evolutionary Computation. Volume 8. (2000) 393–418
22. Lanzi, P.L.: Adding Memory to XCS. In: Proceedings of the IEEE Conference on Evolutionary Computation (ICEC98), IEEE Press (1998)
23. Landau, S., Picault, S., Sigaud, O., Gérard, P.: Further comparison between ATNoSFERES and XCSM. In Stolzmann, W., Lanzi, P.L., Wilson, S.W., eds.: IWLCS-02. Proceedings of the Fifth International Workshop on Learning Classifier Systems. LNAI, Granada, Springer (2002)

Evolving En-Route Caching Strategies for the Internet

Jürgen Branke[1], Pablo Funes[2], and Frederik Thiele[1]

[1] Institute AIFB, University of Karlsruhe, 76128 Karlsruhe, Germany
branke@aifb.uni-karlsruhe.de
[2] Icosystem Corp., 10 Fawcett ST. Cambridge MA 02138 USA
pablo@icosystem.com

Abstract. Nowadays, large distributed databases are commonplace. Client applications increasingly rely on accessing objects from multiple remote hosts. The Internet itself is a huge network of computers, sending documents point-to-point by routing packetized data over multiple intermediate relays. As hubs in the network become overutilized, slowdowns and timeouts can disrupt the process. It is thus worth to think about ways to minimize these effects. Caching, i.e. storing replicas of previously-seen objects for later reuse, has the potential for generating large bandwidth savings and in turn a significant decrease in response time.

En-route caching is the concept that all nodes in a network are equipped with a cache, and may opt to keep copies of some documents for future reuse [18]. The rules used for such decisions are called "caching strategies". Designing such strategies is a challenging task, because the different nodes interact, resulting in a complex, dynamic system. In this paper, we use genetic programming to evolve good caching strategies, both for specific networks and network classes. An important result is a new innovative caching strategy that outperforms current state-of-the-art methods.

1 Introduction

The Internet is a distributed, heterogeneous network of servers. Besides other services, it can be regarded as a huge distributed database. Access to documents on the net is mostly based on a strict client-server model: a client computer generates a request, opens up a connection to a server host, and retrieves the document from the server. This naturally creates a lot of network traffic and, in case of congestions, sometimes causes significant response times or *latencies.* Therefore, it is necessary to think about ways to minimize network traffic.

One starting point is the observation that some popular documents are requested all the time, while others are almost never requested. Therefore, it makes sense to store copies *(replicas)* of popular documents at several places in the network. This phenomenon has prompted server companies (e.g., Akamai [1]), to create forms of mirroring to save bandwidth by servicing requests from hosts that are closer, in Internet topology terms, to the clients making the requests.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 434–446, 2004.

However, this solution obviously works only for long-term data access patterns in which a commercial interest can be matched with monetary investments in distributed regions of the globe.

For a broader perspective, observe that when two neighbors on a block request the same document, two independent channels to the remote server hosting the document are created, even though both requesting computers are connected to the same trunk line. The same data is sent over from the server twice, and relayed by a common chain of routers in between. It would make sense, for any of the intermediate hosts, to keep a copy of the document, allowing it to service the second request directly, without having to contact the remote host at all. Clearly, this would result in a dramatic reduction in network traffic and latency.

Proxy servers sometimes have this capability, being able to optimize Internet access for a group of users in a closed environment, such as a corporate office or a campus network. Much better savings and scalability are possible by using this strategy at all levels: If the campus proxy fails to retrieve the page from the cache, or even, if the two requests come from neighboring university campuses in the same city, then a node further down the chain would have the opportunity of utilizing its cache memory, for the same opportunity exists in every single router on the Internet.

The difficult question is to decide which documents to store, and where to store them. With finite memory, it is impossible for individual hosts to cache all the documents they see.

A global policy control in which a centralized decision-making entity distributes replicas among servers optimally is impractical, for several reasons: the tremendous complexity, because the Internet is dynamically changing all the time, and because no global authority exists. Thus, each server has to decide independently which documents it wants to keep a replica of. The rules used for such decisions are also known as *caching strategies.*

Today, many routers with caching — such as a campus network proxy — use the well-known LRU (Least Recently Utilized) strategy: objects are prioritized by the last time they were requested. The document that has not been used for the longest time is the first to be deleted. Although this makes sense for an isolated router, it is easy to see why LRU is not an optimal policy for a *network* of caching hosts. In our example above, all the intermediate hosts between the two neighbors that requested the same document, and the server at the end of the chain, will store a copy of the document because a new document has the highest priority in LRU. However, it would be more efficient if only one, or a few, but not all intermediate nodes kept a copy. In isolation, a caching host tries to store all the documents with highest priority. In a network, a caching host should try to cache only those documents that are not cached by its neighbors.

Designing good caching strategies for a network is a non-trivial task, because it involves trying to create global efficiency by means of local rules. Furthermore, caching decisions at one node influence the optimal caching decisions of the other nodes in the network. The problem of cache similarity of the above example is one of *symmetry breaking:* when neighbors apply identical, local-information based strategies, they are likely to store the same documents in their caches.

In this scenario, the network becomes saturated with replicas of the same few documents, with the consequent degradation of performance.

In this paper, we attempt to design good caching strategies by means of genetic programming (GP). As we will show, GP is able to evolve new innovative caching strategies, outperforming other state-of-the-art caching strategies on a variety of networks.

The paper is structured as follows: first, we will cover related work in Section 2. Then, we describe our GP framework and the integrated network simulator in Section 3. Section 4 goes over the results based on a number of different scenarios. The paper concludes with a summary and some ideas for future work.

2 Related Work

There is a huge amount of literature on caching at the CPU level (see e.g., [17]). Caching on a network, or web caching, has only recently received some attention. For a survey see [20,7].

Generally, the literature on web caching can be grouped into two categories:

1. Centralized control: this is the idea of a central authority overseeing the entire network, deciding globally which replicas should be stored on which server. Thereby, it is usually assumed that the network structure and a typical request pattern are known.
2. Decentralized control: in this group, decentralized caching strategies are proposed, i.e., rules that allow each server to independently decide which recently seen documents to keep. Since these rules are applied on-line, it is important that they can be processed efficiently and that they do not cause additional communication overhead.

The centralized control version is also known as the "file allocation problem". For a classification of replica placement algorithms, see [10]. In [13], an evolutionary algorithm is used to find a suitable allocation of replicas to servers. Besides retrieval cost, [16, 15] additionally consider the aspect of maintaining consistency in the case of changes to the original document, an aspect which we deliberately ignore in our paper. In any case, while the centralized approach may be valid for small networks, it becomes impracticable for larger networks, as the data analysis, the administrative costs, and the conflict between local authorities take over.

Given the difficulties with a centralized approach, in in this paper we focus on decentralized control. We try to find simple strategies that can be applied independently on a local level, thereby making a global control superfluous. This approach is more or less independent of the network structure and request pattern, and can thus much quicker adapt in a changing environment.

Early work on web caching has simply used or adapted traditional caching strategies from the CPU level, such as *LRU* and *LFU* (Least Frequently Used), see e.g. [21]. The disadvantages of these, namely that they lead to cache symmetry, were described in the introduction. An example of a network-aware caching strategy is *GDSF* (Greedy Dual Size Frequency), which considers the number of

times a particular document has been accessed, its size and the cost to retrieve that document from a remote server (in our case, the distance, measured in hops, to the server holding the next replica).

GDSF's cost-of-retrieval factor avoids the cache repetition problem by reducing the priority of documents that can be found in nearby caches. It has been shown to be among the best caching strategies for networks [6,9]. Another comparison of several web caching strategies can be found in [2].

An interesting alternative has recently been suggested in [18]. There, a node holding a document and receiving a request uses a dynamic programming based method to determine where on the path to the requesting node replicas should be stored. While this approach certainly holds great potential, it requires that all nodes in the network cooperate, and the computation of the optimal allocation of replicas to servers is time-consuming. Furthermore, request frequency information must be stored not only for documents in the cache, but all documents ever seen.

As has already been noted in the introduction, in this paper we attempt to evolve such a decentralized caching strategy by means of GP. A previous example of using GP for the design of caching strategies was demonstrated by Paterson and Livesey [14] for the case of the instruction cache of a microprocessor. Some of their fundamental ideas are similar to ours: GP is a tool that can be used to explore a space of strategies, or algorithms for caching. However, the nature of the CPU cache is different from the problem of distributed network caching because it does not involve multiple interconnected caches.

3 A Genetic Programming Approach to the Evolution of Caching Strategies

In this section, we will describe the different parts of our approach to evolve caching strategies suitable for networks. We will start with a more precise description of the assumed environment and the network simulator used, followed by the GP implementation.

3.1 Network Simulator

The overall goal of our study was to evolve caching strategies for complex data networks like the Internet. However, for testing purposes, we had to design a simplified model of the network.

Edges and Nodes. Our network consists of a set of servers (nodes), connected through links (edges). Each server has a number of original documents (which are never deleted), some excess storage space that can be used for caching, and a request pattern. We assume that the shortest paths from each server to all other servers as well as the original locations of all documents are known. Thereby, we are obviating the problem of *routing* and focusing only on *caching* decisions.

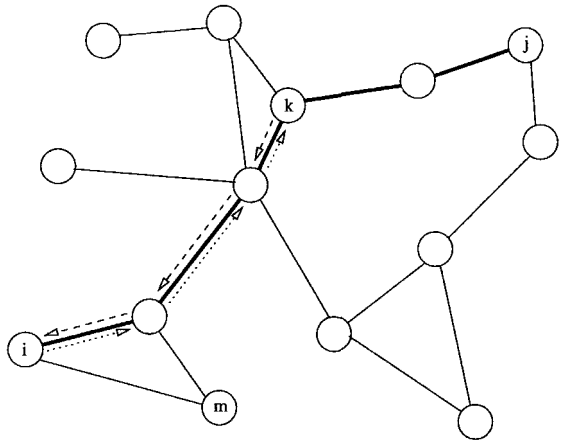


Fig. 1. If node i requests a document from server j, the request is sent along the shortest path (bold), and the first replica found on this path is sent back (say from node k). Other possible replicas not on the path are ignored (e.g., a replica on node m)

When a server requests a document from a remote server, it sends out out this request along the route (shortest path) to the remote server that is known to keep the original copy of the document. All nodes in between check whether they have a cached replica of the requested document. If not, the request is passed through. When either a cached replica or the original document has been found, the document is sent back to the requesting server, following the inverse route. All the intermediate nodes receive the document's packets and pass them along the path to the next node, until the document arrives at its destination. An example is depicted in Figure 1.

Bandwidth, Queuing and Packetizing. When a document travels through the network, it is divided into many small packets. Each link has an associated bandwidth, being able to deliver a limited number of bytes per second. Excess packets wait in an infinite FIFO queue until they can be serviced. We simplified the network operation somewhat by ignoring timeouts.

Efficiency. The main goal is to minimize the average latency of the requests, which is defined as the average time from a request until the arrival of (the last packet of) the document.

3.2 Evolving Caching Strategies

Caching as Deletion. It is easy to see that servers which have not yet filled up their memories, should store every single document they see. Even if an oracle was able to tell that a particular object will never be accessed again, there would

Table 1. Functions used for GP

Function	Meaning	Function	Meaning
add(a,b)	$a + b$	sin(a)	$\sin(a)$
sub(a,b)	$a - b$	cos(a)	$\cos(a)$
mul(a,b)	$a \cdot b$	exp(a)	e^a, $a \in [-100, 100]$
div(a,b)	$\begin{cases} \frac{a}{b} & : (b \neq 0) \\ 1 & : (b = 0) \end{cases}$	iflte(a,b,c,d)	$\begin{cases} c & : (a < b) \\ d & : (a \geq b) \end{cases}$

be no harm in storing it. The problem comes when the caching memory fills up, and a storage action requires the deletion of some other previously cached object. Cache machines have finite disk space and so they must eventually discard old copies to make room for new requests.

In order to evolve caching strategies we thus focused on the problem of deletion. The nodes in our simulated networks store all the documents they receive, until their caches are filled up. If, however, there is not enough memory available to fit the next incoming one, some space must be freed up. All cached objects are sorted according to a priority function, and the document with the lowest priority is trashed. The operation is repeated until enough documents have been deleted so that there is enough space to save the newcomer. In the remainder of this paper we shall define *caching strategy* as the priority rule used for deletion.

Genetic Programming. Given the difficulties to define a restricted search space for caching strategies, we decided to use genetic programming [11, 12], which allows an open-ended search of a space of more or less arbitrary priority functions.

In order to explore the space of caching policies, we employed a generic set of GP functions (Table 1), combined with a set of terminals representing the local information available about each document (Table 2). Each node collects this *observable* information that can be gathered from the objects themselves as they are sent, received and forwarded. That is, for example, why the distance we use is the number of hops a document traveled before reaching the host (observed distance) as opposed to the true distance to the nearest replica, which could only be determined through additional communication.

Another important decision was to avoid measures that need to be recomputed before each usage. This allows a host to maintain its cache sorted rather than re-computing and re-sorting before each deletion. Access frequency, therefore, is calculated as one over the average mean time between accesses, at the time of the last access (see [2]).

Since our focus was more on the application than the search for optimal parameter settings, we used rather standard settings for the test runs reported below: GP has been run with a population size of 60 for 100 generations, the initial population has been generated randomly with depth of at most 6. Tournament selection with tournament size of 2 was used, and a new population was generated in the following way:

Table 2. Terminals used for GP

Variable	Long name	Meaning
A	timeCreated	Time when replica was stored in cache
B	size	Size of document in kilobytes
C	accessCount	Number of times the document has been accessed
D	lastTimeAccessed	Time of last access to document
E	distance	Distance from node that sent the document (in number of hops)
F	frequency	Observed frequency of access (in accesses per second)
$\Re$		Random constant

1/3 of the individuals were simply transferred to the next generation
1/3 of the individuals were generated by crossover
1/3 of the individuals were generated by mutation only.

Crossover was the usual swapping of sub-trees, mutation replaced a sub-tree by a new random tree.

3.3 Evaluating Caching Strategies

Evaluating caching strategies analytically is difficult. Instead, we tested them, using the simulation environment described in Section 3.1. There are two possible scenarios: if the network topology, and the location and request patterns of all documents are known, GP can be used to tailor a caching strategy exactly to the situation at hand. Evaluation is deterministic, as we can simulate the environment exactly and simply test the performance of a particular caching strategy in that environment. We made preliminary tests with fixed networks and obtained excellent results (not included in this paper due to space restrictions).

On the other hand, such topology and patterns may be known only approximately (e.g., "the document request frequencies follow a scale-free distribution"). We would like to evolve caching strategies that perform well in a whole class of possible scenarios. In other words, we are looking for a solution that applies in general to all networks within the range of characteristics we expect to find in the real world. Since it is impossible to test a caching strategy against all possible networks, we defined classes of scenarios and tested each caching strategy against a random representative from that class. Of course, that made the evaluation function stochastic. Following observations from other evolutionary algorithms used for searching robust solutions [4], we decided to evaluate all individuals within one generation by the same network/request pattern, changing the scenario from generation to generation. Naturally, elite individuals transferred from one generation to the next have to be re-evaluated.

Note that using a simulation for evaluation is rather time consuming. A single simulation run for the larger networks used in our experiments takes about 6 minutes. Thus, even though we used a cluster of 6 Linux workstations with 2 GHz each, and even though we used a relatively small population size and only 100 generations, a typical GP run took about 5 days.

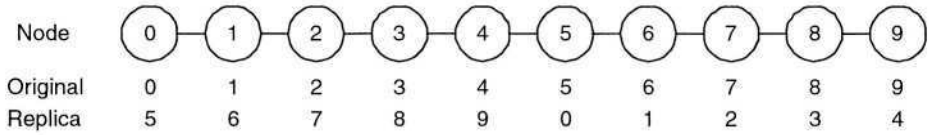
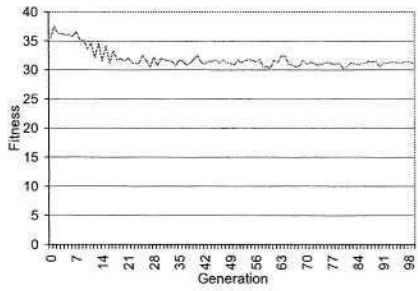
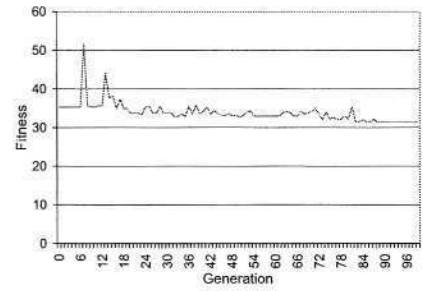


Fig. 2. Optimal distribution of replicas on a simple linear network



(a) Fitness of best individual per generation



(b) Performance of best individual on 10 other request patterns

Fig. 3. Exemplary GP-run on linear network

4 Results

4.1 Linear Networks

First, we tested our approach by evolving strategies for linear networks, for which we were able to determine the optimal placement of replicas, analytically.

The first test involved a purely linear network with 10 nodes connected in a line (i.e. the first and last node have exactly one neighbor, all other nodes have exactly two neighbors). Every node has one original document, each document has the same size, and each node has excess memory to store exactly one additional document. The request pattern is uniform, i.e. each document is requested equally often (on average). Then, the optimal distribution of replicas is depicted in Figure 2 (for a proof see [19]).

A typical test run is shown in Figure 3, where part (a) shows the observed average latency of the best individual in each generation, as observed by GP, and part (b) shows the performance of these individuals on 10 additional random request patterns to test how the solution generalizes. The caching strategy evolved is rather complex and difficult to understand. Its performance, however, was excellent.

Table 3 compares the latency of different caching strategies over 30 additional tests with different random request patterns. Note that GP is a randomized method, and starting with different seeds is likely to result in different caching

Table 3. Average latency of different caching strategies on the linear network ± standard error.

Caching Strategy	Ø latency
OPTIMAL	31.58 ± 0.03
BESTGP	31.98 ± 0.06
GDSF	47.67 ±1.17
DISTANCE	50.40 ± 1.24
RANDOM	61.65 ± 0.14
LRU	74.77 ± 0.19

strategies. Therefore, we run GP 5 times, and used the average performance over the resulting 5 caching strategies as result for each test case. OPTIMAL is the latency observed with the optimal distribution of replicas on this linear network, which is a lower bound. As can be seen, the strategy evolved by GP (BESTGP) was able to find a solution that performs very close to the lower bound. Other standard caching strategies as GDSF or LRU, perform poorly, LRU even worse than RANDOM,[1] where it is randomly decided whether to keep or delete a particular document. Looking at distance only (DISTANCE) is better than looking at the last time accessed only (LRU), but also much worse than the evolved strategy.

4.2 Scale-Free Networks

The Internet has a scale-free structure, with a few servers being highly connected, and many servers having only few connections [8,22,3]. The tests in this subsection are obtained using a scale-free network with 100 nodes. We used the method described in [5] to generate random scale-free networks with similar characteristics as the Internet. There are 100 original documents with document size between 0.1 and 2 MB. In each of 1000 simulated seconds, an average of 1100 requests are generated in a simulated Poisson process. Request frequencies are also scale-free: some documents are requested much more often than others.

With the characteristics of the network defined only as distributions, we searched for individual strategies that could perform well over a wide range of networks in the class. As explained in Section 3.3, this was achieved by using a different random network in each generation.

As it turned out, the observed latencies varied dramatically from one simulation to another, the reason being that large latencies are generated when one or more edges receive more requests than their bandwidth allows them to service. As predicted by queuing theory, a link can be either underutilized (bandwidth > request load) or overutilized (bandwidth < request load). In the first case, latency remains close to zero, and in the second it grows to infinity. We have deliberately set the parameters in our simulation so as to generate networks close

[1] A random strategy, in spite of being completely blind, has the advantage of breaking symmetry (cf. page 435) because all hosts use different random number seeds, thus their caches tend to complement each other

Table 4. Comparison of different caching strategies on the class of scale-free networks with 100 nodes. Table shows average rank according to latency ± standard error.

Caching Strategy	Ø rank(latency)
RUDF	1.13 ± 0.06
GDSF	2.80 ± 0.15
DISTANCE	3.10 ± 0.28
LRU	3.70 ± 0.16
RANDOM	4.27 ± 0.14

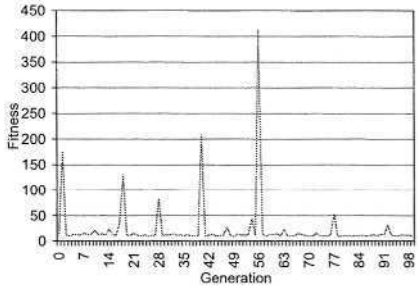
to the saturation point. Depending on the random topology and request pattern, a network can be underutilized, leading to latencies in the order of milliseconds, or overutilized, leading to latencies of of several seconds.

GP had some difficulties with this high variance, which led us to evaluate each individual three times per generation, and take the average as fitness. Even then, the randomness was substantial, as can be seen in the oscillations in performance of the best solution in Figure 4 (again, part (a) shows the fitness of the best individual as observed by GP, while part (b) shows the performance of that individual on 20 additional tests). Nevertheless, the results obtained were convincing. Although GP again generated some rather good but complicated caching strategies, in one run it came up with a surprisingly simple strategy that could be further simplified to the following:

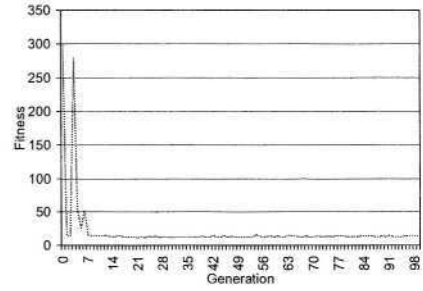
$$priority = lastTimeAccessed \cdot (distance + accessCount)$$

This result is quite astonishing as it adds two totally different parameters, the distance the document has traveled and the number of times it has been accessed. But the combination has meaning: it keeps the documents that have either a very high access count, or a very high distance, but only if they have been accessed recently (otherwise lastTimeAccessed would be small). We call this newly evolved strategy *RUDF* for "Recently Used Distance plus Frequency". This rule performs very well in comparison to all other caching strategies tested (see Table 4), and thus sets a new benchmark. Again, we used 30 test runs to evaluate a strategy, with a new random network and request pattern generated for each test instance. Due to the large variance between test instances, a comparison based on mean latency has only limited explanatory power. We therefore used a non-parametric measure, the average rank the caching strategy achieved when compared to the other strategies tested (with 1 being the best rank, and 5 being the worst rank). The results show that RUDF works well over the randomly generated range of networks in the considered class of scale-free networks.

In order to test how RUDF would perform on completely different networks, we additionally tested it on the linear network from above, and on two scale-free network classes with 30 and 300 nodes, respectively. On the linear network, it resulted in a latency of 35.8 ± 0.42, i.e. worse than the caching strategy evolved particularly for the linear network, but still better than all other tested strategies. The results on the two scale-free network classes are shown in Tables 5 and 6. As



(a) Fitness of best individual per generation



(b) Performance of best individual on 20 other instances

Fig. 4. Exemplary GP-run on the class of scale-free networks

Table 5. Comparison of different caching strategies on the class of scale-free networks with 30 nodes. Table shows average rank according to latency ± standard error.

Caching Strategy	Ø rank(latency)
RUDF	1.03 ± 0.03
GDSF	1.97 ± 0.03
LRU	3.03 ± 0.03
RANDOM	4.20 ± 0.07
DISTANCE	4.70 ± 0.10

Table 6. Comparison of different caching strategies on the class of scale-free networks with 300 nodes. Table shows average rank according to latency ± standard error.

Caching Strategy	Ø rank(latency)
RUDF	1.03 ± 0.03
DISTANCE	2.23 ± 0.16
GDSF	3.20 ± 0.11
RANDOM	4.20 ± 0.16
LRU	4.33 ± 0.13

can be seen, RUDF significantly outperforms all other strategies independent of the network size. DISTANCE seems to be very dependent on the network size, it performs second on large networks, but worse than RANDOM on smaller ones.

5 Conclusions and Future Work

A key inefficiency of the Internet is the tendency to retransmit a single blob of data millions of times over identical trunk routes. Web caches are an attempt to reduce this waste by storing replicas of recently accessed documents at suitable locations. Caching reduces network traffic as well as experienced latency.

The challenge is to design caching strategies which, when applied locally in every network router, exhibit a good performance from a global point of view. One of the appeals of GP is that it can be used to explore a space of algorithms. Here, we have used GP to search for caching strategies in networks that resemble the Internet, with the aim to find strategies that minimize latency. A new rule called RUDF was evolved, which is very simple yet outperformed all other tested caching strategies on the scenarios examined.

An important obstacle we faced was measuring fitness, because fitness could only be determined indirectly through simulation, and different random seeds resulted in a high variance in latency (the criterion we used as fitness). Nevertheless, averaging and multiple-seed evaluation techniques allowed us to evolve robust strategies that are efficient in a wide variety of conditions.

There are ample opportunities to extend our research: first of all, it would be necessary to test the newly evolved caching strategy on a larger variety of networks. Then, the caching strategies could be made dependent on node characteristics (e.g., location in network, number of connections), moving away from the assumption that all nodes should apply identical caching strategies. Finally, we could have independent GPs running on every node, so that the caching strategies on the different nodes would coevolve.

References

1. http://www.akamai.com/en/html/services/edgesuite.html.
2. H. Bahn, S. H. Noh, S. L. Min, and K. Koh. Efficient replacement of nonuniform objects in web caches. *IEEE Computer,* 35(6):65–73, 2002.
3. A.L. Barabasi, R. Albert, and H. Heong. Scale-free characteristics of random networks: the topology of the world-wide web,. *Physica A,* 281:2115, 2000.
4. J. Branke. Reducing the sampling variance when searching for robust solutions. In L. Specter et al., editor, *Genetic and Evolutionary Computation Conference (GECCO'01),* pages 235–242. Morgan Kaufmann, 2001.
5. T. Bu and D. Towsley. On distinguishin between internet power law topology generators. Technical report, Depatrment of Computer Science, University of Massachusetts, 2002.
6. L. Cherkasova and G. Ciardo. Role of aging, frequency, and size in web cache replacement policies. In B. Hertzberger, A. Hoekstra, and R. Williams, editors, *High-Performance Computing and Networking,* volume 2110 of *LNCS,* pages 114–123. Springer, 2001.
7. B. D. Davison. A web caching primer. *IEEE Internet Computing,* 5(4):38–45, 2001.
8. M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *Proceedings of the ACM SIGCOMM, Sept 1999.,* 1999.
9. S. Jin and A. Bestavros. Greedydual* web caching algorithm. *Computer Communications,* 24(2): 174–183, 2001.
10. M. Karlsson, C. Karamanolis, and M. Mahalingam. A framework for evaluating replica placement algorithms. Technical Report HPL-2002-219, Hewlett-Packard, 2002.
11. J. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, 1992.
12. J. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs.* MIT Press, Cambridge, 1994.
13. T. Loukopoulos and I. Ahmad. Static and adaptive data replication algorithms for fast information access in large distributed systems. In *International Conference on Distributed Computing Systems,* pages 385–392, 2000.
14. N. Paterson and M. Livesey. Evolving caching algorithms in C by GP. In *Genetic Programming: Proceedings of the Second Annual Conference.* Morgan Kaufmann, 1997.

15. G. Pierre, M. Van Teen, and A. Tanenbaum. Dynamically selecting optimal distribution strategies for web documents. *IEEE Transactions on Computers,* 51(6):637–651, 2002.
16. S. Sen. File placement over a network using simulated annealing. *ACM,* 1994.
17. A. J. Smith. Cache memories. *ACM Computing Surveys,* 14(3):473–530, 1982.
18. X. Tang and S. T. Chanson. Coordinated en-route web caching. *IEEE Transactions on Computers,* 51(6):595–607, 2002.
19. F. Thiele. Evolutionäre Optimierung von Caching Strategien für das Internet. Master's thesis, Institute AIFB, University of Karlsruhe, 76128 Karlsruhe, Germany, 2004.
20. J. Wang. A survey of web caching schemes for the internet. *ACM SIGCOMM Computer Comm. Rev.,* 29(5):36–46, 2001.
21. S. Williams, M. Abrams, C. R. Standridge, G. Abdulla, and E. A. Fox. Removal policies in network caches for World-Wide Web documents. In *Procedings of the ACM SIGCOMM '96 Conference,* Stanford University, CA, 1996.
22. S. Yook, H. Jeong, and A. Barabasi. Modeling the internet s large-scale topology. PNAS October 15, 2002 vol. 99 no. 21, 2002.

Grammatical Constant Creation

Ian Dempsey[1], Michael O'Neill[1], and Anthony Brabazon[2]

[1] Biocomputing and Developmental Systems Group
University of Limerick, Ireland.
Ian.Dempsey@fmr.com, Michael.ONeill@ul.ie
[2] Dept. Of Accountancy, University College Dublin, Ireland.
Anthony.Brabazon@ucd.ie

Abstract. This study examines the utility of grammatical ephemeral random constants, and conducts an analysis of the preferences of evolutionary search when a number of different grammar based constant generation methods are provided with Grammatical Evolution. Three constant generation techniques are supplied, namely, grammatical ephemeral random constants, digit concatenation, and an expression based approach. A number of constant generation problems are tackled to analyse this approach, with results indicating a preference for both the digit concatenation and grammatical ephemeral random constants. The provision of different constant generation strategies allowing the evolutionary process to automatically determine which technique to adopt would, therefore, appear to be advantageous.

1 Introduction

In earlier studies, a digit concatenation approach to constant creation in Grammatical Evolution has been adopted and some investigations into its utility have been conducted [1,2,3]. The findings of these studies provide evidence to support the superiority of the digit concatenation approach across a range of constant creation problems, when compared to an expression based method in which arithmetic operators are required to generate new constants. We now extend these studies with the introduction of a third constant creation technique based on ephemeral random constants called grammatical ephemeral random constants. Within a grammar, we study the benefits of defining all three constant generation methods simultaneously, and allowing evolution to *select* the most appropriate strategy.

Many applications of Genetic Programming require the generation of constants, hence the discovery of efficient means of generating diverse constants is important. The current standard approach to constant generation is to use ephemeral random constants, whose values are created randomly within a prespecified range at runs' initialisation [4]. These values are then fixed throughout a run, and new constants can only be created through combinations of these values and other items from the function and terminal set, such as +,-, * and /.

There have been a number of variations on the ephemeral random constant idea in tree-based GP systems, all of which have the common aim of making

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 447–458, 2004.

small changes to the initial constant values created in an individual. *Constant perturbation* [5] allows GP to fine-tune floating point constants by multiplying every constant within an individual proto-solution by a random number between 0.9 and 1.1, having the effect of modifying a constants value by up to 10% of their original value. *Numerical terminals* and a *numerical terminal mutation* were used in [6] instead of ephemeral random constants, the difference being that the numerical terminal mutation operator selects a real valued numerical terminal in an individual and adds a noise parameter, drawn from a Gaussian distribution, such that small changes are made to the constant values.

A *numeric mutation* operator, that replaces all of the numeric constants in an individual with new ones drawn at random from a uniform distribution within a specified selection range, was introduced in [7]. The selection range for each constant is specified as the old value of that constant plus or minus a temperature factor. This method was shown to produce a statistically significant improvement in performance on a number of symbolic regression problems ranging in difficulty. More recently Ryan and Keijzer [8] have found that it is important to maintain a diverse set of constants within a population, therefore suggesting that the continual introduction of new constants into the population is critical. In fact, their results suggest that the more disruptive the perturbations are to the constants present the more of them that are adopted in successful solutions.

In addition, there have been a number of attempts to fine tune constants using statistical and mathematical methods such as Linear Scaling, Least Mean Squares and Gradient Descent, for example see [9,10].

This contribution is organised as follows. Section 2 provides a short introduction to Grammatical Evolution. Section 3 describes the problem domains examined, and the experimental approach adopted in this study. Section 4 provides the results, and finally, conclusions and an outline of future work are provided in Section 5.

2 Grammatical Evolution

Grammatical Evolution (GE) is an evolutionary algorithm that can evolve computer programs in any language [11,12,13,14,15], and can be considered a form of grammar-based genetic programming. Rather than representing the programs as parse trees, as in GP [4,16,17,18,19], a linear genome representation is used. A genotype-phenotype mapping is employed such that each individual's variable length binary string, contains in its codons (groups of 8 bits) the information to select production rules from a Backus Naur Form (BNF) grammar. The grammar allows the generation of programs in an arbitrary language that are guaranteed to be syntactically correct, and as such it is used as a generative grammar, as opposed to the classical use of grammars in compilers to check syntactic correctness of sentences. The user can tailor the grammar to produce solutions that are purely syntactically constrained, or they may incorporate domain knowledge by biasing the grammar to produce very specific forms of sentences.

BNF is a notation that represents a language in the form of production rules. It is comprised of a set of non-terminals that can be mapped to elements of the set of terminals (the primitive symbols that can be used to construct the output program or sentence(s)), according to the production rules. A simple example BNF grammar is given below, where <expr> is the start symbol from which all programs are generated. The grammar states that <expr> can be replaced with either one of <expr><op><expr> or <var>. An <op> can become either +, -, or *, and a <var> can become either x, or y.

```
<expr> ::= <expr><op><expr> | <var>
  <op> ::= + | - | *
 <var> ::= x | y
```

The grammar is used in a developmental process to construct a program by applying production rules, selected by the genome, beginning from the start symbol of the grammar. In order to select a production rule in GE, the next codon value on the genome is read, interpreted, and placed in the following formula:

$$Rule = Codon\ Value\ \%\ Num.\ Rules$$

where % represents the modulus operator. Beginning from the left hand side of the genome codon integer values are generated and used to select appropriate rules for the left-most non-terminal in the developing program from the BNF grammar, until one of the following situations arise: (a) A complete program is generated. This occurs when all the non-terminals in the expression being mapped are transformed into elements from the terminal set of the BNF grammar, (b) The end of the genome is reached, in which case the *wrapping* operator is invoked. This results in the return of the genome reading frame to the left hand side of the genome once again. The reading of codons will then continue unless an upper threshold representing the maximum number of wrapping events has occurred during the mapping process of this individual, (c) In the event that a threshold on the number of wrapping events has occurred and the individual is still incompletely mapped, the mapping process is halted, and the individual assigned the lowest possible fitness value. A full description of GE can be found in [11].

3 Problem Domain and Experimental Approach

In this study, we compare the utility of three different constant creation methods for evolving constants by defining all three methods within the same grammar, and allowing evolution to select the most appropriate approach. The constant generation problems tackled are; Finding a Static Constant, Finding Dynamic Real Constants, and the Logistic Equation. A description of each problem follows.

Finding a Static Constant. The aim of this problem is to evolve a single real constant, namely 50. Fitness in this case is the absolute difference between the target and evolved values, the goal being to minimise the error.

Finding Dynamic Real Constants. This problem involves a dynamic fitness function that changes its target real constant value at regular intervals (every 10th generation). Two instances of this problem are tackled, the first sets the successive target values to be 192.47, 71.84, 71.83, 173.59, 192.47, and the second instance oscillates between the two values 192.47 and 71.84. The aim with these problems is to analyse the different constant representations in terms of their ability to adapt to a changing environment, and to investigate that behaviour in the event of both small and large changes. As in the static constant problem, fitness in this case is the absolute difference between the target and evolved values, with the goal being the minimisation of this error.

The Logistic Equation. In systems exhibiting chaos, long-term prediction is problematic as even a small error in estimating the current state of the system leads to divergent system paths over time. Short-term prediction however, may be feasible [20]. Because chaotic systems provide a challenging environment for prediction, they have regularly been used as a test-bed for comparative studies of different predictive methodologies [21,22,23]. In this study we use time-series information drawn from a simple quadratic equation, the logistic difference equation.[1] This equation has the form:

$$x_{t+1} = \alpha x_t(1 - x_t) \qquad x \in (0.0, 1.0)$$

The behaviour of this equation is crucially driven by the parameter α. The system has a single, stable fixed point (at $x = (\alpha - 1)/\alpha$)for $\alpha < 3.0$ [23]. For $\alpha \in (3.0, \approx 3.57)$ there is successive period doubling, leading to chaotic behaviour for $\alpha \in (\approx 3.57, 4.0)$. Within this region, the time-series generated by the equation displays a variety of periodicities, ranging from short to long [24]. In this study, three time-series are generated for differing values of α. The choice of these values is guided by [24], where it was shown that the behaviour of the logistic difference equation is qualitatively different in three regions of the range (3.57 to 4.0). To avoid any bias which could otherwise arise, parameter values drawn from each of these ranges are used to test the constant evolution grammars. The goal in this problem is to rediscover the original α value. As this equation exhibits chaotic behaviour, small errors in the predicted values for α will exhibit increasingly greater errors, from the target behaviour of this equation, with each subsequent time step. Fitness in this case is the mean squared error, which is to be minimised. 100 initial values for x_t were used in fitness evaluation, and for each x_t iterating 100 times (i.e. x_t to x_{t+100}).

Constant Creation Grammar. Three constant generation techniques are provided within the same grammar for this study, with the grammar adopted provided below.

[1] This is a special case of the general quadratic equation $y = ax^2 + bx + c$ where $c = 0$ and $a = -b$.

```
<exp> ::= <value>
<value> ::= <trad> | <catR> | <ephemeral>
<op> ::= + | - | / | *
<trad> ::= <trad> <op> <trad> | <tradT>
<tradT> ::= 0.0 | 1.0 | 2.0 | 3.0 | 4.0 | 5.0
          | 6.0 | 7.0 | 8.0 | 9.0
<catR> ::= <cat> <dot> <cat> | <cat>
<cat> ::= <cat> <catT> | <catT>
<catT> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<dot> ::= .
<ephemeral> ::= <ephemeral> <op> <ephemeral> | <ephemeralT>
<ephemeralT> ::= ''100 randomly generated real constants''
```

The concatenation part of the grammar (<cat>) only allows the creation of constants through the concatenation of digits. This is in contrast to the Traditional part of the grammar (<trad>) that restricts constant creation to the generation of values from expressions using a fixed set of constants specified by the non-terminal <tradT>. The third part of the grammar concerns grammatical ephemeral random constants. In this method, a set of 100 real-valued constants are generated randomly in the range 0 to 100 inclusive at the outset of a run and these are then directly incorporated as choices for the nonterminal <ephemeralT>. In a standard GP manner, these constants can then be utilised in arithmetic expressions to generate new constant values, however, unlike in GP these ephemeral random constants can be switched on or off by simply selecting their corresponding production rule thus overcoming potential deletion from the population.

4 Results

For every problem instance, 30 runs were conducted using population sizes of 500, running for 50 generations on the dynamic constant problems, and 100 generations for the static and logistic equation instances, adopting one-point crossover at a probability of 0.9, and bit mutation at 0.1, along with roulette selection and a replacement strategy where the worst performing 25% of the population is replaced each generation.

Finding a Static Constant. The results presented in Fig. 1 indicate a strong preference by GE for the Concatenation method in this problem.
By the final generation, on average, across thirty runs GE evolved 309 individuals using the concatenation method against 126 and 13 for the Ephemeral random constants and the Traditional methods respectively. Of the best performing individuals in the final generation 75% had evolved a Concatenated individual, 14% an Ephemeral random constant individual and 10% a Traditional. It was also worth noting from this set of experiments that after generation four GE consistently mapped each individual of type Concatenation while Ephemeral and

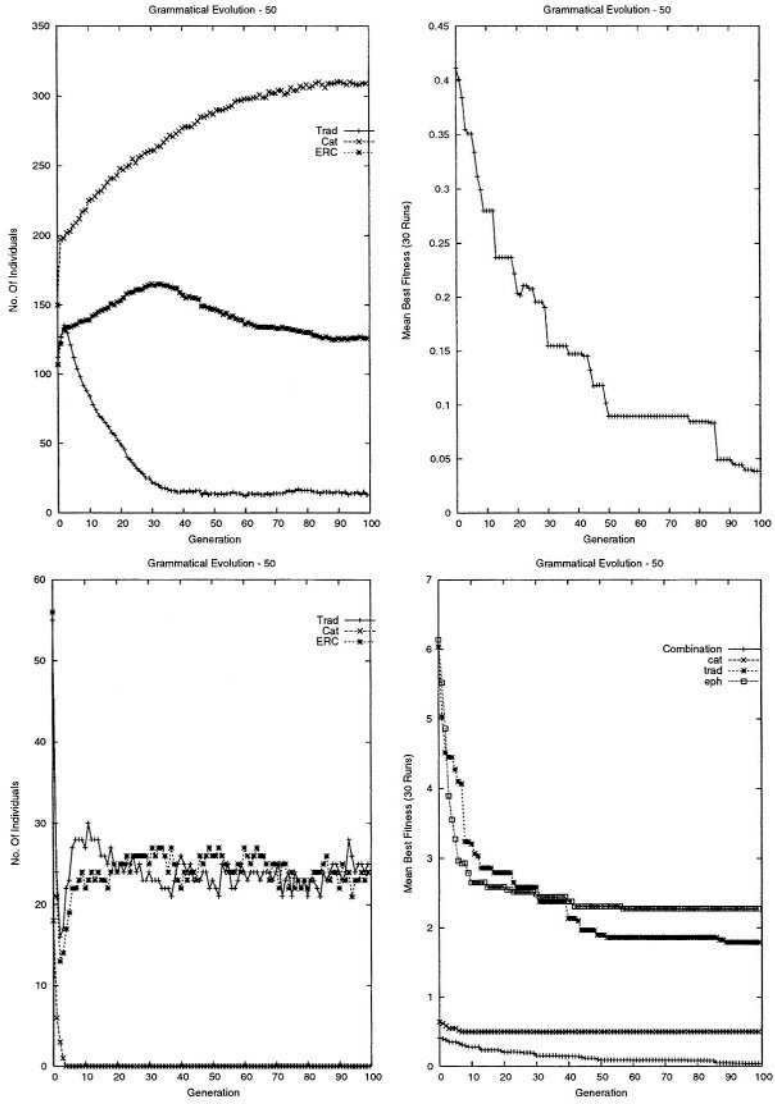


Fig. 1. Plot of the number of individuals that use each of the three constant generation methods (top left), the mean best fitness (top right), and the number of population members using each of the constant creation techniques that fail to map (bottom left), on the static constant problem instance. A comparison of the combination grammar with the performance of the individual component grammars on their own is given (bottom right).

Traditional produced a fluctuating number of un-mappable individuals within the range of 20 to 30 individuals after the early generations.

Among the experiments incorporating each constant creation method exclusively as presented in Fig. 1 the benefits of the Concatenation method are clearly

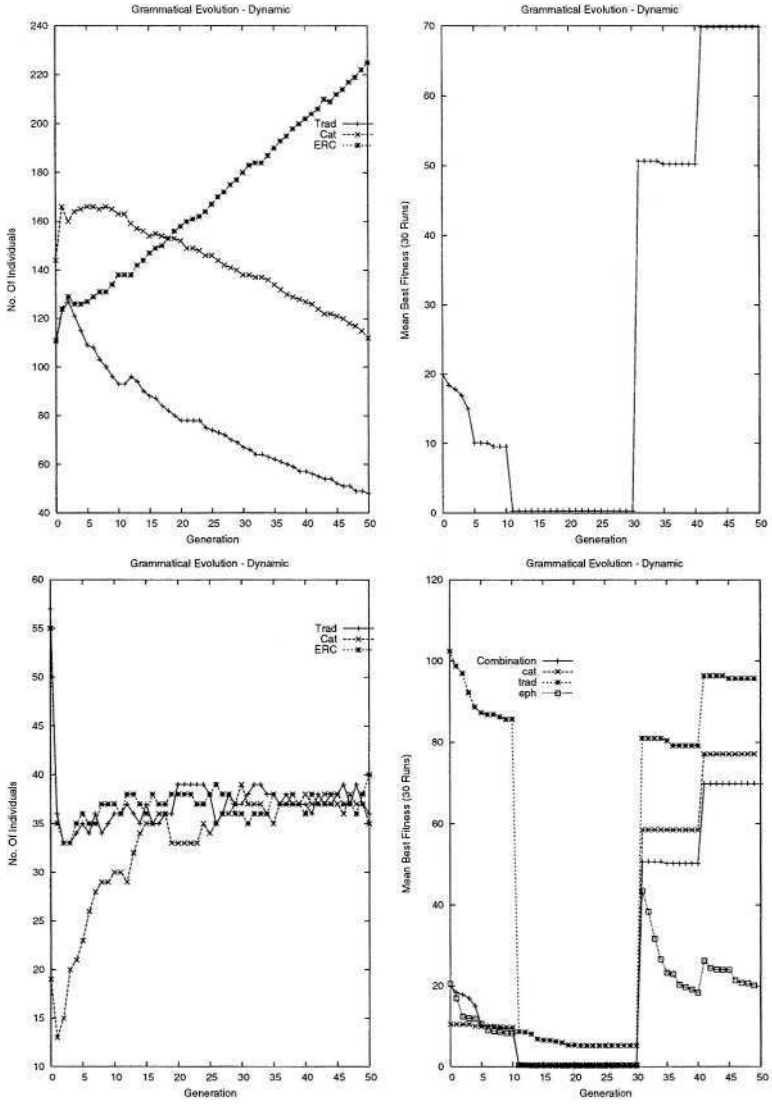


Fig. 2. Plot of the number of individuals that use each of the three constant generation methods (top left), the mean best fitness (top right), and the number of population members using each of the constant creation techniques that fail to map (bottom left), on the first dynamic constant problem instance. A comparison of the combination grammar with the performance of the individual component grammars on their own is given (bottom right).

confirmed for this problem. Over the course of 30 runs Concatenation produced best performers with an average fitness of 0.50024 compared against 1.7931 and 2.27586 for the Traditional and Ephemeral methods respectively.

Finding Dynamic Real Constants. In Fig. 2 graphs are presented for the experiments where the set of numbers to be evolved over the course of a run are: 192.47, 71.84, 71.83, 173.59 and 192.47. This time the Ephemeral constants gain a stronger foothold in the population over the course of the run, overtaking Concatenation before generation 20 at the same time presenting good fitness. However at generation 30, where the target changes to 173.59, this fitness deteriorates significantly. This suggests that while the target was with in the range of the Ephemeral constants it was able to quickly attain a high fitness and a strong position in the population but was unable to successfully evolve from this position once the target left its range.

In the single method grammars however, the Ephemeral method does express a stronger ability to evolve to the targets outside its range taking large evolutionary steps towards the target after its initial change. The Concatenation and Traditional methods present performances similar to the combination grammar's performance.

Results for the oscillating non-stationary problem instance are presented in Fig. 3. In the second instance of this problem where the target oscillates from 192.47 to 71.84 every 10 generations we notice a similar trend. Again by generation 20 Ephemeral constants have reached a strong position within the population after a period with 71.84 as the target. The fitness drops drastically when the target changes to 192.47. When the target reaches the higher number for the third time the fitness is worse again due perhaps to a further loss of diversity in the population.

With the single grammars in the dynamic problem the results for the oscillation experiments provide similar performances with the Ephemeral method being able to take the larger evolutionary steps once the target changes.

The Logistic Equation. Fig. 4 presents a sample of the results for the logistic equation with α values of 3.59, 3.8 and 3.84, which were very similar across the three α values. Here the Concatenation method gains the dominant position within the population as evolution progresses. The proportion of Ephemeral constants in the population is seen to approach the level of Concatenation constants initially, as in the dynamic experiments, however this time as evolution progresses, the Concatenation method gains the dominant position within the population. Among the best performing individuals for 3.59 60% were Ephemeral and 40% Concatenation, for 3.8 73% were Concatenation and with the remaining 27% being Ephemeral and for 3.84 80% Concatenation, 20% Ephemeral. No Traditional individuals achieved best performer in any test.

5 Conclusions and Future Work

Benefits of both the concatenation and newly introduced grammatical ephemeral random constants methods for constant creation are described in this paper with the traditional method performing consistently poorer than the competition for the problems analysed.

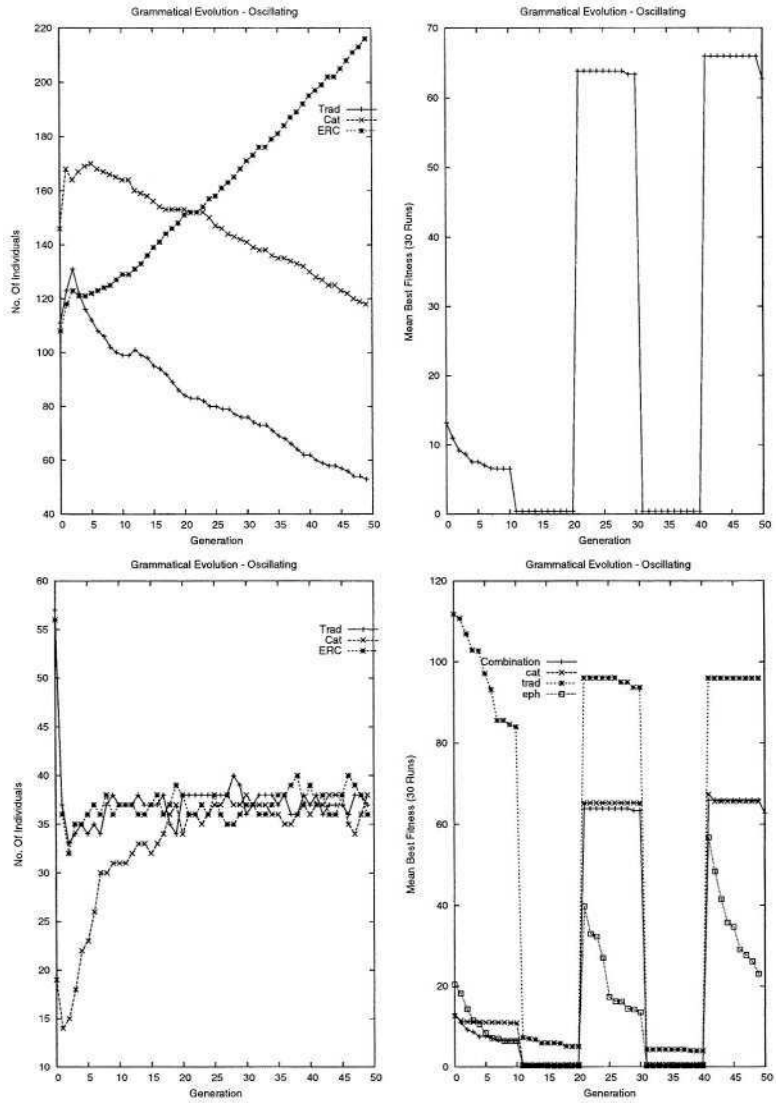


Fig. 3. Plot of the number of individuals that use each of the three constant generation methods (top left), the mean best fitness (top right), and the number of population members using each of the constant creation techniques that fail to map (bottom left), on the second dynamic constant problem instance. A comparison of the combination grammar with the performance of the individual component grammars on their own is given (bottom right).

Given the dominance of the concatenation and grammatical ephemeral random constants techniques on the problem instances tackled here and the fact that there is an unclear winner between these two methods, further analysis is re-

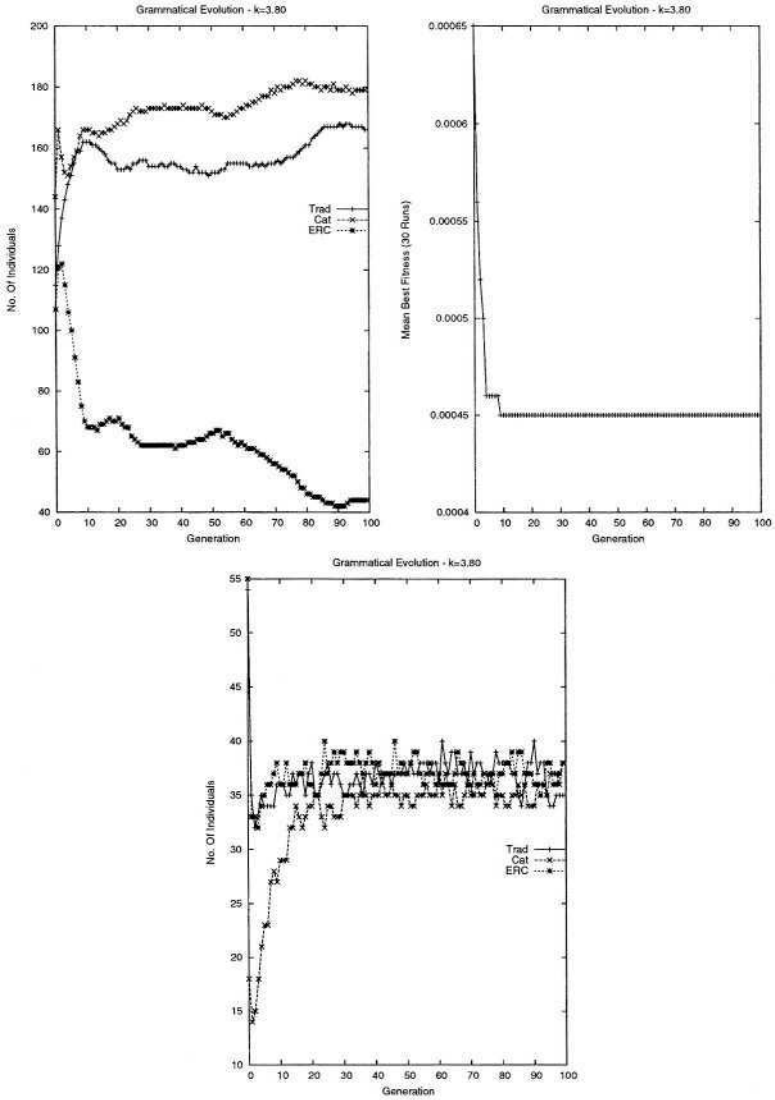


Fig. 4. Plot of the number of individuals that use each of the three constant generation methods (top left), the mean best fitness (top right), and the number of population members using each of the constant creation techniques that fail to map (bottom), on the logistic equation problem instance where α=3.80.

quired on additional domains to ascertain the generality of these findings. It is not clear from this study as to why either of these approaches is superior, just that they are used in preference to the traditional method. Further investigations in this direction are therefore required.

Additionally, future work in this area will focus on the two stronger methodologies in examining their performance in evolving complex numbers outside the grammatical ephemeral random constant range, and where the concatenation method is also allowed to form expressions. A grammar that combines grammatical ephemeral random constants and concatenated constants in expressions will be examined in contrast to the grammar described here, which only allowed the exclusive use of one method for each individual. On the dynamic problem front, a more in-depth analysis of the role diversity plays within such populations is required, and into the number of generations required to improve accuracy when a target is changed.

From the results presented it is recommended that a grammar similar in form to the one in this study is adopted, which provides a facility to switch between the newly introduced grammatical ephemeral random constants and digit concatenation.

References

1. O'Neill, M., Ryan, C. (1999). Automatic Generation of Caching Algorithms, In K. Miettinen and M.M. M*ä*kel*ä* and J. Toivanen (Eds.) Proceedings of EUROGEN99, Jyv*ä*skyl*ä*, Finland, pp. 127-134, University of Jyv*ä*skyl*ä*.
2. Dempsey, I., O'Neill, M. and Brabazon, T. (2002). Investigations into Market Index Trading Models Using Evolutionary Automatic Programming, In *Lecture Notes in Artificial Intelligence*, *2464*,Proceedings of the 13th Irish Conference in Artificial Intelligence and Cognitive Science, pp. 165-170, edited by M. O'Neill, R. Sutcliffe, C. Ryan, M. Eaton and N. Griffith, Berlin: Springer-Verlag.
3. O'Neill, M., Dempsey, I., Brabazon, A., Ryan, C. (2003). Analysis of a Digit Concatenation Approach to Constant Creation. In LNCS 2610 Proceedings of the 6th European Conference on Genetic Programming, EuroGP 2003, pp.173-182. Springer-Verlag.
4. Koza, J.R. (1992). Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press.
5. Spencer, G. (1994). Automatic Generation of Programs for Crawling and Walking. In Kenneth E. Kinnear, Jr. (Ed), Advances in Genetic Programming, Chapter 15, pp. 335-353, MIT Press.
6. Angeline, Peter J. (1996). Two Self-Adaptive Crossover Operators for Genetic Programming. In Peter J. Angeline and K. E. Kinnear, Jr. (Eds.), Advances in Genetic Programming 2, Chapter 5, pp.89-110, MIT Press.
7. Evett, Matthew and Fernandez, Thomas. (1998). Numeric Mutation Improves the Discovery of Numeric Constants in Genetic Programming, Genetic Programming 1998: Proceedings of the Third Annual Conference, University of Wisconsin, Madison, Wisconsin, USA, pp.66-71, Morgan Kaufmann.
8. Ryan, C., Keijzer, M. (2003). An Analysis of Diversity of Constants of Genetic Programming. In LNCS 2610 *Proceedings of the 6th European Conference on Genetic Programming, EuroGP 2003*, pp.404-413. Springer-Verlag.
9. Keijzer, M. (2003). Improving Symbolic Regression with Interval Arithmetic and Linear Scaling. In LNCS 2610 *Proceedings of the 6th European Conference on Genetic Programming, EuroGP 2003,* pp.70-82. Springer-Verlag.

10. Topchy, A., Punch, W.F. (2001). Faster Genetic Programming based on local gradient search of numeric leaf nodes. In *Proceedings of the Genetic and Evolutionary Computation Conference GECCO 2001*, pp.155-162, Morgan Kaufmann.
11. O'Neill, M., Ryan, C. (2003). Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language. Kluwer Academic Publishers.
12. O'Neill, M. (2001). Automatic Programming in an Arbitrary Language: Evolving Programs in Grammatical Evolution. PhD thesis, University of Limerick, 2001.
13. O'Neill, M., Ryan, C. (2001) Grammatical Evolution, *IEEE Trans. Evolutionary Computation*, 5(4):349-358, 2001.
14. Ryan C., Collins J.J., O'Neill M. (1998). Grammatical Evolution: Evolving Programs for an Arbitrary Language. *Lecture Notes in Computer Science 1391, Proceedings of the First European Workshop on Genetic Programming*, 83-95, Springer-Verlag.
15. O'Neill, M., Ryan, C., Keijzer M., Cattolico M. (2003). Crossover in Grammatical Evolution. *Genetic Programming and Evolvable Machines*, Vol. 4 No. 1. Kluwer Academic Publishers, 2003.
16. Koza, J.R. (1994). Genetic Programming II: Automatic Discovery of Reusable Programs. MIT Press.
17. Banzhaf, W., Nordin, P., Keller, R.E., Francone, F.D. (1998). Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications. Morgan Kaufmann.
18. Koza, J.R., Andre, D., Bennett III, F.H., Keane, M. (1999). Genetic Programming 3: Darwinian Invention and Problem Solving. Morgan Kaufmann.
19. Koza, J.R., Keane, M., Streeter, M.J., Mydlowec, W., Yu, J., Lanza, G. (2003). Genetic Programming IV: Routine Human-Competitive Machine Intelligence. Kluwer Academic Publishers.
20. Holland, J. (1998). *Emergence from Chaos to Order*,Oxford: Oxford University Press.
21. Nie, J. (1997). Nonlinear time-series forecasting: A fuzzy-neural approach, *Neurocomputing*, 16:63-76.
22. Castillo, E. and Gutierrez, J. (1998). Nonlinear time series modeling and prediction using functional networks. Extracting information masked by chaos, *Physics Letters A*, 244:71-84.
23. Saxen, H. (1996). On the approximation of a quadratic map by a small neural network, *Neurocomputing*, 12:313-326.
24. May, R. (1976). Simple mathematical models with very complicated dynamics, *Nature*, 261:459-467.

Memetic Crossover for Genetic Programming: Evolution Through Imitation

Brent E. Eskridge and Dean F. Hougen

University of Oklahoma, Norman OK 73019, USA
{eskridge,hougen}@ou.edu,
http://air.cs.ou.edu/

Abstract. For problems where the evaluation of an individual is the dominant factor in the total computation time of the evolutionary process, minimizing the number of evaluations becomes critical. This paper introduces a new crossover operator for genetic programming, *memetic crossover*, that reduces the number of evaluations required to find an ideal solution. Memetic crossover selects individuals and crossover points by evaluating the observed strengths and weaknesses within areas of the problem. An individual that has done poorly in some parts of the problem may then imitate an individual that did well on those same parts. This results in an intelligent search of the feature-space and, therefore, fewer evaluations.

1 Introduction

As genetic programming is being applied to problems where the evaluation times dominate the total compilation time, the feasibility of finding solutions in a reasonable time frame becomes increasingly important. This requirement becomes even more evident when genetic programming is applied to domains such as robot control (e.g., [1]) where the time to evaluate an individual's fitness many times is prohibitively expensive, as the evaluation time is often orders of magnitude larger than the remaining computation time required by genetic programming. In such cases it is crucial that the maximum amount of information be extracted from each evaluation, even at the cost of greater computation time.

For these reasons we introduce *memetic crossover*. Memetic crossover is based on imitation—people learn, not only from their own direct experiences with the world, but also by following patterns, models, or examples they have observed. This imitation is an active process by which people acquire cultural knowledge.

This new approach extracts more information from an individual than just its overall fitness. It allows the system to evaluate how portions of the individual perform in specific areas of the problem (referred to as *sub-problems*). This information allows the system to intelligently combine individuals by identifying the most promising portions of each one. As a result, fewer evaluations are required and the feasibility of finding a solution in a reasonable time is increased.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 459–470, 2004.

2 Related Work

Dawkins [2] proposed the evolution of cultural knowledge through the use of cultural replicators he termed *memes*. The concept of *memetic algorithms* was proposed by Moscato [3] in an effort to model evolutionary computation on cultural evolution instead of biological evolution. Moscato recognized that there are at least two great differences between biological evolution and cultural evolution: (1) individuals cannot choose their own genes, whereas memes can be acquired intentionally, and (2) individuals may modify and improve upon the memes that they acquire, whereas they cannot do so with their genes.

Hougen et al. [4] noted that Moscato chose to implement only the second difference and, therefore, presented a new approach called memetic *learning* algorithms that implements the first difference. In this approach, individuals are given the opportunity to learn through the imitation of other individuals. An individual will imitate when it is able to find another that was more successful than itself in some situation. In the present paper and, concurrently, elsewhere [5] we extend the concept of memetics from genetic algorithms to genetic programming. (In that other work we consider the artificial ant problem using the quite difficult Los Altos trail and the royal tree problem. Here we consider the artificial ant problem using the better-known Santa Fe trail and the bumblebee problem.)

The use of Koza-style Automatically Defined Functions (ADFs) [6,7] restricts the crossover operator to operate within the branches of the individual, namely the main execution branch and the defined functions. However, the selection of the branches or nodes on which to perform crossover is done randomly.

Work has been done on restricting crossover to nodes that reside in similar contexts within the individual [8]. Here again, emphasis is placed on the genotype of the individual, namely the locations of the nodes, not the phenotype.

Langdon [9] tracks the progress of an individual while it is being evaluated in an effort to only perform crossover on trees that have performed poorly in a particular portion of a problem. However, once a tree has been selected for crossover, node selection is performed randomly.

The automatic discovery of subroutines during the evolutionary process [10] has been investigated as a way to detect useful building blocks within an individual. This approach requires two methods of analysis. First, viable individuals are selected when they demonstrate marked improvement in fitness with respect to the fitness of their worst parent. Second, possibly useful subroutines are chosen by their relatively high execution rate within the individual. This approach differs from our approach in that only potentially successful areas are identified, not successful and failed nodes, and the fitness of the various potential subroutines is inferred from the overall fitness of the individual, not observed directly.

While multi-objective fitness [11] attempts to utilize more information learned from the evaluation of an individual than a simple fitness value, it does not delve into the individual itself in an effort to determine the fitness of its components.

3 Proposed Approach

Memetic crossover is directly inspired by Dawkins' idea of the acquisition of memes through imitation [2]. For imitation to be effective in reducing the amount of time spent learning, it must be able to discriminate between parts that lead to success and parts that lead to failure.

Memetic crossover achieves this requirement through a three-step preparatory process. First, the problem must be broken down into *sub-problems*. These sub-problems can be pre-defined, determined upon initialization, or observed during execution. Second, the evaluation order of nodes within each sub-problem is tracked during the fitness evaluation of the individual. Note that this tracking in no way affects the process of evaluating fitness of the individual. Third, for each node, the sub-problems for which it contributed to success and failure are analyzed and used to rank the nodes for each result category. In standard crossover, a bias towards selecting non-terminal nodes 90% of the time is made. We favor this option so a multiplier is applied to non-terminal nodes during the ranking process. This allows us to bias the ranking without falling back to random selection (as in standard crossover).

When a *recipient* individual is selected for memetic crossover, an effort is made to find a compatible *donor* individual. The sub-problem performance of the worst nodes of the recipient and the best nodes of the donor are compared. If the match between the sub-problems for a pair of nodes exceeds the threshold value for that generation, the individuals are considered compatible and crossover occurs with the recipient receiving the donor's "good" node. If a compatible donor is not found after a predetermined number of selections, the donor containing the best found match is used. When appropriate, the required match threshold between nodes starts low and increases with each generation. This is done to counter the relatively high failure rates in sub-problems early on that result from the fact that the initial population is generated randomly. As individuals become more successful in sub-problems, the probability of finding a compatible match between nodes increases.

It is important to note that the memetic crossover operator does not require any problem-specific information. Sub-problems are only referred to by an identifier for the matching process. Thus, no passing of problem information to the operator is required. While the addition of problem-specific information is not prohibited, this was not done in any of the experiments presented so as not to give our approach an unfair advantage.

4 Experiments

To evaluate memetic crossover, two standard problems were chosen. The artificial ant problem was chosen because it can easily be decomposed into unique sub-problems. The bumblebee problem was chosen because the sub-problems are not unique.

4.1 The Artificial Ant Problem

Background. For our first test of memetic crossover, we use the standard genetic programming benchmark problem of the artificial ant using the Santa Fe trail [6]. The goal is to evolve a program that can successfully traverse a food trail with gaps. Fitness is calculated as the amount of uneaten food that remains after traversal. It is important to note that the program may be evaluated as many times as is necessary to either eat all the food or reach the movement limit.

Configuration. First we must find the sub-problems involved in order to adequately evaluate the fitness of the individual's components. For this experiment, we define a sub-problem as navigating a gap in the trail (i.e., eating the food items on either side of the gap in succession). The trail is analyzed upon initialization and the gaps are labeled. Note that there is no preferred order of traversal for the gap. To have a preferred order, the trail would have to be analyzed by traversing it, thus negating the reason for evolving a solution.

As the individual is evaluated, the order of execution of the nodes in the evaluation is recorded. If a node is involved when the ant eats the food on either side of the gap in succession, it is labeled as contributing to the successful completion of the sub-problem. If, however, the node is involved when the ant wanders off the path at a gap and runs into the trail at another point, it is labeled as contributing to the failed completion of the sub-problem. Once evaluation of the individual is completed, nodes are ranked in two categories: best and worst. The top N (in our case, 5, see section 5) are saved for use in crossover.

The parameters used for the system are the same as those used by Langdon and Poli [12]. The baseline implementation uses 90% standard, one child crossover and 10% elite cloning. It is also important to note that instead of the 400 maximum moves used by Koza [6], a value of 600 was used. To adequately explore the effects of varying the amount of memetic crossover and the non-terminal multiplier, several values for each variable were evaluated. Memetic crossover contributions of 10% to 90% were used in conjunction with 10% cloning—the remaining percentage being standard crossover. Non-terminal multiplier values of 1 to 9 were used for each memetic crossover percentage evaluated. For each combination, including the standard implementation, 500 runs were made.

Results. For the standard crossover alone, 465,500 of individuals need to be processed to yield a solution 99% of the time (Fig. 1), which is consistent with Koza's result [6] of 450,000 individuals. When 30% of a generation is created with memetic crossover and a non-terminal multiplier of 7, only 322,000 individuals need to be processed, a 30% reduction (Fig. 2). On average for all non-terminal multipliers evaluated for a 30% memetic crossover contribution, 394,500 individuals need to be processed, a 15% reduction.

As the memetic crossover contribution is increased, however, a different picture arises. At the extreme of 90% memetic crossover and 10% cloning, only

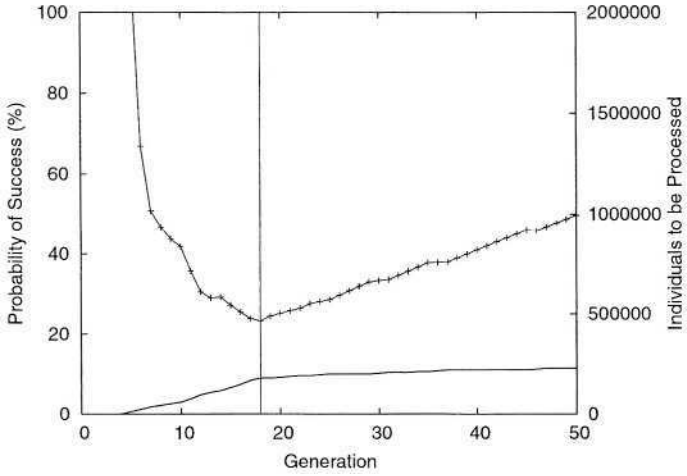


Fig. 1. Performance curves for the ant problem with no memetic crossover. The vertical line indicates E = 465,500, the minimal number of individuals to be processed to yield a solution 99% of the time. The solid plot represents the cumulative probability of success $P(M, i)$ and the marked plot represents the number of individuals that need to be processed.

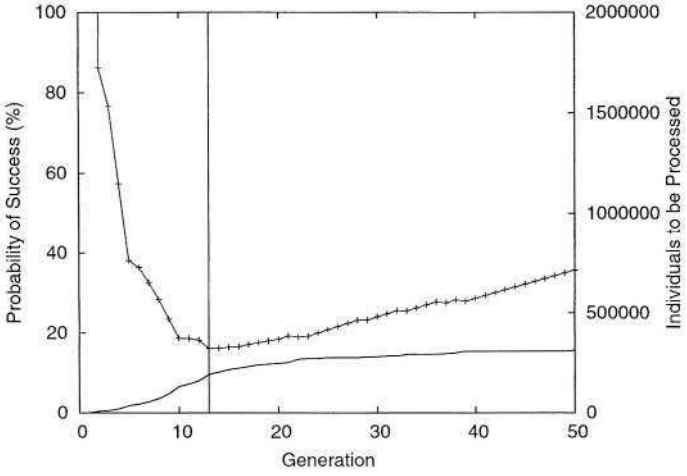


Fig. 2. Performance curves for the ant problem with 30% memetic crossover and a non-terminal multiplier of 7. (E = 322,000 individuals)

two solutions were found in 4,500 runs and those were found in the initial generation. While initially surprising, the result is logical. Without the standard crossover, there is no exploration of the search space. All the individuals are trying to imitate one another and none are attempting to learn anything new.

Of the percentages examined, a 30% memetic crossover contribution showed the best performance (Fig. 3). However, even with memetic crossover, the problem remains quite difficult as less than 20% of runs end in success.

Running Time. Because some researchers may be interested in applying memetic crossover in problem domains in which evaluation time per individual may not be the dominating factor, we also looked at the run-time cost of memetic crossover using the artificial ant problem. An additional 500 runs were performed using the standard crossover method and the memetic crossover configuration that yielded the best results. In the standard case, the total calculated run time to reach a 99% probability of finding an ideal individual was 178,966 ms, while the run time of the memetic case was 218,490 ms. While using memetic crossover for this problem increased the total run time by approximately 22%, it is important to note that this problem represents the extreme lower end of problems in which memetic crossover is applicable as the evaluation time for individuals is quite low (approximately 0.15 ms per individual). Furthermore, we should note that our code has not been optimized for run time and believe that the runtime for the memetic case would decrease should we attempt to optimized the code. In other words, in cases where the evaluation time for each individual is exceedingly small, using memetic crossover may be worse in terms of run time than standard genetic programming methods. However, we are most interested in cases in which evaluation time for each individual *does* dominate. In such a case, memetic crossover has the potential for tremendous speed-up. In the limit (as evaluation time per individual goes to infinity) a 30% reduction in evaluations, for example, would result in an overall 30% reduction in effort.

Table 1. The minimum number of individuals to be processed (E) observed in the ant problem for memetic crossover rates of 10% to 90% and non-terminal multiplier values of 1 to 9. The standard, non-memetic approach yields a result of E = 465,500. "MAX" indicates that there were no solutions found in any run and E is at the maximum value.

	1	2	3	4	5	6	7	8	9
0.1	427500	448500	351000	375000	360000	390000	389500	390000	382500
0.2	357000	348500	434000	392000	389500	429000	360000	480000	399500
0.3	384000	357000	392000	441000	470000	340000	322000	391000	450500
0.4	496000	405000	389500	528000	470000	416500	460000	448000	352000
0.5	527000	374000	563500	427500	413000	442500	498000	462000	512000
0.6	520000	773500	616000	589000	622500	800000	646000	892500	622500
0.7	978000	960500	772500	1059500	870000	700000	1120000	960500	833000
0.8	1287000	1798500	1026000	1137500	1573000	1704000	864000	1910000	1040000
0.9	MAX	1150500	MAX	MAX	MAX	MAX	MAX	1150500	MAX

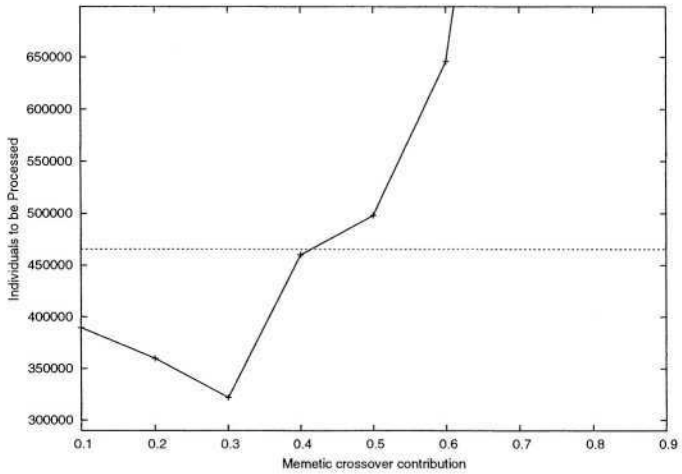


Fig. 3. The minimum number of individuals to be processed for varying percentages of memetic crossover. The dashed line indicates the number of individuals to be processed when no memetic crossover is used. (E = 465,500 individuals)

4.2 The Bumblebee Problem

Background. The second test uses the well-known bumblebee problem [7]. For this problem, the goal is to evolve a program that navigates a bee to all the flowers located in a plane. This problem was specifically designed to be scalable in difficulty. The more flowers that exist in the plane, the more difficult the problem becomes. Since the individual will only be executed once during evaluation, it will be large and full, with its size increasing as the difficulty of the problem increases. For the flower counts used in this experiment, the bumblebee problem is much easier to solve than the artificial ant problem.

Configuration. Since the locations of the flowers for each fitness case are determined randomly upon startup, individual sub-problems—navigating from one flower to another—will have to be observed during execution and not determined beforehand. Furthermore, there is no preferred order of traversal for the flowers as the next flower to visit is generated randomly each time. Since enumeration of all the possibilities would be prohibitive as the problem scales, we only consider the total number of successful and failed sub-problems for each node.

For this problem, we define successful completion of a sub-problem as navigating to the next flower using the minimum number of moves. Since all movement is orthogonal in the plane, the minimum number of moves in the general case is two. Each node's execution is tracked as the individual solves a sub-problem. Once the individual has been evaluated, the top N nodes in each category are saved. The memetic crossover implementation used for this problem is similar to that of the ant problem with one exception—the specific sub-problems of the

donor and recipient nodes are not compared with one another since sub-problems are not unique. We predict that this will hinder the effectiveness of the memetic approach as we are less able to gauge effectively the match between nodes.

The parameters used for the system are the same as described by Koza [7]. The baseline implementation uses 90% standard crossover and 10% elite cloning. Again, several values for the memetic crossover contribution and non-terminal multiplier were used. Values ranging from 1 to 5 were used for the non-terminal multiplier because they seemed to provide the most interesting insight. A value of 9 was also used since standard crossover is biased and selects non-terminals 90% of the time. A total of 60 runs were made for each combination of variables.

Results. As was predicted, the benefit of memetic crossover was hampered by the the operator's inability to to create an effective match between the crossover nodes. While the purely standard crossover approach on a 10-flower problem yielded a result of 80,000 individuals requiring processing, the best memetic crossover approach yielded 68,000. This is certainly an improvement over the standard method, but not nearly as dramatic as the previous results.

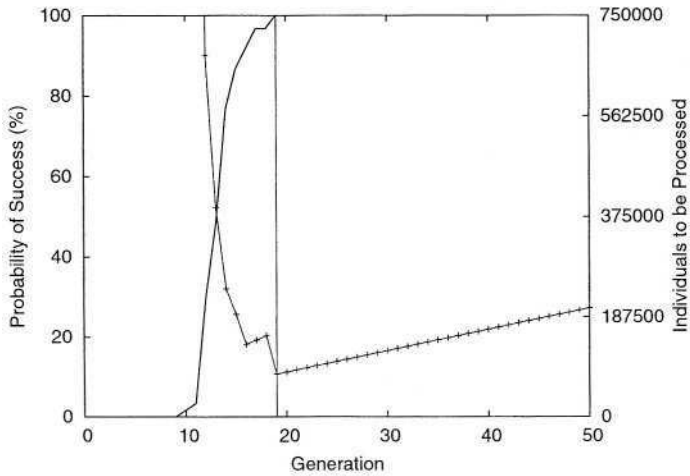


Fig. 4. Performance curves for the bumblebee problem with 10 flowers using standard crossover. (E = 80,000 individuals)

For the 15-flower version of the problem, the addition of memetic crossover yielded a much better result, 128,000 individuals to be processed as opposed to the 176,000 individuals of the purely standard method. However, the benefit only appeared in standard and memetic crossover combinations with relatively low contributions from memetic crossover. In both versions of this problem, the negative impact of a purely memetic approach was not as apparent as in the ant

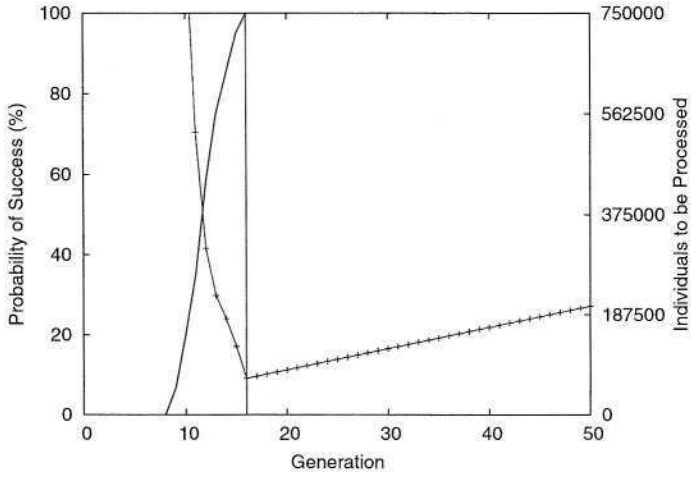


Fig. 5. Performance curves for the bumblebee problem with 10 flowers using 30% memetic crossover and a non-terminal multiplier of 3. (E = 68,000 individuals)

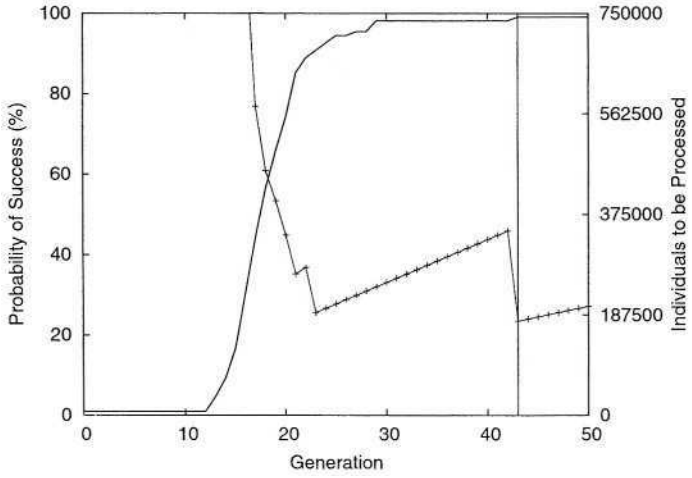


Fig. 6. Performance curves for the bumblebee problem with 15 flowers using standard crossover. (E = 176,000 individuals)

problem. This is most likely a result of the selection of crossover nodes being far more lenient and choosing a broader range and context of nodes.

5 Discussion

For both problems, the addition of the memetic crossover reduced the the number of individuals that are required for processing. The nature of the ant problem,

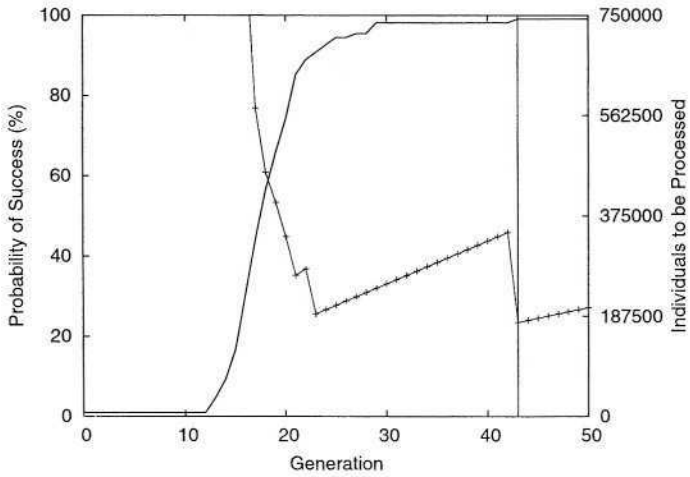


Fig. 7. Performance curves for the bumblebee problem with 15 flowers using 10% memetic crossover and a non-terminal multiplier of 5. (E = 128,000 individuals)

namely its easy decomposition into reasonable sub-problems, allowed it to exploit memetic crossover to a greater degree than the bumblebee problem. In the ant problem, memetic crossover was able to effectively match a failure-prone node to a success-prone node based on the related sub-problems. Thus, individuals were able to imitate the actions of another within a particular set of sub-problems. Since the sub-problems for the bumblebee problem were far less specific than for the ant problem, memetic crossover's impact was limited. Since there was no particular context associated the success or failure of a sub-problem, the imitation exhibited by individuals was far less intelligent. This disparity is not readily evident in the results as the improvement in the bumblebee problem is almost as dramatic as that in the ant problem. We believe that this is because each sub-problem requires effectively the same solution, minimizing the lack of context for sub-problems. Furthermore, for the complexities used, the bumblebee problem is much easier to solve with genetic programming than the ant problem.

The choice of retaining N = 5 nodes in each category for use in crossover was made to allow for multiple opportunities to find a effective match while not requiring the system to retain the full list of nodes. For the bumblebee problem, a value of N = 1 could have been used without affecting the results since no context was associated with the node's fitness.

6 Conclusions

We have introduced a new genetic programming operator that allows for an intelligent search of feature-space. Memetic crossover's ability to exploit an individual's successful experiences in a particular sub-problem and allow others the

opportunity to imitate the actions that led to that success has a dramatic impact on the time required to find a solution. Its purpose is not to replace the standard crossover operator but to enhance it. The standard crossover operator explores the feature-space in search of better solutions, but has the undesired effect of being destructive much of the time. Memetic crossover allows the population as a whole to better benefit from the knowledge gained through this exploration by propagating these partial solutions to others in context, which reduces the destructiveness associated with crossover. For the memetic approach to reach its potential, the problem being attempted must lend itself to the organized decomposition into sub-problems, for they provide the context in which the crossover operates. Without context, much of the intelligence of the approach is lost.

As was seen, memetic crossover does incur additional processing costs. However, when these costs are considered in the anticipated problem domain where the evaluation time for individuals is orders of magnitude larger than those encountered in the current work, these costs are negligible when compared to the time saved through the reduction in the number of evaluations.

7 Future Work

This paper should be viewed as an introduction to the memetic crossover approach. A number of areas are available as future work. The role of the non-terminal multiplier used in sorting successful and failed nodes is not well understood. Further investigation into exactly how this affects the overall learning in the system is warranted. Currently, selection of the donor uses tournament selection, which is random. If this selection could be made more intelligently, based on its successes with matching sub-problems, the resulting crossover may be more productive. It would also be interesting to apply this approach to problems with strongly typed variables [13]. This would allow node matching to be more accurate and allow for better replacement of poorly performing areas.

We wish to apply memetic crossover to problems, such as robot learning, in which evolutionary approaches are generally considered "off-line" methods due to their need for many expensive learning experiences [14].

Acknowledgments. This material is based upon work supported by the U. S. Army Research Laboratory and the U. S. Army Research Office under grant number DAAD19-03-1-0142. All work performed in this paper was done with the use of Sean Luke's ECJ package [15].

References

1. Nordin, P., Banzhaf, W.: Real time control of a Khepera robot using genetic programming. Cybernetics and Control **26** (1997) 533–561
2. Dawkins, R.: The Selfish Gene. Oxford University Press (1976)

3. Moscato, P.: On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical Report C3P 826, California Institute of Technology, Pasadena, CA (1989)
4. Hougen, D.F., Carmer, J., Woehrer, M.: Memetic learning: A novel learning method for multi-robot systems. In: International Workshop on Multi-Robot Systems. (2003) Available at http://www.cs.ou.edu/~hougen/mrs2003.pdf.
5. Eskridge, B.E., Hougen, D.F.: Imitating success: A memetic crossover operator for genetic programming. To appear in: Proceedings of the Congress on Evolutionary Computation, IEEE (2004)
6. Koza, J.R.: Genetic programming: On the programming of computers by natural selection. MIT Press, Cambridge, Mass. (1992)
7. Koza, J.R.: Genetic Programming II: Automatic Discovery of Reusable Programms. MIT Press, Cambridge, MA (1994)
8. D'haeseleer, P.: Context preserving crossover in genetic programming. In: Proceedings of the 1994 IEEE World Congress on Computational Intelligence. Volume 1., Orlando, Florida, USA, IEEE Press (1994) 256–261
9. Langdon, B.: Genetic Programming and Data Structures. PhD thesis, University College, London (1996)
10. Rosca, J.P., Ballard, D.H.: Discovery of subroutines in genetic programming. In Angeline, P.J., Kinnear, Jr., K.E., eds.: Advances in Genetic Programming 2. MIT Press, Cambridge, MA, USA (1996) 177–202
11. Coello, C.A.C.: A comprehensive survey of evolutionary-based multiobjective optimization techniques. Knowledge and Information Systems **1** (1999) 129–156
12. Langdon, W.B., Poli, R.: Fitness causes bloat. In Chawdhry, P.K., Roy, R., Pan, R.K., eds.: Second On-line World Conference on Soft Computing in Engineering Design and Manufacturing, Springer-Verlag London (1997) 13–22
13. Montana, D.J.: Strongly typed genetic programming. Technical Report #7866, 10 Moulton Street, Cambridge, MA 02138, USA (1993)
14. Moriarty, D.E., Schultz, A.C., Grefenstette, J.J.: Evolutionary algorithms for reinforcement learning. Journal of Artificial Intelligence Research **11** (1999) 241–276
15. Luke, S.: (ECJ 10: A Java-based evolutionary compuation and genetic programming research system) Avaliable at http://cs.gmu.edu/~eclab/projects/ecj/.

Virtual Ramping of Genetic Programming Populations

Thomas Fernandez

Florida Atlantic University, Department of Computer Science and Engineering,
777 Glades Road, Boca Raton, Florida, 33431-0991
tom@cse.fau.edu
www.cse.fau.edu/~thomas

Abstract. Genetic Programming often uses excessive computational resources because the *population size* and the *maximum number of generations per run* are not optimized. We have developed a technique called Virtual Ramping which eliminates the need to find an optimal value for these parameters and allows GP to be applied effectively to more difficult problems. Virtual Ramping continuously increases the size of a virtual population and the number of generations without requiring additional system memory. It also allows sub-populations to evolve independently which reduces premature convergence on suboptimal solutions. Empirical experiments with problems of varying difficulty, confirm these benefits of Virtual Ramping.

1 Introduction

Genetic Programming(GP) is expected to be computationally intensive. Unfortunately it often uses far more resources than necessary because two parameters, *population size* and *maximum generations per run*, are rarely optimized. These two parameters are perhaps the most important [1] because they set the space and time requirements for a GP run. It is possible to determine the optimal values for these parameters [2], but this is rarely done with real world problems because the determination of these optimal values is done empirically and is too computationally intensive. Research in Genetic Algorithms has shown that optimal population sizes will vary with the problem complexity [3] and it is reasonable to expect that optimal population sizes in GP will also depend on the problem being solved.

In previous experiments the difficulty in determining optimal values has forced us to just guess at the appropriate *population size* and *maximum generations per run*. Sometimes the selection of these parameters has been based on values used by other researchers with different problems. At other times, we would set the *population size* to some large value based on the amount of memory available. Our experience has shown us that regardless of the theoretical optimal population size, if it is set too large then virtual memory is required and this will dramatically slow down GP. Research has shown that in general larger populations are better [4], but computational effort was not considered in this

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 471–482, 2004.

conclusion, and it did not address whether better results could be achieved for the same effort by using smaller populations with more runs or more generations per run.

In our previous work we also had no methodology for setting the *maximum number of generations per run*, so we would set it to a large value, based on the limit of the amount of computer time that we had available to expend on a problem.

In his first book on GP Koza explains that more runs with fewer generations per run can be more efficient but again the computational effort involved in determining the optimal values is prohibitive for real world problems [2]. Only with toy problems is it it feasible to solve the problem hundreds of times to determine optimal parameters values.

To address the GP problem of finding optimal *population sizes* and *maximum numbers of generations per run* we have developed a technique called Virtual Ramping (VR). VR eliminates the need to predetermine these parameters by continuously increasing a virtual population size and the number of generations until GP has sufficient space and time resources to solve a given problem. We refer to our technique of dynamically increasing a GP population size as Virtual Ramping because it does not require an increase in system memory or the use of virtual memory as the virtual population size grows. The virtual population is represented by an actual population of individuals which is fixed in size.

VR also allows parts of the virtual population to evolve independently of each other. Researchers who work with endangered species understand that if the diversity in a population drops below a certain level, the populations viability can be restored by introducing new individuals into the population[5]. This benefit has been reported when sub-populations are maintained separately during parallel implementations of GP[6]. Maintaining separate sub-populations prevents an individual that is a local maxima, from saturating the entire population with it's genetic material and impeding the search for better solutions. Other research shows that separate sub-populations in parallel GP systems reduces code bloat [7].

In this research we experimentally determine that GP with VR evolves better solutions than GP without VR when both are given the same amount of computational resources. By better solutions we mean that the models evolved using VR have a higher accuracy when classifying observations outside of the training data.

2 The Test System

The Genetic Programming System we have written for these experiments is called NUGP. It is a classical Koza type GP system designed as an experimental test bed for new GP techniques. It has an object oriented design where the GP individuals are based on the *Strategy* pattern [8]. Its most important class is a singleton called *Experiment*. NUGP includes a Statistical Analyzer that

compares test scores to see if there are significant statistical differences between sets of GP runs [9].

3 Fair Allocation of Computational Resources

In order to allow a fair comparison of different GP systems we need to be certain that each is given the same amount of computational resources. Often computational effort in GP is measured in generations, but this may not be accurate. Generations usually take increasing amounts of time as GP progresses. This is illustrated by the graph in Figure 1 which shows the average time per generation in one of our typical GP runs (without VR) over a 5 minute period. The latter generations are taking 6 to 7 times as long as the first generation.

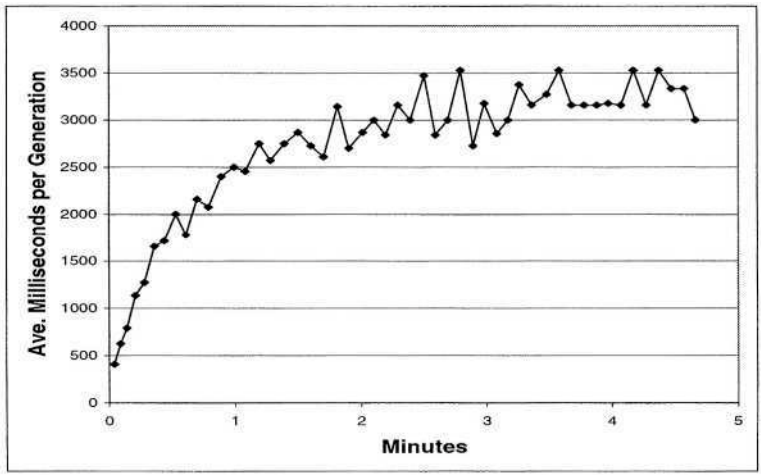


Fig. 1. Average milliseconds per generation as GP runs progress in time

This increase is apparently caused by code bloat[10] as illustrated in the graph in Figure 2, which shows that the increase in *time per generation* corresponds closely to the increase in the average tree size (with a coefficient of linear correlation equal to 0.97).

At best counting the number of generations per run is an uneven measure of computational effort in GP and at worst it is an unfair one. Since GP with VR tends to generate smaller S-expression trees it uses less computational resources than GP without VR when they are run for the same number of generations.[1]

[1] Strictly speaking NUGP does not have generations. It runs asynchronously. We do however still have an analogous term *Round* which is the time during which an asynchronous system will allow n individuals to mate where n is the size of the population. In this paper we will refer to a *Round* as a Generation.

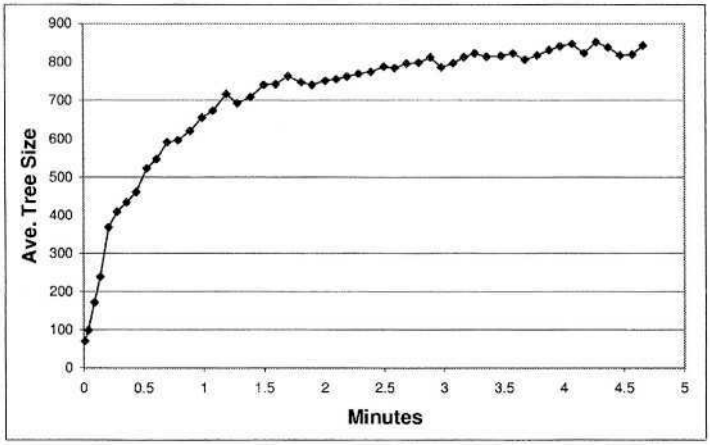


Fig. 2. Average S-expression tree size as GP runs progress in time

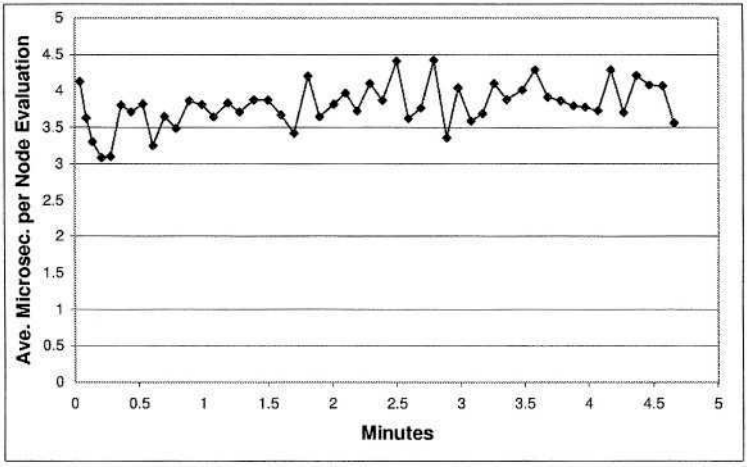


Fig. 3. Average microseconds per S-expression node evaluation as GP runs progress in time

Execution time has been used as a measure of GP system resources [11]. This might work on some platforms but NUGP runs on multiple machines with different configurations that do not have a real time operating system. One of our goals for NUGP was to be able to perform GP runs on different systems and still be able to compare the performance objectively.

We have chosen the *the number of S-expressions nodes evaluated* as our measure of computational effort. We have chosen it because it is independent of hardware performance and software implementation. In the graph in Figure 3 we see that the average number of microseconds per node evaluation is rea-

sonably consistent throughout a GP run. This confirms that the *number of S-expressions evaluated* is a good unit of measure for allocating resources in GP research. Since this number is fairly large we use the term MNE to stand for Million Nodes Evaluated.

4 Methodology

To understand Virtual Ramping we will first consider another hypothetical technique, which we will refer to simply as Ramping. Ramping cannot actually be implemented because its memory requirements grow exponentially. We will then examine Virtual Ramping and see that it provides the benefits of Ramping while maintaining a constant memory requirement. The flow chart for the Ramping process is shown in Figure 4. Ramping would be implemented as follows. A variable s is set to the initial population size. Typically it would be started at a small value such as ten. A variable g is set to an initial number of generations. This would also be a small value such as one. A GP population referred to as the *saved population* or P_{saved} would be generated with s individuals and run for g generations.

Now the main loop of the Ramping algorithm will begin. Another population called P is generated with a population size of s and allowed to run for g generations. Then population P is merged into population P_{saved}, which will double it in size. Population P_{saved} is allowed to run for g more generations. At this time the variables s and g will both be doubled. Then the best individual in P_{saved} is examined to see if it is a satisfactory solution and in this case process is done and has been successful. If the problem is not solved and memory or time resources are not exhausted then the loop will continue to run.

Since the values of s and g are doubled, each time through the loop, the population P_{saved} will continually double in size and will keep running for twice as many generations. In this way Ramping will continuously double the memory and time resources used, until a solution is found or the resources are exhausted.

The advantage of Ramping is clear. The GP practitioner does not have to determine or guess the required *population size* and the *maximum number of generations per run* needed to solve a problem. Less difficult problems will be solved quickly with small populations running for fewer generations. Harder problems will be solved (hopefully) as the population size and the number of generations are increased sufficiently. The disadvantage of Ramping is also clear. The constant doubling of the population size will quickly exhaust all available computational resources.

How can we modify this technique so that the memory requirements do not grow exponentially? The answer is a modified version of Ramping, which we call Virtual Ramping or VR. At each point where Ramping would double the population size, VR also discards half the population using a tournament selection to ensure that the less fit individuals are more likely to be discarded.

In this way VR maintains a constant memory requirement. Since a tournament selection is used when cutting the population in half, the surviving half

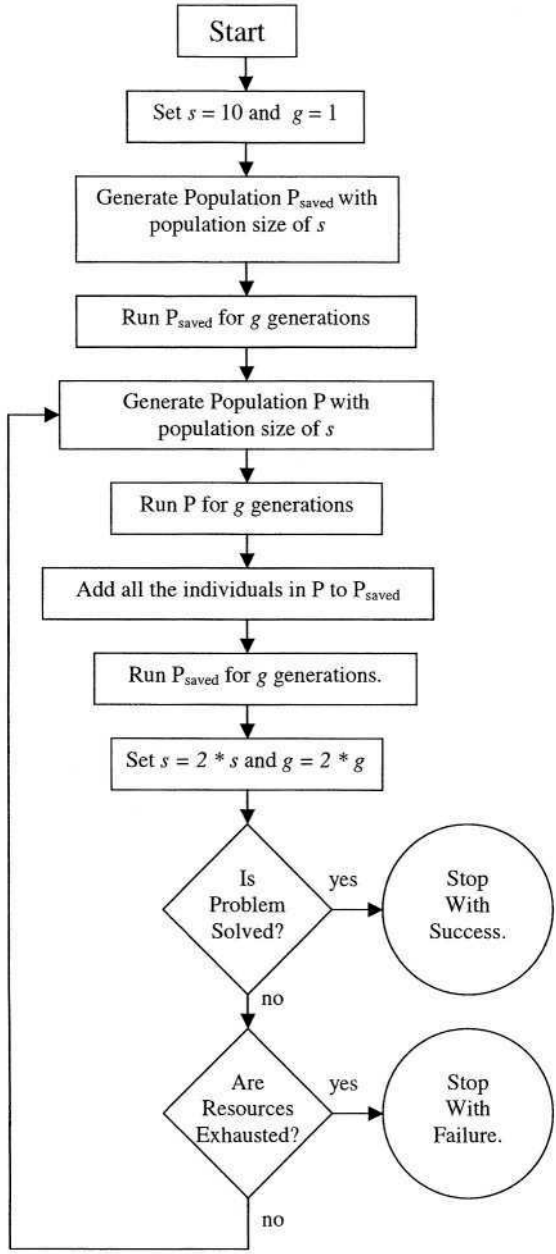


Fig. 4. Flow Chart for Ramping

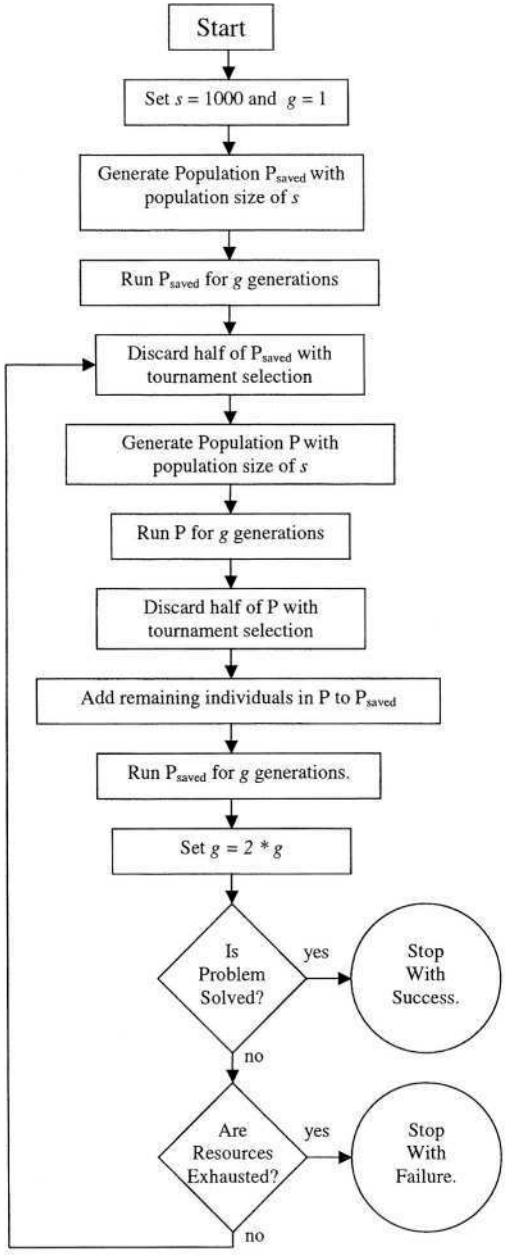


Fig. 5. Flow Chart for Virtual Ramping

should retain the majority of the population's "good" genetic material and thus keep most of the benefit of Ramping without the exponential increase in memory requirements.

The flow chart for VR, Figure 5 shows the details of how this works. It is essentially the same as Ramping with a few minor but important changes. At the end of the loop the value of g is doubled but the value of s does not change. Before the population P is merged into P_{saved} half of the individuals are discarded from both populations using a tournament selection in an effort to keep the best individuals. These reductions ensure that the size of population P_{saved} will never exceed the value of s. Since the value of s remains constant it is started out at a much larger value such as 1000.

5 The Experiments

We now ask, does VR benefit the GP process? It eliminates the need to determine the *population size* and the *maximum number of generations per run* but does it generate solutions that make more accurate classification of test observations that were not used during training? To answer this question we have conducted some experiments. VR has been tested using three problems of different difficulty which we refer to as *Easy, Medium,* and *Hard.* All of the problems are symbolic regression problems with a set of independent variables and a dependant variable, which was derived from a predetermined formula. For each problem there were three sets of 30 observations. There was a training set, which was used by GP for training, and there were two tests sets referred to as *Test1* and *Test2.* GP was trained using the training set and a fitness function described in the following formula:

$$fitness = w_e \left(\frac{\sum_{i=1}^{n} (2^{E_i} - 1)}{n} \right) . \tag{1}$$

E_i is the prediction error for observation i defined as

$$E_i = |y_i - \hat{y}_i| . \tag{2}$$

Where y_i and $\hat{y}_i$ are the value of dependent variable value and the prediction of the dependent variable for observation i. In this case n is 30 because there are 30 observations in the training data set.

The use of 2^{E_i} in formula (1) makes it an exponential fitness function, which is a means of directing GP to concentrate on the largest errors. The reason for subtracting one from the exponential error term is so that zero will signify no error.

After using the exponential fitness function to drive the GP process the best individuals from each GP run are tested using the two sets of test observations. The evaluation function for the test sets consists of the percentage of the observations, which are misclassified in one of the three categories: *Less than zero,*

Close to zero, and *Greater than zero.*[2] An observation will be considered correctly classified if both the dependant variable and the GP model prediction are both greater than zero, both less than zero, or both within a given threshold of zero, i.e., both close to zero.

For the *Medium* and *Hard* problems the threshold for being close to zero was set to be between –2 and 2. The *Easy* problem was so easy that at first GP always evolved models that were 100% accurate at classification with or without VR. The ability to do this is a credit to GP, but also makes the *Easy* problem useless for the purpose of evaluating the benefit of VR. In order to make the *Easy* problem more difficult the threshold for being close to zero was set to being exactly zero. Also where GP was applied to the *Easy* problem the actual population size was reduced from 1000 to 100. After making these changes the *Easy* problem was tackled with and without VR and we were then able to determine that the system with VR produced better results.

For each of the three problems an experiment was run with one hundred GP runs with VR and another one hundred runs without. During each generation 98% of the new individuals were created through reproduction. Individuals were selected for mating with a tournament size of two. Mutation was applied to 2% of the individuals. Individuals were randomly selected for mutation with a uniform distribution. S-expression trees were limited to 1024 nodes. Initial populations were generated with sparse and bushy trees with depths between 3 and 6 levels. In each generation Numeric Mutation was applied to 10% of the numeric constants in 10% of the population. Numeric Mutation replaces the constants with a randomly generated value using a normal distribution around the previous value and a standard deviation of an individual's fitness score divided by ten[12]. All of these parameters were held constant for all runs. All runs in the experiment with the *Easy* problem were allowed to continue until 100 million S-expression nodes had been evaluated. In the experiment with the *Medium* and *Hard*problems, runs were executed until one billion S-expression nodes were evaluated.

The GP function set includes addition, subtraction, multiplication, protected division, and a function called IFGTZ (if greater than zero). IFGTZ has three arguments. If the first argument is greater than zero then it will return the second argument, otherwise it returns the third argument. The terminal set included the independent variables and the random ephemeral constant.

In each experiment the only difference between the two sets of runs was that VR was used with one set and not used in the other.

The training and the two test data sets for each of the three problems were synthetically generated. For the *Easy* problem each observation consisted of two independent variables, which were selected at random between –2 and 2 with a uniform distribution. We will refer to these two values as x_1 and x_2. The dependent value y was calculated using the formula

$$y = (3.456x_1) + (6.543/x_2)\ . \tag{3}$$

[2] all of the data has been normalized to have an mean of zero.

For the *Medium* problem each observation consisted of five independent variables, which were also selected between –2 and 2 with a uniform distribution. The independent variables are referred to as x_1,x_2,x_3,x_4 and x_5. The dependent value y was calculated for each observation as

$$y = (x_1 x_2) + ((3x_3^2 + 5x_4)/x_5) \; . \tag{4}$$

The *Hard* problem was generated in the same way as the *Medium* problem with two important exceptions. After calculating the dependent variable, each of the independent variables values was modified by replacing it with a randomly generated value selected between plus or minus 10% of its value using a uniform distribution. This random component is a simulated measurement error added to make the problem harder for GP and more like a real world problem. Also five additional independent variables were added, also randomly generated like the original five independent variables, but these were not used to generate the dependent variable values. There were added to challenge GP's ability to do model selection and determine which independent variables should be used when building models.

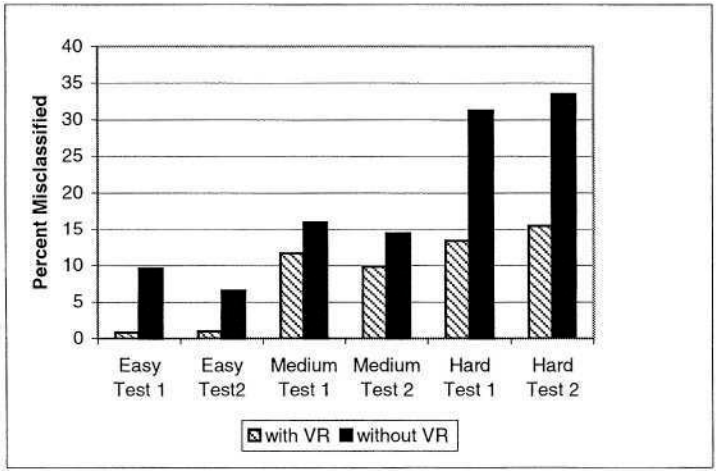


Fig. 6. Average percent misclassified in test data sets with and without Virtual Ramping

6 The Experimental Results

For all three problems 100 GP runs were made with VR and another 100 GP runs were made without VR. As shown in Figure 6, for all three cases the average percent misclassified (in both test sets) was smaller with VR.

To understand the significance of this we must also consider the sample variances and calculate the z statistic. This data is summarized in Table 1. In all

Table 1. Statistics for the three problems with and without Virtual Ramping. (All samples consist of 100 runs.)

	Easy Problem		Med. Problem		Hard Problem	
With VR	Test 1	Test 2	Test 1	Test 2	Test 1	Test 2
Ave. Misclassified	0.00833	0.00967	0.11667	0.09867	0.13367	0.15467
Variance	0.00142	0.00108	0.00946	0.00666	0.01357	0.01442
Without VR						
Ave. Misclassified	0.09567	0.06567	0.15967	0.14367	0.31233	0.33500
Variance	0.01429	0.00715	0.00676	0.00617	0.02875	0.02690
z statistic	-6.9670	-6.1720	-3.3759	-3.9727	-8.6855	-8.8713

Table 2. Statistics for the three problems with and without Virtual Ramping in the repeated experiments. (All samples consist of 100 runs.)

	Easy Problem		Med. Problem		Hard Problem	
With VR	Test 1	Test 2	Test 1	Test 2	Test 1	Test 2
Ave. Misclassified	0.00933	0.01067	0.11367	0.09567	0.11700	0.14867
Variance	0.00103	0.00065	0.00657	0.00432	0.01011	0.01435
Without VR						
Ave. Misclassified	0.08933	0.06600	0.16433	0.13133	0.30167	0.31733
Variance	0.01066	0.00563	0.00600	0.00615	0.02770	0.02832
z statistic	-7.3980	-6.9814	-4.5177	-3.4856	-9.4957	-8.1653

cases the absolute value of the z statistic is greater than 2.576 (required for 99% confidence with a two tailed test). We conclude (with a 99% certainty)[13] that for all three problems VR reduces misclassification in test data sets. All three experiments were repeated with similar results shown in Table 2.

7 Conclusions

We have run experiments with three different problems, which we have named *Easy, Medium,* and *Hard.* In each experiment we have concluded, with a 99% confidence, that Virtual Ramping improves GP's ability to classify observations outside of the training set. We found that this observation was true using two separate sets of test data for each of the three problems. To further reinforce our conclusion, the entire experiment was repeated with similar results.

The technique of Virtual Ramping could be applied to other forms of Evolutionary Computation. We suspect that it will provide similar benefit but experiments are needed to confirm this.

References

1. R. Feldt and P. Nordin. Using factorial experiments to evaluate the effect of genetic programming parameters. In Riccardo Poli, Wolfgang Banzhaf, William B. Langdon, Julian F. Miller, Peter Nordin, and Terence C. Fogarty, editors, *Genetic Programming, Proceedings of EuroGP 2000, Edinburgh, UK, April 15-16, 2000,* volume 1802 of *LNCS,* pages 271–282. Springer-Verlag, 2000.
2. J. Koza. *Genetic programming: on the programming of computers by means of natural selection.* MIT Press, 1992.
3. J.T. Alander. On optimal population size of genetic algorithms. In *CompEuro 1992: Computer Systems and Software Engineering Proceedings,* pages 65–70, 4–8 May 1992.
4. K. M. Dill, J. H. Herzog, and M. A. Perkowski. Genetic programming and its applications to the synthesis of digital logic. In *Communications, Computers and Signal Processing, PACRIM 1997,* volume 2, pages 823–826, Victoria, BC, Canada, 20-22 August 1997.
5. K. Nowell and P. Jackson. *Wild Cats: Status Survey and Conservation Action Plan.* IUCN Publications Services, 1996.
6. Makoto Iwashita and Hitoshi Iba. Island model GP with immigrants aging and depth-dependent crossover. In David B. Fogel, Mohamed A. El-Sharkawi, Xin Yao, Garry Greenwood, Hitoshi Iba, Paul Marrow, and Mark Shackleton, editors, *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002,* pages 267–272. IEEE Press, 2002.
7. G. Galeano, F. Fernandez, M. Tomassini, and L. Vanneschi. Studying the influence of synchronous and asynchronous parallel GP on programs' length evolution. In *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002.*
8. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995.
9. R. Jain. *The Art of Computer System Performance Analysis.* Wiley, 1991.
10. W. B. Langdon. The evolution of size in variable length representations. In *1998 IEEE International Conference on Evolutionary Computation,* pages 633–638, Anchorage, Alaska, USA, 5-9 May 1998. IEEE Press.
11. N. Tokui and H. Iba. Empirical and statistical analysis of genetic programming with linear genome. In *System, Man and Cybernetics: IEEE International Conference Proceedings, Tokyo, Japan, 1999,* pages 610–615 vol.3. IEEE Press, 1999.
12. T. Fernandez and M. Evett. Numeric mutation as an improvement to symbolic regression in genetic programming. In *Evolutionary Programming VII: Proceedings of the Seventh Annual Conference on Evolutionary Programming, San Diego, California, USA, March 25-27, 1998,* 1998.
13. W. Mendenhall and L. Ott. *Understanding Statistics.* Duxbury Press, 1980.

Evolving Local Search Heuristics for SAT Using Genetic Programming

Alex S. Fukunaga

Computer Science Department, University of California, Los Angeles, CA 90024, USA
fukunaga@cs.ucla.edu

Abstract. Satisfiability testing (SAT) is a very active area of research today, with numerous real-world applications. We describe CLASS2.0, a genetic programming system for semi-automatically designing SAT local search heuristics. An empirical comparison shows that that the heuristics generated by our GP system outperform the state of the art human-designed local search algorithms, as well as previously proposed evolutionary approaches, with respect to both runtime as well as search efficiency (number of variable flips to solve a problem).

1 Introduction

Satisfiability testing (SAT) is a classical NP-complete, decision problem. Let V be a set of boolean variables. Given a *well-formed formula F* consisting of positive and negative *literals* of the variables, logical connectives $\vee$, $\wedge$, For example, the formula $(a \vee b \vee \neg c) \wedge (\neg a \vee c) \wedge (\neg a \vee \neg b \vee c)$ is *satisfiable,* because if $a = true, b = false, c = true$, then the formula evaluates to true. On the other hand, the formula $(a \vee b \vee c) \wedge (\neg a \vee \neg b) \wedge (\neg b \vee \neg c) \wedge (\neg a \vee \neg c)$ is *unsatisfiable* because there is no assignment of boolean values to a,b,c such that the formula evaluates to true. SAT has recently become a very active field of research because many interesting real world problems (e.g., processor verification, scheduling) can be reformulated as SAT instances, spurring the development of fast SAT solvers. Local search procedures for satisfiability testing (SAT) have been widely studied since the introduction of GSAT [13], and it has been shown that for many problem classes, incomplete local search procedures can quickly find solutions (satisfying assignments) to satisfiable CNF formula.

The most widely used SAT local search algorithms can be succinctly described as the template of Figure 1, where the "key detail" is the choice of *variable selection heuristic* in the inner loop. Much research in the past decade has focused on designing a better variable selection heuristic, and as a result, local search heuristics have improved dramatically since the original GSAT algorithm. Some of the most successful local search algorithms include: GSAT with Random Walk [11], Walksat [12], Novelty/R-Novelty [7], and Novelty+/R-Novelty+ [5].

Evolutionary computation has also been applied to SAT. A thorough review of previous applications of evolutionary algorithms to SAT can be found in [3].

The **CLASS** (**C**omposite **L**earned **A**lgorithms for **S**AT **S**earch) system, proposed in [1] was developed in order to automatically discover variable selection

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 483–494, 2004.

```
T:= randomly generated truth assignment
For j:= 1 to cutoff
 If T satisfies formula then return T
 V:= Choose a variable using some
  variable selection heuristic
 T':=T with value of V reversed
Return failure (no satisfying
  assignment found)
```

Fig. 1. SAT local search algorithm template

heuristics for SAT local search. It was noted in [1] that many of the best-known SAT heuristics were expressible as decision-tree like combinations of a set of primitives, and thus, it should be possible for a machine learning system to automatically discover new, efficient variable selection heuristics by exploring the space of combinations of these primitives.

Whereas previous applications of evolutionary computation to SAT has focused on the design of evolutionary algorithms that searched the space of candidate solutions to the SAT problem directly, CLASS applies EC at the meta-level by generating heuristics which are embedded in a standard local search framework. It was shown that CLASS was capable of generating heuristics that were competitive with the best known SAT local search heuristics, with respect to the number of variable flips required to solve benchmark problems.

A major shortcoming of the original CLASS system was that although the automatically generated heuristics reduced the number of flips to find satisfying assignments, the actual runtime required to solve the problems was much slower (up to an order of magnitude) than the best available implementations of the standard algorithms. This was largely due to the increased complexity of the variable selection heuristic. It seemed that CLASS was making an unfavorable tradeoff between search efficiency, and the time spent on each decision. Although search efficiency is a key criterion for evaluating a SAT algorithm, in the end, what matter for practical applications (e.g., SAT solvers embedded in circuit verification tools) is runtime. Thus, from [1] it was not possible to conclude that CLASS had generated heuristics that were *truly better* than than state of the art algorithms. Furthermore, CLH-1, the best heuristic generated by CLASS, was generated by using a population pre-seeded with hand-coded heuristics, and used very sophisticated primitives which by themselves were as complex as the Novelty and R-Novelty heuristics. Although [1] reported two other heuristics (CH-1 and CH-2) which did not rely on this sophisticated "seed library" feature, it remained an open question whether a learning system using only simpler primitives could outperform CLH-1.

This paper describes CLASS2.0, which addresses the open issues described above which could not be resolved with the first version, and provides empirical evidence that genetic programming can be used to generate heuristics that outperform the state of the art SAT local search heuristics with respect to both runtime and the number of steps executed. In section 2, we give an overview

of CLASS2.0, highlighting the improvements made compared to the first version. We then present new empirical results, comparing the heuristics learned by CLASS2.0 to state of the art local search algorithms, previously studied evolutionary approaches to SAT, and heuristics generated by CLASS1.0.

2 The CLASS2.0 System

2.1 Key Concepts and Definitions

We introduce some terminology to facilitate the discussion of the common structural elements of Walksat-family SAT variable selection heuristics throughout this paper.

Definition 1 (Positive/Negative/Net Gain) *Given a candidate variable assignment T for a CNF formula F, let B_0 be the total number of clauses that are currently unsatisfied in F. Let T' be the state of F if variable V is flipped.*

Let B_1 be the total number of clauses which would be unsatisfied in T'. The net gain *of V is $B_1 - B_0$. The* negative gain *of V is the number of clauses which are currently satisfied in T, but will become unsatisfied in T' if V is flipped. The positive gain of V is the number of clauses which are currently unsatisfied in T, but will become satisfied in T' if V is flipped.*

Definition 2 (Variable Age) *The* age *of a variable is the number of flips since it was last flipped.*

The best-known SAT variable selection heuristics can be succinctly described in terms of the above definitions. For example:
GSAT [13]: Select variable with highest net gain.
Walksat [12]: Pick random broken clause *BC* from *F*. If any variable in *BC* has a negative gain of 0, then randomly select one of these to flip. Otherwise, with probability p, select a random variable from *BC* to flip, and with probability $(1 - p)$, select the variable in *BC* with minimal negative gain (breaking ties randomly).
Novelty [7]: Pick random unsatisfied clause *BC*. Select the variable v in *BC* with maximal net gain, unless v has the minimal age in *BC*. In the latter case, select v with probability $(1-p)$; otherwise, flip v_2 with second highest net gain.

2.2 A Genetic Programming Approach to Learning Heuristic Functions

The CLASS system was motivated by the following key observations:

First, the well-known, SAT local search heuristics could all be expressed as a combination of a relatively small number of primitives. These primitives included: (1) scoring of variables with respect to a gain metric, (2) ranking of variables and greediness. (3)variable age, and (4) conditional branching Heuristics such as GSAT, Walksat, Novelty, Novelty+, could be implemented as relatively simple functions built by composing the various primitives discussed above. Even

R-Novelty[7], which is one of the most complex heuristics proposed to date, can be represented as a 3-level decision diagram [5].

Second, historically, new heuristics were often generated by "structural" innovations which recombined existing primitives in new ways. For example, GWSAT and Novelty+ added random walk to GSAT and Novelty after observing behavioral deficiencies of the predecessors [11,5].

Third, it appeared relatively difficult for even expert researchers to design successful heuristics, and the design of new heuristics required However, some major structural innovations involve considerable exploratory empirical effort. For example, [7] notes that over 50 variants of Walksat were evaluated in their study (which introduced Novelty and R-Novelty).

From these observations, [1] claimed that the domain of *SAT heuristic design* seemed to be a good candidate for the application of a genetic programming approach. CLASS represents variable selection heuristics in a Lisp-like s-expression language. In each iteration of the innermost loop of the local search (Figure 1), the s-expression is evaluated in place of a hand-coded variable selection heuristic.

To illustrate the primitives built into CLASS, Figure 2 shows some standard heuristics represented as CLASS s-expressions. Due to space restrictions, we can not define the language here, but a more detailed description of the CLASS language primitives can be found in [1].

```
Walksat:
(IfVarCond == NegativeGain 0
   VarBestNegativeGainBC0
   (If (rand 0.5)
        VarBestNegativeGainBC0
   RandomVarBC0))
```

```
Novelty:
(IfNotMinAge BC0 VarBestNetGainBC0
    (If (rand 0.5)
        VarBestNetGainBC0
        VarSecondBestNetGainBC0))
```

Fig. 2. Walksat and Novelty represented in the CLASS language.

CLASS uses a population-based search algorithm similar to strongly-typed genetic programming [9] to search for good variable selection heuristics (Figure 3). This algorithm is conceptually similar to the standard genetic programming algorithm [6]. The `Initialize` function creates a population of randomly generated s-expressions. The expressions are generated using a context-free grammar as a constraint, so that each s-expression is guaranteed to be a syntactically valid heuristic that returns a variable index when evaluated. The `Select` function picks two s-expressions from the population, where the probability of selecting a particular s-expression is a function of its rank in the population according to its objective function score (the higher an expression is ranked, the more likely it is to be selected). The `Compose` operator (detailed below) is applied to the parents to generate a set of children, which are then inserted into the population. Each child replaces a (weighted) randomly selected member of the population, so that the population remains constant in size (the lower the ranking, the more likely it is that an s-expression will be replaced by a child). The best heuristic found during the course of the search is returned.

The most significant difference between the CLASS learning algorithm and standard strongly typed genetic programming is how children are created. Instead of applying the standard crossover and mutation operators, CLASS uses a *composition* operator, which is intended to generate children whose behavior is a blend of the parents' behavior. Given two s-expressions `sexpr1` and sexpr2, composition combines them by creating a new conditional s-expression where the "if" and "else" branches are sexpr1 and sexpr2, i.e., `(conditional-sexpr sexpr1 sexpr2)`. The actual composition operator results in 10 children being created at each mating, where each child is the result of a different conditional s-expression combining the two parents. Each child is evaluated, and then is inserted into the population. CLASS uses composition rather than the standard genetic operators because (1) composition was a good model of how human researchers derived some of the most successful SAT local search heuristics, and (2) if at least one of the parent heuristics has the PAC-property [4], a subset of the children derived via composition are guaranteed to be PAC[1].

We have also explored the use of a more "standard" strongly-typed GP using standard crossover, selection, and mutation operators to CL. However, we have not yet been able to discover a parametrization of the standard GP to be competitive with our current algorithm.

The composition operator used by CLASS naturally leads to code "bloat", since each child will be, on average, twice as complex as its parents. Uncontrolled bloat would be a major problem in CLASS because fast evaluation of the s-expression heuristics is critical to achieving our goal of fast local search runtimes. We implemented a depth bound so that any interior nodes in the s-expression tree that exceed the given depth bound in the run are transformed to a leaf node (with a compatible type).

```
Initialize(population,populationsize)
For I = 1 to MaxIterations
Select parent1 and parent2 from population
 Children = Compose(parent1,parent2)
 Evaluate(Children)
 Insert(Children,Population)
```

Fig. 3. CLASS Meta-Search Algorithm

2.3 Fitness Function

A candidate s-expression is executed on a set of training instances from the distribution of hard, randomly generated 3-SAT problems from the SAT/UNSAT phase transition region [8].[1] First, the heuristic is executed on 200, 50 variable,

[1] `mkcnf` (ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/contributed/UCSC/instances/Cnfgen.tar.Z) was used to generate training instances, using the flag to force satisfiable.

215-clause random 3-SAT instances, with a cutoff of 5000 flips. If more than 130 of these local search descents was successful, then the heuristic is run on 400, 100-variable, 430-clause random 3-SAT instances with a 20000 flip cutoff. The score of an individual is: (# of 50-var successes) + (5 * (# of 100-var successes)) + (1/*MeanFlipsInSuccessfulRuns*) The 50-variable problems serve to quickly filter out very poor individuals and not waste time evaluating them further with more expensive training instances, while the larger 100-variable problems enable us to gain finer distinctions between the better individuals.

2.4 CLASS2.0

From the above description, it should be clear that searching for good heuristics using CLASS is *extremely* time-consuming. Each call to the evaluation function requires hundreds of descent of a local search algorithm, and during each descent, thousands or variable flips are executed, and each flip requires the evaluation of the candidate heuristic, i.e., each call to the fitness function results in hundreds of thousands of evaluations of the candidate s-expression. Each run of the CLASS genetic programming system (5000 fitness evaluations) took several days on a Pentium-3 based workstation, making experimentation with the system very cumbersome.

Although the original version, **CLASS1.0,** was able to generate some promising results, as reported in [1], it became difficult to make further progress without making the system substantially faster. In addition, the most important outstanding question was whether the heuristics discovered by CLASS were actually better than the state of the art, human-designed heuristics after the additional complexity of the heuristic was considered. Throwing additional computing power at the problem (e.g., using additional, faster CPUs in a parallel GP system) would make it easier to perform experiments, but it would not settle the issue of whether the resulting heuristics were actually faster than the standard heuristics when executed on the same CPU. Furthermore, the use of the *library* feature in CLASS1.0 (i.e., the CLASS-L extensions), which seeded the initial population with hand-coded heuristics, made it difficult to assess how powerful the representation was.

Therefore, after carefully studying the various performance bottlenecks in CLASS1.0, we developed a new system, **CLASS2.0,** which sought to address the crucial performance issues. Profiling CLASS showed that there were two functions that consumed the vast majority of computational resources. Not surprisingly, the execution of the variable selection heuristic code was the biggest bottleneck, especially when the s-expressions representing the heuristic were large (since the complexity of evaluating the s-expression grows exponentially with the depth of the expression tree). Second, when the variable selection heuristics were small (roughly depth 3 or shallower), then the function for flipping a variable and incrementally updating the gain values (defined in section 2.1) which were used by the variable selection heuristic was a major bottleneck. We addressed both of these issues. Also, all of the the CLASS-L extension features were eliminated from CLASS2.0.

2.5 Faster S-expression Evaluation

CLASS1.0 was an ad-hoc, multilingual implementation, where the genetic programming component was implemented in `tcl`, and the local search engine was implemented in C++. The GP component would generate a candidate s-expression (as described above), then this s-expression would be translated to an equivalent C++ expression, which would then be compiled using the C++ compiler (gcc) and linked to the local search engine.

The fundamental problem with this architecture was that although the GP component must emit code that is executed by the local search component, it is extremely difficult and cumbersome, to correctly implement code in the GP component with enough introspective capability into the s-expression local search component such that the s-expressions could be optimized for fast execution. The current implementation, CLASS2.0, is implemented entirely in Common Lisp, allowing seamless integration between the GP and local search components. Although Lisp is often cited as being an interpreted language that is significantly slower than C/C++, modern ANSI Common Lisp implementations, using optional type declarations, are usually able to generate native machine code comparable (i.e., not much worse than a factor of 2) to code generated by C++ compilers. All of the results in this paper were obtained using CMUCL version 18e (http://www.cons.org/cmucl/)

Several improvements were implemented in order to enable faster expression of the evolved heuristic s-expressions. First, when a child is generated, CLASS2.0 probabilistically performs an *approximate simplification* of the s-expression. For example, consider the expression on the left below. The function `((rand x)` returns a random variable between 0 and x. Note that the probability of s-expr1 being evaluated is only 0.01. Thus, we can eliminate that subtree and "approximately" simplify the expression to the expression on the right.

```
(if (rand 0.1)                (if (rand 0.1)
    (if (rand 0.1)  -->           s-expr2
        s-expr1                   s-expr3)
        s-expr2)
    s-expr3)
```

This results in behavior that approximates the behavior of the original s-expression (i.e., 99% of the time, the modified expression returns the same value as the original expression), but executes faster due to the simpler structure. There is a tradeoff between the evaluation speedup gained by this technique versus possible loss in search efficiency due to the approximation.

Another improvement to s-expression evaluation was in the efficient computation and caching of subexpression values. For example, if an s-expression requires the evaluation of both "the variable in broken clause `BC1` with highest net gain", and "the variable in `BC1` with second highest net gain", then both of these can be computed by a single pass through the variables in BC1, rather than one pass for each query.

Also, if the same value is required multiple times during the evaluation of an s-expression, then it is worth caching the result when first computed. However, if

it can't be guaranteed that an expression is required more than once (e.g., when it appears in a conditional subexpression), then it may in fact hurt performance to pay the overhead of caching the first instantiation. Before evaluating an evolved s-expression, CLASS2.0 applies a number of rules to transform the original s-expression into a semantically equivalent expression where some of the functions and terminals are the caching equivalents of the original functions and terminals (if the transformation module determines it worthwhile to do so).

2.6 Faster Local Search Engine

The local search engine of CLASS1.0, while reasonably fast, did not have the fastest possible implementation of variable flipping. CLASS2.0 implements the most efficient local search engine found so far in the literature, with some enhancements that yielded significant speedups. A detailed description of these, as well as an extensive survey of efficient data structures and algorithms to support local search can be found in [2].

The standard implementation of SAT local search is the C implementation by Henry Kautz and Bart Selman (Walksat version 43).[2] This is an efficient, actively maintained implementation. We compiled Walksat-v43 using gcc and -O3 optimization. To evaluate the performance of the CLASS2.0 local search engine, we compared it against Walksat-v43, as well as CLASS1.0, as follows:

We hand-coded the standard Walksat-SKC heuristic [7] as an s-expression in the CLASS language (See Figure 2), generating CLASS1.0 and CLASS2.0 implementation of Walksat (Walksat-CLASS1.0, and Walksat-CLASS2.0). Each of the three solvers (Walksat-v43, Walksat-CLASS1.0, Walksat-CLASS2.0) was executed on the `uf250` set of 100, 250 variable, 1065 clause instances[3]. Each instance was run 10 times, with max-flips set to 10 million flips per instance. All experiments in this paper were performed on a 1.3GHz AMD Athlon CPU running Linux.

The results were as follows: All three solvers were 100% successful (i.e., all tries found satisfying assignments within the max-flips limit). Walksat-v43 solved all instances with an average of 73338 flips, executing 448962 flips per second. Walksat-CLASS2.0 solved all instances with an average of 68933 flips, executing 403781 flips per second. Walksat-CLASS1.0 solved all instances in 69014 flips, executing 278060 flips per second. As expected, the search effort *(#* of flips to solution) is almost identical for all three engines. The CLASS2.0 engine (implemented in Common Lisp) obtains 90% of the flip rate of the Walksat-v43 engine (implemented in C). CLASS2.0 is substantially faster than the CLASS1.0 engine (implemented in C++) due to significantly improved data structures. In theory, the CLASS2.0 engine should be somewhat faster if the entire system were rewritten in C, but we were satisfied that the underlying infrastructure was fast enough so that the evolved heuristics, when embedded into the CLASS2.0 framework, had a chance of outperforming Walksat-v43.

[2] available at http://www.cs.washington.edu/homes/kautz/walksat

[3] available at http://satlib.org

Table 1. Results with cutoff=300,000 flips: Success Rate (SR) (%), Average Flips to Solution (AFS), and Runtime (10 runs/instance for Walksat,Novelty+, C2-D3, and C2-D6

Algorithm	Suite B, n=75			SuiteB, n=100			Suite C, n=80			Suite C n=100		
	SR	AFS	time	SR	AFS	time	SR	AFS	time	SR	AFS	time
RFEA2+	96	39396	n/a	81	80282	n/a	95	49312	n/a	79	74459	n/a
ASAP	87	39659	n/a	59	43601	n/a	72	45492	n/a	61	34548	n/a
Walksat-v43	88.8	50706	0.21	62.6	44335	0.47	78.4	54178	0.34	67.0	40080	0.43
Novelty+	96.6	17108	0.06	71.6	34849	0.29	89.7	33518	0.16	75.7	31331	0.25
C2-D3	97.2	19646	0.11	78.6	40085	0.35	90.8	25744	0.19	80.9	37815	0.42
C2-D6	98.2	15199	0.16	82.0	38883	0.69	95.1	27527	0.32	83.8	36762	0.63

3 Empirical Evaluation of Evolved Heuristics

We compare the performance of a CLASS2.0-generated heuristic to the performance of the state of the art local search and evolutionary algorithms.

First, we extend the comparative study of SAT algorithms performed by Gottlieb, Marchiori, and Rossi [3] to include heuristics generated by CLASS. We used identical benchmark instances and computational resources, in order to allow a direct comparison with previously evaluated algorithms. The set `Suite B` consists of 50 75-variable problems and 50 100 variable problems. The set `Suite C` consists of 100 80-variable problems and 100 100-variable instances. All of the instances[4] are hard, satisfiable random 3-SAT instances from the phase-transition region, with a clause-to-variables ratio of 4.3.[5]

As with previous studies, we measured: (1) the *success rate,* which is the percentage of successful runs (runs where a solution is found), using a computation limit of 300,000 variable flips, and (2) the *average # of flips to solution (AFS),* the mean number of flips to find a solution in a run, when a solution was found (i.e., flips in unsuccessful runs are not counted).

The data for RFEA2+ and ASAPare copied from [3]. The results for Suite B are based on 50 runs per instance for RFEA+, and 10 runs per instance for ASAP. For Suite C, the data for RFEA2+ is for 4 runs, and ASAP is for 5 runs.

The results for Walksat-v43 were obtained by running Kautz and Selman's Walksat version 43 code (see Section 2.6) using the `-best` heuristic flag and a noise setting of 50. This is the same configuration as the **WSAT** runs reported in [3]; we reran the algorithm so that we could obtain runtimes for comparison.

We also tried to find the best possible configuration of the Walksat-v43 solver (which implements not only Walksat but all of the `novelty` variants) on this

[4] Downloaded from http://www.in.tu-clausthal.de/ gottlieb/benchmarks/3sat

[5] Due to space restrictions, we only show results for the *largest* problem instances in Suites B and C of [3]. Experiments were performed with all 662 instances from Suites A, B, and C. the results were qualitatively similar for Suite A (omitted here because there were only 12 instances), and the smaller instances in Suites B and C excluded from Figure 1.

Table 2. Results with cutoff=100,000,000 flips: Success Rate (SR) (%), Average Flips to Solution (AFS), and Runtime

Algorithm	Suite B, n=75			SuiteB, n=100			Suite C, n=80			Suite C n=100		
	SR	AFS	time	SR	AFS	time	SR	AFS	time	SR	AFS	time
Walksat-v43	100	244856	0.13	96	3134042	15.8	100	1790021	3.79	98	3633661	12.21
Novelty+	100	31027	0.07	100	2218112	5.42	100	167908	0.41	100	1228156	3.07
C2-D3	100	85555	0.34	100	267770	1.04	100	62194	0.24	100	233481	0.90
C2-D6	100	33656	0.28	100	404983	3.38	100	47506	0.40	100	268342	2.21

benchmark set, by trying the `-best`, `-novelty`, `-rnovelty`, `-novelty+`, and `+rnovelty+` heuristics for a noise ranging from 10 to 90 (tested at 5% increments), and report the results for the best configuration on this benchmark set, which was Novelty+ (noise=75).

The heuristics evolved by CLASS2.0 included in this study are **C2-D3** and **C2-D6.** These are the results of running CLASS2.0 for 5500 individual evaluations. C2-D3 was evolved with a depth limit on the s-expression of 3, while C2-D6 was evolved with a depth limit of 6. We emphasize that *the training instances used during evolution are different from the benchmark instances,* although they are from the same problem class (generated using the mkcnf utility with the same control parameters).

The results in Table 1 show that the evolved heuristics C2-D3, and C2-D6 are quite competitive with the previous evolutionary algorithms as well as the state of the art local search algorithms. C2-D6 has a higher success rate than C2-D3, although it is paying a price in runtime due to the greater complexity of its s-expression.

Note that we simply picked C2-D3 and C2-D6 from the first GP runs we performed using a depth 3 limit and depth 6 limit. Heuristics with similar scores can be routinely evolved (see Section 4)

While the above study followed the standard methodology and allowed a direct comparison with the results of [3], the relatively low cutoff limit and the use of the success rate + AFS metric makes it difficult to see exactly how much faster one algorithm is than another. To provide another perspective, we repeated the experiments using a much larger resource threshold of 100,000,000 flips, and 1 run per instance.

The results are as shown in Table 2. For the hardest problems, C2-D3 and C2-D6 clearly outperformed Walksat-v43 and Novelty+.

4 Empirical Evaluation of the Evolutionary Algorithm

Is the GP-like population-based algorithm used in CLASS (Figure 3) a reasonably efficient meta-level search algorithm? We compared our algorithm to a simple, random generate-and-test (G&T) algorithm. G&T simply samples the space of s-expressions by repeatedly calling the random s-expression generated used during the initialization of the population in Figure 3. We ran each algorithm 10 times. Each run involved 5500 fitness function evaluations. For the GP,

we generated and evaluated a population of 1000 and ran until 4500 children had been generated and evaluated. For G&T, 5500 random individuals were evaluated. Both algorithms were run with an s-expression depth limit of 6. We then compared the fitness function scores (Section 2.3) of the best individual found by the two algorithms. The mean value of the *best* individual score over 10 runs for the GP was 1377, and was 1213 for G&T. Also, the mean fitness of the final population in the GP was 1306, while the mean fitness of all individuals generated by G&T was 411. These results show that the GP was searching the space more effectively than G&T, and that the GP population was focusing on high-quality individuals (as opposed to getting lucky and generating a single better individual than G&T).

5 Conclusions

This paper summarized extensions to the work started in [1]. We showed that CLASS2.0, a genetic programming system for learning variable selection heuristics for SAT local search could generate heuristics that were truly competitive (both in terms of runtime and search efficiency) with state of the art local search algorithms, as well as previous evolutionary approaches.

Previously, a number of algorithms have been proposed which outperform Walksat and Novelty+ with respect to the number of flips required to solve satisfiable instances, including those reported in [1], as well as algorithms that used more sophisticated objective functions and/or clause weighting (e.g., SDF [10] and previous evolutionary algorithms [3]). Such previous studies have been more concerned with search efficiency than with the runtimes of the algorithms. However, search efficiency does not necessarily imply a superior algorithm, because the per-flip overhead of an efficient search algorithm , incurred due to a more complex objective function (e.g., clause weighting) or variable selection heuristic may be inherently too high for it to outperform the standard algorithms in practice. With the recent interest in real-world applications that have been formulated as SAT instances, SAT is no longer merely an academic benchmark problem. In the end, all that matters is speed, and a SAT solver must actually *run faster* than the standard algorithms in order to be truly useful to practitioners. From this perspective, CLASS1.0 was *far* from the goal of generating human-competitive results. A substantial engineering effort to implement CLASS2.0 was required in order to demonstrate that faster SAT solvers could indeed be generated using this approach.

Thus, the main contribution of this paper is a demonstration that genetic programming can be used to attack an important problem that has been intensively studied by human researchers for over a decade, and can routinely generate algorithms that objectively outperform the state of the art, hand-developed heuristics for a class of difficult instances. The performance of the evolved heuristics generalize fairly well to instances from other classes of SAT instances (not presented here due to space constraints, but generalization results for `C2-D3` and `C2-D6` are similar to those obtained for the heuristics evolved by CLASS-1.0[1].

Further improvements to the learning process to maximize the generalization is the major area of future work.

In addition, we showed that this could be accomplished without seeding the population with a "library" of hand-coded solutions or excessively complex primitives, as was done in [1]. We also showed that the CLASS genetic programming algorithm significantly outperformed random search in the space of CLASS-language s-expressions. These results confirm that the CLASS population-based algorithm is indeed a reasonable meta-learning algorithm, i.e., that the power of the system is not just in the representation but in the combination of the representation and the search algorithm.

Note: Source code for CLASS, including the C2-D3 and C2-D6 expressions, is available at the author's home page (search Google for current location).

References

[1] A. Fukunaga. Automated discovery of composite sat variable-selection heuristics. In *Proc. AAAI,* pages 641–648, 2002.

[2] A. Fukunaga. Efficient implementations of sat local search, to appear in Proceedings of SAT-2004, May 2004.

[3] J. Gottlieb, E. Marchiori, and C. Rossi. Evolutionary algorithms for the satisfiability problem. *Evolutionary Computation,* 10(1):35–50, 2002.

[4] H. Hoos. *Stochastic local search - methods, models, applications.* PhD thesis, TU Darmstadt, 1998.

[5] H. Hoos and T. Stutzle. Local search algorithms for sat: An empirical evaluation. *Journal of Automated Reasoning,* 24:421–481, 2000.

[6] J. Koza. *Genetic Programming: On the Programming of Computers By the Means of Natural Selection.* MIT Press, 1992.

[7] D. McAllester, B. Selman, and H. Kautz. Evidence for invariants in local search. In *Proc. AAAI,* pages 459–465, 1997.

[8] D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of sat problems. In *Proc. AAAI,* pages 459–65, 1992.

[9] D. Montana. Strongly typed genetic programming. Technical report, Bolt, Beranek and Neuman (BBN), 1993.

[10] D. Schuurmans and F. Southey. Local search characteristics of incomplete sat procedures. *Artificial Intelligence,* 132:121–150, 2001.

[11] B. Selman and H. Kautz. Domain-independent extensions to gsat: Solving large structured satisfiability problems. In *Proc. Intl. Joint Conf. Artificial Intelligence (IJCAI),* 1993.

[12] B. Selman, H. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proc. AAAI,* 1994.

[13] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proc. AAAI,* pages 440–446, 1992.

Shortcomings with Tree-Structured Edge Encodings for Neural Networks

Gregory S. Hornby

QSS Group Inc., NASA Ames Research Center
Mail Stop 269-3, Moffett Field, CA 94035-1000
hornby@email.arc.nasa.gov

Abstract. In evolutionary algorithms a common method for encoding neural networks is to use a tree-structured assembly procedure for constructing them. Since node operators have difficulties in specifying edge weights and these operators are execution-order dependent, an alternative is to use edge operators. Here we identify three problems with edge operators: in the initialization phase most randomly created genotypes produce an incorrect number of inputs and outputs; variation operators can easily change the number of input/output (I/O) units; and units have a connectivity bias based on their order of creation. Instead of creating I/O nodes as part of the construction process we propose using parameterized operators to connect to pre-existing I/O units. Results from experiments show that these parameterized operators greatly improve the probability of creating and maintaining networks with the correct number of I/O units, remove the connectivity bias with I/O units and produce better controllers for a goal-scoring task.

1 Introduction

Neural networks are one of the more common types of controllers used for artificial creatures and evolutionary robotics [1]. Since representations that directly encode the weights and connections of a network have scalability problems indirect representations must be used for larger networks – although to achieve better scalability the indirect representation must allow for reuse of the genotype in creating the phenotype [2]. One type of indirect representation that is becoming increasingly popular for encoding neural networks is to use a tree-structured genotype which specifies how to construct them. Advantages of indirect, tree-structured representations are that they better allow for variable sized networks than directly using a weight matrix, and Genetic Programming style recombination between two trees is easier and more meaningful than trying to swap sub-networks with a graph-structured representation.

One of the original systems for encoding neural networks in tree-structured assembly procedures is cellular encoding [3]. Yet cellular encoding has been found to have shortcomings due to its use of node operators: subtrees swapped through recombination do not produce the same subgraphs because node operators are execution-order dependent and specifying connection weights is prob-

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 495–506, 2004.

lematic since node operators can create an arbitrary number of edges [4]. Consequently, of growing interest is the use of edge-encoding commands in which operators act on edges instead of nodes [5,6,7].

In this paper we point out three different shortcomings of edge-encoding languages. First, regardless of whether the first N nodes are taken as input/output (I/O) units or if special node-construction commands are used for creating I/O units, when creating an initial population it is difficult to ensure that randomly created genotypes have the correct number of them. A second problem is that as evolution proceeds the variation operators have a high probability of changing the genotype so that it produces networks with incorrect numbers of I/O units. Finally, a more serious problem with tree-structured assembly procedures is the node creation-order connectivity bias (NCOCB). The NCOCB problem is that nodes created from edge operators at the bottom of the genotype will have only a single input and output, whereas nodes created from operators higher up in the genotype will have a connectivity proportional to 2^{height}.

One way to address the problems of producing the correct number of I/O nodes and the NCOCB with I/O nodes is by changing the construction language. Rather than having commands in the language for creating a new I/O unit, or assigning the Nth created unit as the ith I/O unit, we propose starting network construction with the desired number of I/O units and then using parameterized-connection operators for adding edges to these units. Problems in creating and maintaining networks with the correct number of I/O units are reduced since all networks start with the desired number and no commands exist for creating/removing them. Also, parameterized connection commands mean that the expected number of connections for all I/O units is equal for randomly created genotypes and does not suffer from the NCOCB.

In the following sections we first describe a canonical method for using edge encoding operators to represent neural networks as well as our parameterized operators for connecting to I/O units. Next we present our experiments which show the different biases with standard edge-encoding operators and demonstrate that evolution with the parameterized operators for connecting to I/O units produces better controllers on a goal-scoring task. Finally we close with a discussion on the underlying problem with edge operators and tree-structured representations and a conclusion in which we restate our findings.

2 Encoding Neural-Networks

In this section after describing the type of neural networks that we want to evolve we then describe a tree-structured representation for encoding them, followed by two different methods for handling input and output (I/O) units. The first method for handling I/O units uses a standard edge-encoding language (SEEL) and has special commands for creating I/O nodes. Since this method has problems in creating the correct number of I/O nodes and also has a node creation-order connectivity bias (NCOCB) we then describe a second method for handling I/O units. In this second method the initial network starts with

the desired number of I/O units and operators in the language connect to them using operator parameters to specify which unit to connect to (PEEL, for parameterized edge-encoding language).

2.1 Neural Network Architecture

The neural networks used in these experiments are continuous-time, recurrent networks similar to those of Beer and Gallagher [8], and of our previous work [9,10]. Each non-input neuron has an input bias, θ, and a time constant, τ. The activation value of a non-input neuron a_i at time t is:

$$a_{i,t} = \tau_i a_{i,t-1} + (1-\tau) tanh\Big(\sum_j W_{ji} a_{j,t-1} + \theta_i\Big) \quad (1)$$

For input neurons, their activation value is the value of the corresponding sensor.

2.2 Creating a Network from a Tree

The different methods for encoding neural networks or graphs with a tree-structured assembly procedure all start with a single node and edge and then new nodes/edges are added by executing the operators in the assembly procedure. Using an edge-encoding language in which graph-construction operators act on the edge connecting from unit *A* to unit *B,* a typical set of commands are as follows.

- `add_reverse` creates a link from *B* to *A*.
- **`add_split`**(n) creates a new neuron, *C,* with a bias of $\theta = n$, and adds a new link from *A* to *C* and creates a new edge connecting from neuron *C* to neuron *B*. The bias of this node is set to $\theta = n$, and the time constant is set to zero.
- **`add_split_cont`**(m, n) acts the same as `add_split ( )`, only it creates a continuous time neuron with a bias of $\theta = m$ and a time constant of $\tau = n$.
- connect creates a new link from neuron *A* to neuron *B*.
- `dest_to_next` changes the to-neuron in the current link to its next sibling,
- `loop` creates a new link from neuron *B* to itself. The `no-op` command performs no operation.
- **`set_weight`**(n) sets the weight of the current link to n.
- `source_to_next` changes the from-neuron in the current link to its next sibling.
- `source_to_parent` changes the from-neuron in the current link to the input-neuron of the current from-neuron.

Of these commands **`add_split`**(n) and **`add_split_cont`**(m, n) have exactly three children commands since after their execution the edge they act on becomes three edges. The **`set_weight`**(n) command has no children, consequently it is always a leaf node and the `no-op` has either zero or one children so it can be either a leaf node and halt the development of the graph on the current edge,

or it can be used to delay execution on the current edge for a round allowing time for the rest of the graph to develop more. The rest of the commands result in the addition of a single new edge to the graph so they have exactly two children commands: one to continue graph construction along the original edge one command to construction along the new edge.

Using the commands described above the sequence of graphs from figure 1b-i shows the construction of a network from the genotype in figure 1.a. Graphs are constructed from this genotype by starting with a single neuron linked to itself, figure 1.b, and executing the commands in the assembly procedure in breadth-first order. First, executing `split(0.01)` adds node `b` with a bias of 0.01 and pair of links, figure 1.c. The command `set-weight (0.1)` sets the weight of the link $\overrightarrow{aa}$ to 0.1, `no-op` performs no operation, and then `split(0.02)` results in the creation of neuron `c` with a bias of 0.02 and two more links, figure 1.d. `Source-to-parent` creates a second link $\overrightarrow{ab}$, and `set-weight (0.2)` sets the weight of the link $\overrightarrow{ba}$ to 0.2, figure 1.e. The second `source-to-parent` command creates the link $\overrightarrow{ac}$, executing `set-weight (0.3)` sets the weight of the link $\overrightarrow{ca}$ to 0.3 and `set-weight(-0.2)` results in a weight of -0.2 assigned to the link $\overrightarrow{ab}$, figure 1.f. The `source-to-next` command results in the link $\overrightarrow{bb}$ being created, figure 1.g. The command `set-weight(0.4)` sets the weight of link $\overrightarrow{bc}$ to 0.4 and then executing connect creates an additional link $\overrightarrow{ac}$, figure 1.h. Executing `set-weight(0.5)` sets the weight of link $\overrightarrow{ab}$ to 0.5, `set-weight(0.6)` sets the weight of link $\overrightarrow{bb}$ to 0.6, `no-op` sets the weight of link $\overrightarrow{ab}$ to 0.0, and `set-weight (0.7)` sets the weight of link $\overrightarrow{ac}$ to 0.7, figure 1.i. In addition, after all tree-construction operators have been executed, there is a pruning phase that consolidates the weights of links with the same source and destination neurons, figure 1.j, and removes hidden neurons that are not on a directed path to an output neuron.

2.3 Standard Edge Encoding Language

The graph-construction language of the previous sub-section can be used to create neural networks either by assigning the first n units as I/O units or by adding commands specifically for creating I/O units. Assigning arbitrary units to be I/O units has the drawback that changes in the genotype can add/delete units in the network so that units shift position and what was a the ith input unit in the parent becomes the $i+1$ input unit in the child. To avoid this disruption the SEEL we use has specialized commands for creating I/O units.

I/O units are created through the use of the `add_input` and `output_split`(n) commands. Since these are edge operators, we label the edge they are associated with to be from the vertex `A` to the vertex `B`. Executing the `add_input` command creates a new input unit and an edge connecting from this unit to `A`. Output units are created with the `output_split`(n) command, which performs a `split` on the existing edge and the newly created neuron is set as an output unit with a bias of $\theta = n$.

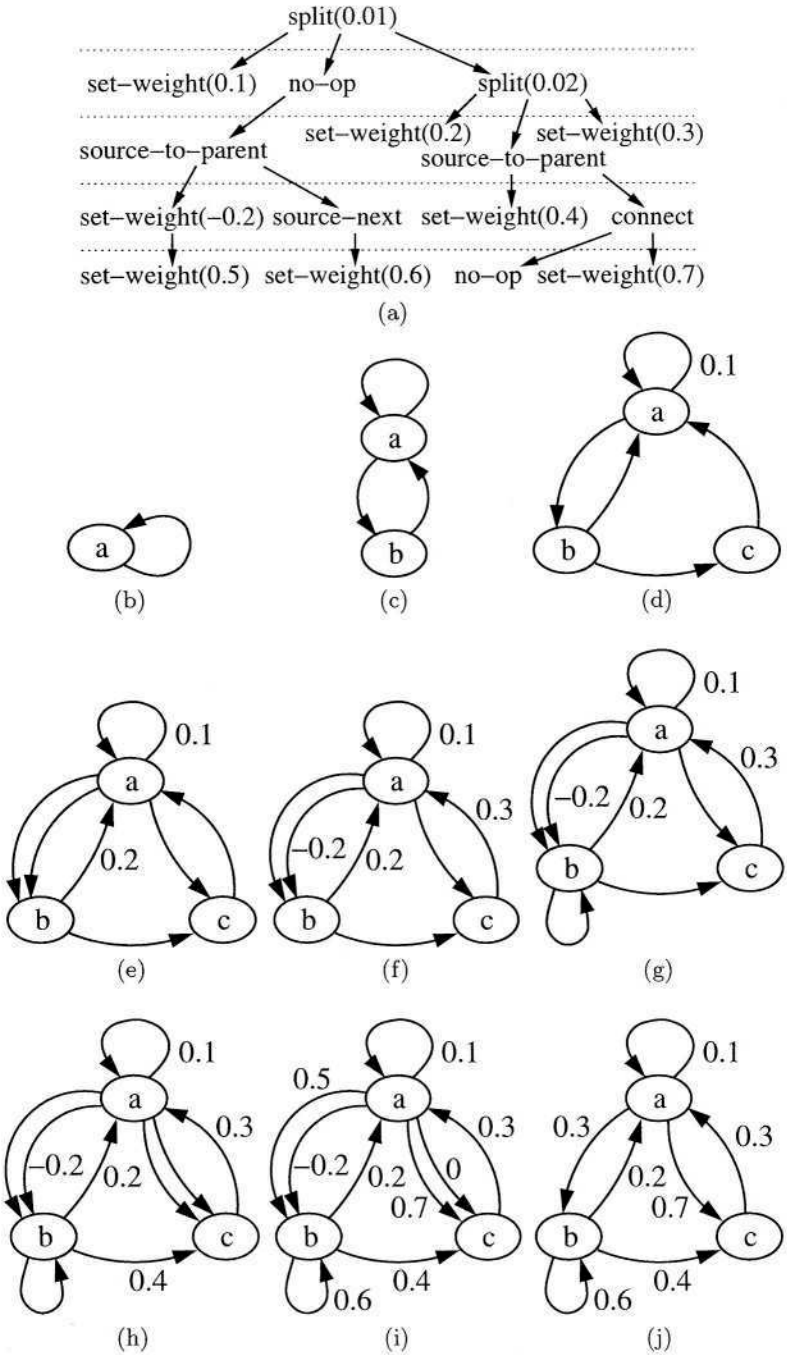


Fig. 1. A tree-structured encoding of a network (a), with dashed-lines to separate the layers, and (b-j) construction of the network it encodes.

2.4 Parametric Edge Encoding Language

A method to remove the connectivity bias with I/O nodes is by having these nodes exist in the initial graph and then adding connections to them, such as with the commands `connect_input(`i`)` and `connect_output(`i`)`. Labeling the current edge as connecting from unit A to unit B, `connect_input(`i`)` creates a link from the ith input neuron to B and `connect_output(`i`)` creates a link from B to the ith output neuron. Since each of these commands creates a new edge, both commands have exactly two children operators: one to continue network construction along the original edge and one to construct along the new edge.

3 Experiments

In this section we present our experiments comparing SEEL with PEEL. First we show that randomly created genotypes using SEEL have problems producing networks with the desired number of I/O neurons whereas this problem is greatly reduced when using PEEL. Next we show that networks encoded with PEEL are more robust to maintaining the correct number of I/O units under mutation and recombination than are networks encoded with SEEL. In our third set of experiments we demonstrate the existence of the node creation-order connectivity bias. Finally, we demonstrate that using PEEL results in better neural-controllers for the evolution of a goal-scoring behavior.

3.1 Initialization Comparison

One benefit of starting with the desired number of I/O neurons is that randomly created, network-constructing, assembly procedures are more likely to have the correct number of I/O neurons. This can be shown by comparing the number of valid networks created using both network construction languages. A network is considered valid if it has four input neurons and four output neurons (arbitrary values selected for this experiment) and for each input neuron there is a path to at least one output neuron and each output neuron is on a path from at least one input neuron. Table 1 shows the number of valid networks created from ten thousand randomly created assembly procedures for various tree depths. From this table it can be seen that valid networks are significantly more likely to be created with PEEL than with SEEL. The reason PEEL does not score 100% even though it starts with the correct number of I/O neurons is because some input neurons may not be on a path to an output neuron.

3.2 Variation Comparison

In addition to the problem of creating initial individuals with the correct number of I/O units, SEELs have difficulty maintaining these numbers under mutation and recombination. To show that PEELs better maintain valid networks we compare the number of networks that still have four inputs and four outputs after mutation and recombination from valid parents.

Table 1. Number of valid networks generated out of ten thousand randomly created tree-structured assembly procedures.

Depth	≤4	5	6	7	8	9	10	11	12	13
SEEL	0	3	103	183	93	34	13	6	2	0
PEEL	0	0	12	314	1973	4657	6733	8072	8643	8848

For this comparison the mutation operator modifies an individual by changing one symbol with another, perturbing the parameter value of a symbol, adding/deleting some symbols, or recombining an individual with itself. Two types of recombination are used, with equal probability of using one or the other. The first recombination operator is the standard GP recombination that swaps random subtrees between parents [11]. The second recombination operator is similar to one-point crossover [12] and we call it matched recombination (MR). MR works by lining up two trees and, starting at the root, matches up the children nodes by type and argument values, finds the locations at which subtrees differ and then picks one of these places at random to swap.

Since random trees of depth seven produced the most valid networks with SEEL, we compared ten thousand mutations and recombinations between SEEL and PEEL on valid, randomly created individuals. With SEEL mutation had a success rate of 84.8% and recombination had a success rate of 79.2%. In comparison, with PEEL mutation produced valid children 93.5% of the time and recombination did so 89.5% of them. These results show that networks encoded with a PEEL are more robust to variation operators.

3.3 Node Creation Order Connectivity Bias

A more serious problem with tree-structured assembly procedures is the node creation-order connectivity bias (NCOCB). Nodes created from commands early in the construction process tend to have a greater number of edges into and out of them then nodes created later in the the process. One consequence of this bias is that I/O neurons that are created early in the construction process will have a significantly higher number of outputs/inputs than those I/O neurons created at the end of the construction process.

The graph in figure 2.a shows the average connectivity (sum of inputs and outputs) of a node plotted against its creation order. From this graph it can be seen that nodes created earlier in the construction process have more connections than those created later and most nodes only have two connections: one input and one output link. Thus if I/O nodes are created by the tree-structured assembly procedure, the first I/O nodes will have significantly more inputs/outputs from/to them than those created later in the construction process. For input neurons, this suggests that the first inputs are likely to have a greater influence on the behavior of the network than the latter inputs and for output neurons this suggests that more processing of inputs is happening for the activation values of the first output neurons than for the latter output neurons.

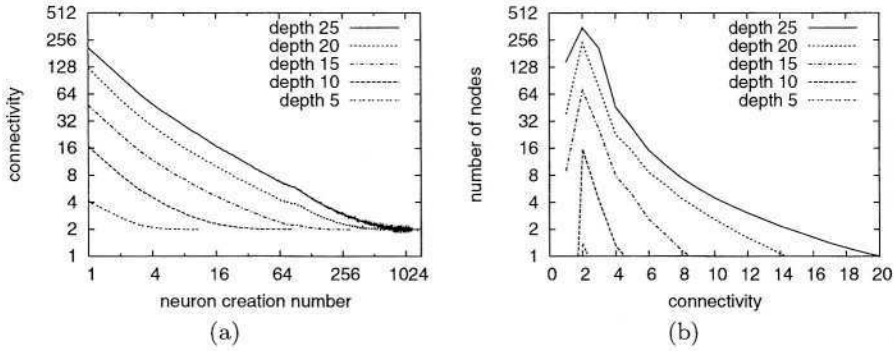


Fig. 2. Graphs of (a) the average node connectivity by order of creation, and (b) the number of nodes with a given connectivity for randomly created individuals.

Because the connectivity of a node is strongly biased by its height in the tree-structured assembly procedure and since most commands are near the leaves in the tree this results in a bias in the distribution of the number of nodes with a given connectivity. Most nodes in the network will have a connectivity of two – one input and one output – and the number of nodes with a given connectivity decreases exponentially (figure 2.b).

3.4 Comparison on Evolving a Goal-Scoring Behavior

While PEEL has been shown to be better than SEEL for removing various biases, of practical importance is whether evolution with PEEL produces better controllers than evolution with SEEL. To test this we evolve neural-controllers for a goal-scoring task.

Goal-scoring takes place in a computer-simulated, 275x152.5 walled, soccer field with goals at each end (figure 3.a). Inside the soccer field is a two-wheeled, soccer player which has seven sensor inputs (three sensors to detect distance to the wall (one pointing directly in front and the other two at 30° to the left and right), and four sensors that return angle to the ball, distance to the ball, angle to the goal and distance to the goal) and two outputs (desired wheel-speed for the left and right wheels) (figure 3.b).

Evaluating an individual consists of summing the score from eight trials, two each with the ball initially placed in each of the four corners of the field, and the soccer-player placed in the middle of the field. Initial locations for both the player and ball are perturbed by a small random amount and then the player is given 60 seconds (at 30fps this results in 1800 network updates) to score as many goals as it can. For each goal scored the distance from the vehicle's starting position to the ball plus the distance from the ball's initial location to the goal is added to the network's score. After a goal is scored the ball is randomly located at the center of the field (±30, ±30), the minimum distances to the ball and to

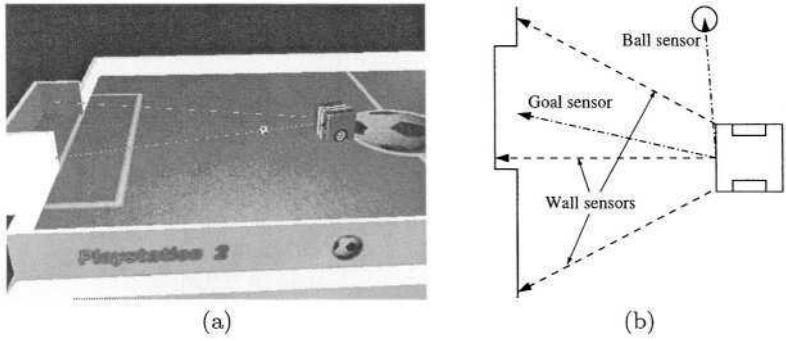


Fig. 3. (a) The soccer field and (b) the soccer player and its sensors.

the goal are reset, and the network is allowed to try to score another goal. Once time runs out, a network's score is increased by how much closer it moved the player to the ball and how much closer it moved the ball to the goal. In addition, if the player scores an own-goal, its score is reduced by the distance it moved the ball from its starting position to the goal.

To perform these experiments the EA was run on a Linux-PC with evaluations farmed out to five PlayStation® 2[1] development systems. Each experiment consisted of evolving fifty individuals for fifty generations. A generational EA was used and new individuals were created with equal probability of either mutation or recombination and an elitism of three. Evaluating one generation of fifty individuals took approximately four minutes. The results of experiments are shown in figure 4 and show that evolution with PEEL produces soccer players almost twice as fit as with SEEL.

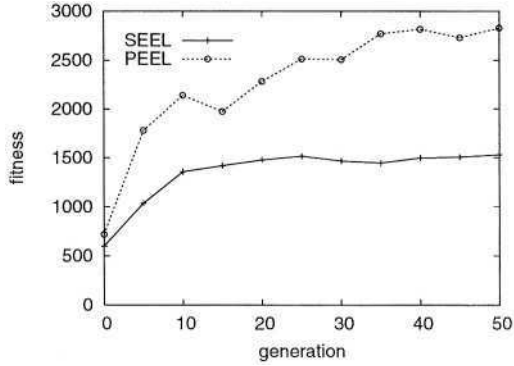


Fig. 4. Fitness of the best evolved goal-scores averaged over four trials.

[1] PlayStation is a registered trademark of Sony Computer Entertainment Inc.

The higher fitness of networks encoded with PEEL is reflected in the behaviors produced by these networks. Networks encoded with SEEL produced soccer players that tended to spin in place and move awkwardly or in a looping pattern. These networks only moved toward the ball somewhat haphazardly and generally did not appear to be aiming their shots. In contrast, networks encoded with PEEL would move to position themselves on the other side of the ball from the goal and then either push the ball toward the goal or spin to kick it toward the goal. The best of these networks seldom missed in its shots and an example of its behavior is shown in the sequence of images in figure 5.

4 Discussion

While the results of the experiments section show that various biases hold for the edge-encoding languages presented here, of interest is the degree to which these biases exist in other edge-encoding languages. The edge-encoding language of section 2 differs from Luke's [4] in that edges are not explicitly deleted, rather they disappear if they are not assigned a weight, and the split command does not delete the link $\overrightarrow{ab}$ when it creates the new neuron c and links $\overrightarrow{ac}$ and $\overrightarrow{cb}$. A command for explicitly deleting links would not necessarily change the biases in resulting networks since the `no-op` command with no children has the same effect. In contrast, since the split operator used here adds links to existing neurons without removing any, it should produce a larger bias than Luke's split operator. Although the differences in operators between different edge encoding languages affect the degree of connectivity bias that can be expected, the main cause of the biases is the tree-structure of the representation. When a neuron is created it has a single input and output edge. Since edge operators can add one input or output to an existing neuron (except for the `loop` command, which adds both an input and an output) the expected connectivity of a neuron is on the order of 2^{height}.

Since PEEL only addresses the NCOCB for I/O units and does not scale for large networks the direction to go for addressing the various shortcomings of edge encodings is not clear. One way to remove the NCOCB is to change from tree-structured to graph-structured genotypes, but then there are difficulties in creating meaningful recombination operators. Another way is to switch to operators in which the connectivity of a new node is not dependent on its depth in the genotype; but these would be node operators which have their own shortcomings [4].

5 Conclusion

In this paper we identified three shortcomings with typical edge encoding operators for representing neural networks: individuals created at random in the initialization phase do not usually have the correct number of inputs/outputs; variation operators can easily change the number input/output neurons; and the node creation-order connectivity bias (NCOCB). To address these problems we

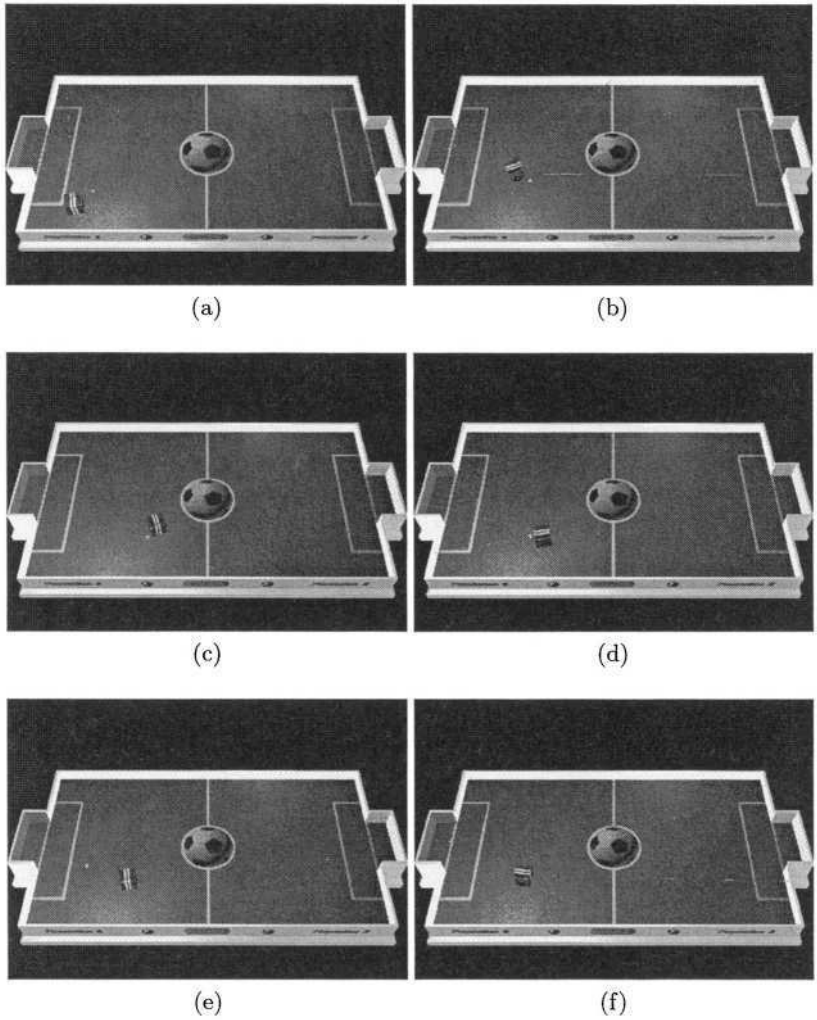


Fig. 5. An evolved goal-scorer in action: (a)-(c) the soccer-player circles around the ball; (d) it pushes the ball toward the goal; (e)-(f), while the ball is going into the goal the player moves to the center of the field where the ball will re-appear after the goal is scored.

proposed using parameterized operators for connecting to input/output units and demonstrated that evolution with these operators produces better neural networks on a goal-scoring task. While these parameterized operators greatly improve the probability of creating and maintaining networks with the correct number of input/output units it does not address the NCOCB problem for hidden units. Consequently the contribution of this paper is more an observation that these shortcomings exist. Future work with edge encoding operators will

need to address more general solutions to these problems that scale with the size of the network and work for hidden units.

Acknowledgements. Most of this research was conducted while the author was at Sony Computer Entertainment America, Research and Development and then at Brandeis University. The soccer game and simulator was developed by Eric Larsen at SCEA R&D.

References

1. Nolfi, S., Floreano, D., eds.: Evolutionary Robotics. MIT Press, Cambridge, MA (2000)
2. Hornby, G.S., Pollack, J.B.: Creating high-level components with a generative representation for body-brain evolution. Artificial Life **8** (2002) 223–246
3. Gruau, F.: Neural Network Synthesis Using Cellular Encoding and the Genetic Algorithm. PhD thesis, Ecole Normale Supérieure de Lyon (1994)
4. Luke, S., Spector, L.: Evolving graphs and networks with edge encoding: Preliminary report. In Koza, J., ed.: Late-breaking Papers of Genetic Programming 96, Stanford Bookstore (1996) 117–124
5. Brave, S.: Evolving deterministic finite automata using cellular encoding. In Koza, J.R., Goldberg, D.E., Fogel, D.B., Riolo, R.L., eds.: Genetic Programming 1996: Proceedings of the First Annual Conference, Stanford University, CA, USA, MIT Press (1996) 39–44
6. Hornby, G.S., Pollack, J.B.: Body-brain coevolution using L-systems as a generative encoding. In: Genetic and Evolutionary Computation Conference. (2001) 868–875
7. Koza, J., Bennett, F., Andre, D., Keane, M.: Genetic Programming III: Darwinian Invention and Problem Solving. Morgan Kaufmann (1999)
8. Beer, R.D., Gallagher, J.G.: Evolving dynamical neural networks for adaptive behavior. Adaptive Behavior **1** (1992) 91–122
9. Hornby, G.S., Mirtich, B.: Diffuse versus true coevolution in a physics-based world. In Banzhaf, W., et al., eds.: Proc. of the Genetic and Evolutionary Computation Conference, Morgan Kaufmann (1999) 1305–1312
10. Hornby, G.S., Takamura, S., Hanagata, O., Fujita, M., Pollack, J.: Evolution of controllers from a high-level simulator to a high dof robot. In Miller, J., ed.: Evolvable Systems: from biology to hardware; Proc. of the Third Intl. Conf. Lecture Notes in Computer Science; Vol. 1801, Springer (2000) 80–89
11. Koza, J.R.: Genetic Programming: on the programming of computers by means of natural selection. MIT Press, Cambridge, Mass. (1992)
12. Poli, R., Langdon, W.B.: Schema theory for genetic programming with one-point crossover and point mutation. Evolutionary Computation **6** (1998) 231–252

Adapting Representation in Genetic Programming

Cezary Z. Janikow

Department of Math and Computer Science
University of Missouri–St. Louis
St Louis, MO 63121 USA
cjanikow@ola.cs.umsl.edu

Abstract. Genetic Programming uses trees to represent chromosomes. The user defines the representation space by defining the set of functions and terminals to label the nodes in the trees. The sufficiency principle requires that the set be sufficient to label the desired solution trees. To satisfy this principle, the user is often forced to provide a large set, which unfortunately also enlarges the representation space and thus, the search space. Structure-preserving crossover, STGP, CGP, and CFG-based GP, give the user the power to reduce the space by specifying rules for valid tree construction. However, the user often may not be aware of the best representation space, including heuristics, to solve a particular problem. In this paper, we present a methodology, which extracts and utilizes local heuristics aiming at improving search efficiency. The methodology uses a specific technique for extracting the heuristics, based on tracing first-order (parent-child) distributions of functions and terminals. We illustrate these distributions, and then we present a number of experimental results. ...

1 Introduction

Genetic programming (GP), proposed by Koza [2], solves a problem by utilizing a population of solutions evolving under limited resources. The solutions (chromosomes), are evaluated by a problem-specific user-defined evaluation method. They compete for survival based on this evaluation, and they undergo simulated evolution by means of simulated crossover and mutation operators.

GP differs from other evolutionary methods by using trees to represent potential problem solutions. Trees provide a rich representation that is sufficient to represent computer programs, analytical functions, and variable length structures, even computer hardware [1][2]. The user defines the representation space by defining the set of functions and terminals labelling the nodes of the trees. One of the foremost principles is that of *sufficiency* [2], which states that the function and terminal sets must be sufficient to solve the problem. The reason is obvious: every solution will be in the form of a tree, labelled only with the user-defined elements. Sufficiency usually forces the user to artificially enlarge the sets to avoid missing some important elements. This unfortunately dramatically

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 507–518, 2004.

increases the search space. Even if the user is aware of the functions and terminals needed in a domain, he/she may not be aware of the best subset to solve a particular problem. Moreover, even if such a subset is identified, questions about the specific distribution of the elements of the subset may arise. One question is whether all functions and terminals should be equally available in every context, or whether there should be some heuristic distribution. For example, a terminal *t* may be required but never as an argument to function *f1*, and maybe just rarely as an argument to *f2*. All of the above are obvious reasons for designing:

- methodologies for processing such heuristics,
- methodologies for automatically extracting those heuristics.

Methodologies for processing user heuristics have been proposed over the last few years: structure-preserving crossover [2], STGP [6], CGP [3], and CFG-based GP [9].

This paper presents a methodology for extracting such heuristics, called *Adaptable Constrained GP* (ACGP). It is based on the technology of CGP, which allows for efficient processing syntax, semantics, and heuristic constraints in GP [3]. In Sec. 2, we briefly describe the CGP technology. In Sec. 3, we introduce the ACGP methodology for extracting heuristics, and then present the specific distribution-based technique that was implemented for the methodology. In Sec. 4, we define the problem we will use to illustrate the technique, trace the first-order distribution of functions/terminals during evolution, and present some results. Finally, in concluding Sec. 5, we elaborate on future work needed to extend the technique and the methodology.

2 The CGP Technology

Even in early GP applications, it became apparent that functions and terminals should not be allowed to mix in an arbitrary way. For example, a 3-argument *if/else* function should use, on its condition argument, a subtree that computes a Boolean and not temperature or angle. Because of the difficulties in enforcing these constraints, Koza has proposed the principle of *closure* [2], which usually requires very elaborate semantic interpretations to ensure the validity of any subtree in any context. Structure-preserving crossover was introduced as the first attempt to handle such constraints [2] (the primary initial intention was to preserve structural constraints imposed by automatic modules ADFs).

Structure–preserving crossover wasn't a generic method. In the nineties, three independent generic methodologies were developed to allow problem–independent constraints on the tree construction. Montana proposed STGP [6], which used types to control the way functions and terminals can label local tree structures. For example, if the function *if* requires Boolean as its first argument, only Boolean–producing functions and terminals would be allowed to label the root of that subtree. Janikow proposed CGP, which originally required the user to explicitly specify allowed and/or disallowed local tree structures [3]. These local constraints could be based on types, but also on some problem specific

heuristics. In v2.1, CGP also added type–processing capabilities, with function overloading mechanisms. For example, if a subtree needs to produce an integer, and we have the function + *(add)* overloaded so that it produces integers only if both arguments are integers, then only this specific instance of *add* would be allowed to label the root of that subtree. Finally, those interested more directly in program induction following specific syntax structure have used similar ideas to propose CFG-based GP (*e.g.,* [9]).

CGP relies on closing the search space to the subspace satisfying the desired constraints. The constraints are local distribution constraints on labelling the tree (only parent–child relationships can be efficiently processed) — the processing was shown to impose only constant overhead for mutation and one more tree traversal for crossover [3].

CGP v1 allowed processing only parent–one–child contexts. This context constraint is independent of the position of the subtree in the tree, and of the other labels beyond this context (even the siblings)[1]. CGP v2 has one additional unique feature. It allows a particular local context to be weighted, to reflect some detailed heuristics. For example, it allows the user to declare that the function *if,* even though it can use either *f1* or *f2* for its condition child, it should use *f1* more frequently. This particular efficient technology is utilized in ACGP to express and process the heuristics.

Previous experiments with CGP have demonstrated that proper constraints can indeed greatly enhance the evolution, and thus improve problem–solving capabilities. However, in many applications, the user may not be aware of those proper constraints. For example, as illustrated with the 11-multiplexer problem, improper constraints can actually reduce GP's search capabilities while proper constraints can greatly speed up evolution [3]. This paper presents a new methodology, which automatically updates the constraints, or heuristics, to enhance the search characteristics with respect to some user-defined objectives (tree quality and size at present). In what follows, we describe the methodology and a specific technique implementing it, and then present some experimental results.

3 ACGP and the Local Distribution Technique

ACGP is a methodology to automatically modify the heuristic weights on typed mutation sets[2] in CGP. The basic idea is that there are some heuristics on the distribution of labels in the trees both at the local level (parent–child) and at a more global level (currently not done). These ideas are somehow similar to those applied in Bayesian Optimization Network [7], but used in the context of GP and functions/terminals and not binary alleles.

We have already investigated two ACGP techniques that allow such modifications. One technique observes the utility of specific local contexts when applied in mutation and crossover, and based on their utility (parent–offspring fitness

[1] Types and function overloading in v2 allows this context to be extended to the other siblings. This provides technology to extend the current first-order distribution.

[2] CGP expresses its heuristics by so called weighted mutation sets [3].

relationships) it increases or decreases the weights for the applied heuristics. A very simple implementation of this technique was shown to increase GP problem solving capabilities. However, mutation was much more problematic due to its bucket-brigade problem [4], and thus the overall improvements were marginal.

In here, we investigate a similarly simple technique, one that observes the distribution of functions and terminals in all/best trees (and thus the surviving distribution of all/best parent–child contexts). Note that we are using distribution to refer to the local context. Examples of such distributions are presented in Sec. 4. This idea is somehow similar to that used for CFG–based GP as recently reported in [8].

ACGP basic flowchart is illustrated in Fig. 1. ACGP works in iterations — *iteration* is a number of generations ending with extracting the distribution. The distribution information is collected and used to modify the actual mutation set weights (the heuristics). The modification can be gradual *(slope on)* or complete replacement *(slope off).* Then, the run continues, with the population undergoing the standard reproduction, or with a randomly regrown *(regrow on)* population. The regrowing option was found beneficial with longer iterations, where likely some material gets lost before being accounted for in the distributions, and thus needs to be reintroduced by regrowing the population (as reported in [5], regrowing is destructive for shorter iterations). Note that the newly regrown population is generated based on new (or updated) heuristics and thus may be vastly different from the first initial population — see Sec. 4 for illustrations.

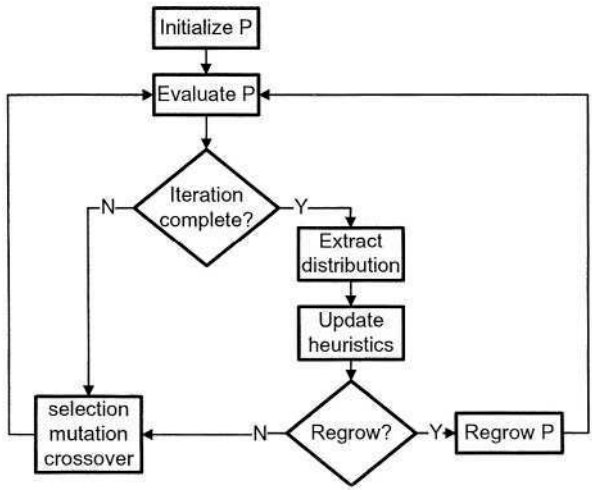


Fig. 1. The flowchart for the ACGP algorithm

ACGP can also work with simultaneous independent multiple populations, to improve its distribution statistics. ACGP can in fact correlate the populations by exchanging selected chromosomes — however, we have not tested such settings

yet. Moreover, at present all independent populations contribute to and use the same single set of heuristics — we have not experimented with maintaining separate heuristics, which likely would result in solving the problem in different subspaces by different populations.

Each population is ordered based on *2-key* sorting, which compares sizes (ascending) if two fitness values are relatively similar, and otherwise compares fitness (descending). The more relaxed the definition of relative fitness similarity, the more importance is placed on sizes.

Table 1. Examples of extracted distributions (partial matrix)

	f1	*f2*	*t1*	*t2*
Function *f1 arg1*	20	40	10	30
Function *f1 arg2*	10	10	80	0

Subsequently, the best *use* percent of the ordered chromosomes are selected into a common pool (from all populations) and resorted again. This pool of chromosomes is used to compute distribution statistics (from all or from *use* percent of the top chromosomes). The distribution is a *2-dim* matrix counting the frequency of parent-child appearances. Table 1 illustrates some extracted context distributions. Assume the function *f1* has 2 arguments (as shown), and there are 2 functions and two terminals in the user set $\{f1, f2, t1, t2\}$. This function (*f1*) appears 100 times in the selected set of trees (total for each row is 100). The cell $f1\ arg1[f1] = 20$ says that in 20 of the 100 cases the root of the first subtree is also labelled with *f1*. The 0 entry in the last cell indicates that the terminal *t2* never labels the second subtree of *f1* in the selected chromosome set.

4 Illustrative Experimental Results

To illustrate the concepts, we traced the local distributions in the population, measured fitness gains in subsequent iterations, and also attempted to visualize the extracted heuristics both quantitatively and qualitatively, as compared to those previously identified to be beneficial.

All reported experiments used 1000 trees per population, five populations, the standard mutation, crossover, and reproduction operators at the rate of 0.05, 0.85, and 0.1, and for the sake of sorting, trees with fitness values differing by no more than 2% of the fitness range values in the population were considered the same on fitness (and thus ordered ascending by size). Unless otherwise noted, all experiments report the average of the best of five populations, and they executed with *iteration* = 25 generations and *regrow on.*

4.1 Illustrative Problem: 11-multiplexer

To illustrate the behavior of ACGP, we selected the well-known 11-multiplexer problem [2]. This problem is not only well known and studied, but we also know from [3] which specific constraints improve the search efficiency. Our objective was to attempt to discover some of the same constraints automatically, and to observe how they change the search properties over multiple ACGP iterations.

The 11-multiplexer problem is to discover the Boolean function that passes the correct data bit (out of eight $d0 \ldots d7$) when controlled by three addresses ($a0 \ldots a2$). There are 2048 possible combinations. Koza [2] has proposed a set of four atomic functions to solve the problem: 3-argument *if/else,* 2-argument *and* and *or,* and 1-argument *not,* in addition to the data and address bits. This set is not only sufficient but also redundant. In [3] we have shown that operating under a sufficient set such as *{not, and}* degrades the performance, while operating with only *if* (sufficient by itself) and possibly *not* improves the performance. Moreover, we have shown that the performance is further enhanced when we restrict the *if*'s condition argument to choose only addresses, straight or negated, while restricting the two action arguments to select only data or recursive *if* [3]. This prior information is beneficial as we can compare ACGP–discovered heuristics with these previously identified and tested — as we will see, ACGP discovers virtually the same heuristics.

4.2 Change in Distribution of Local Heuristics

Here we traced the change in distribution of functions and terminals in the populations, just to visualize specific characteristics and behavior of the distribution. The distribution change is measured in terms of the local contexts, as explained in the previous section, and with respect to a reference distribution (the distribution in either the initial or in the previous generation). The distribution change is defined as the sum of squared differences between the two populations on individual frequencies (*e.g.,* normalized cells in Table 1). This change can be measured for individual functions, function–arguments, and even function–argument–values. However, unless indicated otherwise, the illustrations presented in the paper use all functions/terminals for the change measure.

Fig. 2 illustrates the change in the distribution of the local heuristics in the whole population, and also in just the best 4% of the population, when executed on the 11-multiplexer problem with *iteration*=25 and *slope on* (graduate change in heuristics). Fig. 2a uses initial population as the reference. As can be seen, the distribution grows with generations, but then diminishes upon regrowing the population at the end of each iteration. However, overall each new random population becomes more distant from the original initial population — an illustration of the fact the the heuristics change causes different distribution of labels in each regrown population. Moreover, the changes eventually saturate — as we will see shortly, this is because the heuristics have been extracted. Fig. 2b uses the previous population as the reference. As can be seen, each regrow dramatically changes the population (spikes) — however, the changes eventually

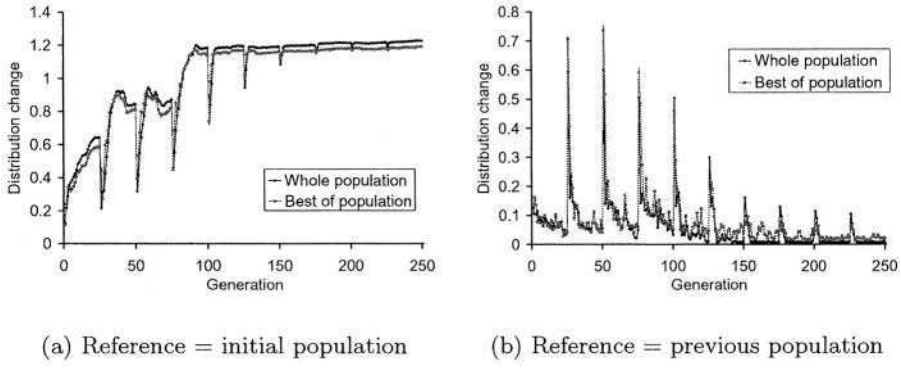


Fig. 2. Function/terminal distribution in the whole population and the best 20%

diminish, indicating (as see Fig. 3) that even the randomly regrown populations contain high quality chromosomes.

4.3 Change in the Speed of Evolution

The obvious question is what is the effect of the new heuristics on the learning curves. In this section, we answer the question in terms of average fitness growth, while in the next section we look at the extracted quantitative and qualitative heuristics.

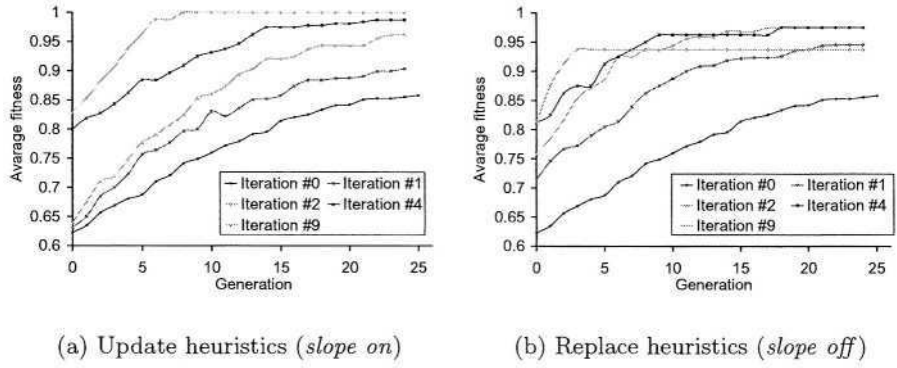


Fig. 3. Average fitness across five populations in selected iterations (with regrow)

We have conducted two experiments, with *iteration*=25 and total of 250 generations (10 iterations). In one experiment, we adjusted the heuristics *(slope on)*

while in the other we replaced the heuristics with those from the distribution *(slope off)*. Fig. 3 presents the results, shown separately for each of the 10 iterations. As seen, the average fitness grows much faster in subsequent iterations, indicating that ACGP did indeed extract helpful heuristics, which can be helpful in subsequent runs (in Fig. 8 we show that ACGP can also improve on the first run vs. standard GP). Moreover, the initial fitness in the initial random population of each iteration (regrow causes each new iteration to start with a new random population) also increases. Between the two, we can see that the second case *(slope off)* causes much faster learning but it is too greedy and indeed fails to solve the problem consistently (average saturation below 1.00 fitness). Inspection of the evolved heuristics revealed that some terminals in the evolved representation had very low weights, making it harder to consistently solve the problem in all populations even though it made it easier to "almost" solve the problem (in fact, the same experiment with smaller sampling completely dropped one terminal from the representation, making it impossible to solve the problem).

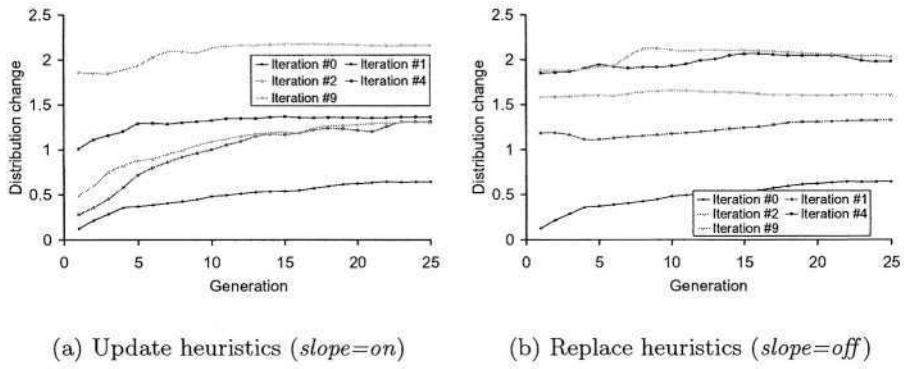
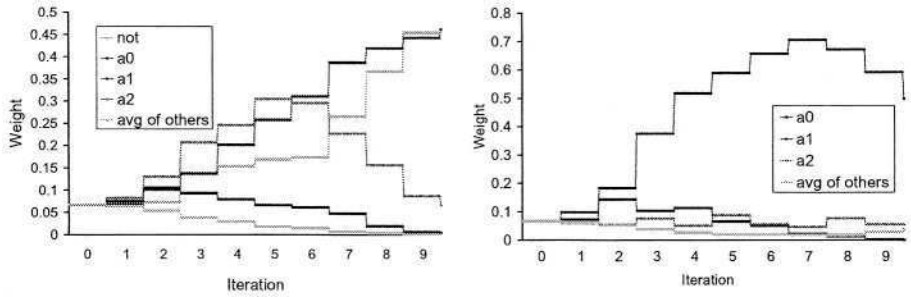


Fig. 4. Distribution changes in the whole population vs. the initial population, for each iteration separately

Finally, we also traced the same distribution change as in the previous section, for individual iterations. The results are presented in Fig. 4, which shows distribution changes in the whole population, with the initial population as the reference. They support the conclusions from Fig. 3 — *slope off* causes much faster changes in the distributions, but as seen before this can be too greedy. It will be important in the future to be able to predict the trade off between the pace of extraction of heuristics and and loss of representative power.

4.4 The Evolved Heuristics

In this section, we look at the evolved heuristics, attempting to qualify them and compare against those previously identified and tested for this problem. Recall that the best results were obtained with only the *if* function, the three addresses (directly or through *not*) in the condition subtree of *if,* and then recursive *if* and the data bits in the two action subtrees [3].



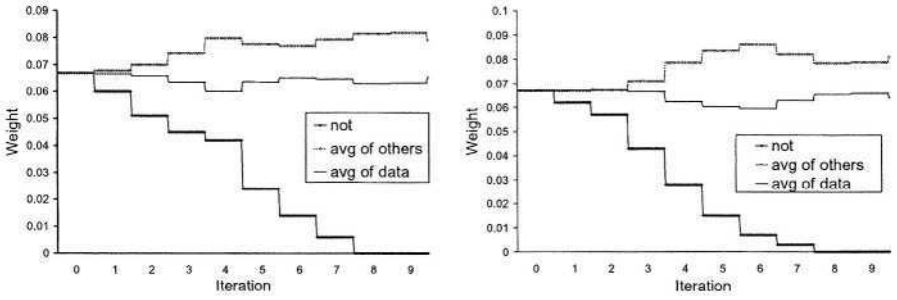
(a) Heuristics on function *if* test argument

(b) Heuristics on function *not*

Fig. 5. Evolved heuristics for *if,* the condition argument (direct and indirect through *not*

Fig. 5 illustrates the evolution of the heuristics on the condition part of *if.* All functions and terminals start equally (no prior heuristics on the first iteration). In the course of the ten iterations, we can observe that *and, or,* and the data bits are indeed dropped off (average shown). We can also see that the best heuristics indeed evolved around iteration six–seven (in fact the learning curves dropped off slightly after this point). This illustrates the need for other means to determine termination of the process. Fig. 5a illustrates the direct heuristics on *if* condition argument. As seen, *not* is highly present (to allow indirect addresses, see Fig. 5b). Among the address bits, *a0* is highly present, *a2* is marginally present, and *a1* is virtually absent. However, as seen in Fig. 5b, *a1* is very highly supported through indirect *not,* while *a2* has additional support as well.

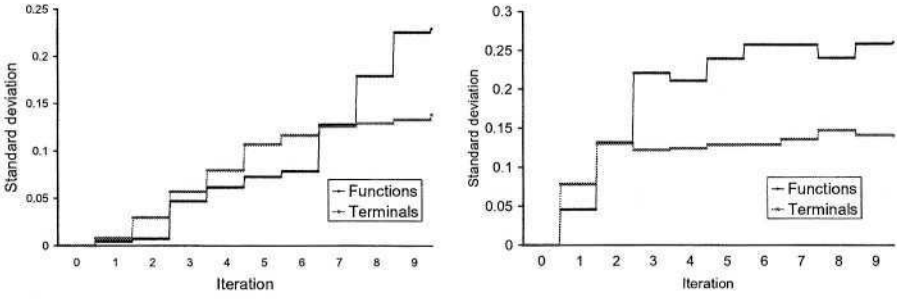
Fig. 6 illustrates the total heuristics on the two action parts of *if.* Recall that the action part in the best scenario should allow only data bits, and then recursive *if.* Moreover, *if* should have higher probability to allow deeper trees. We can see that this is indeed what has evolved. The function *if* spikes, and the data bits remain relatively stable - however because the remaining functions/terminals drop off, in fact the data bits are extracted. In the future, we plan to extend the heuristics not only to types but also to different levels - in this case we would hopefully observe that the *if*'s weight diminishes with node depth.



(a) Heuristics on function *if* first action

(b) Heuristics on function *if* second action

Fig. 6. Evolved heuristics for *if,* action arguments



(a) Update heuristics (*slope on*)

(b) Replace heuristics (*slope off*)

Fig. 7. Standard deviation on the function/terminal heuristics for function *if* test argument

Finally, we observe the changes in the heuristics of the *if*'s condition argument by tracing the standard deviation in its distribution of functions and terminals, separately for both. The results are illustrated in Fig. 7. Of course the deviations start very low on the first iteration (driven by the initially equal heuristics). At subsequent iterations, the weights become more diverse as the heuristics are learned, and they also change less (relative to the iteration), indicating again that heuristics extraction has saturated. Between the two, again slow updates (a) in heuristics led to slower changes, while drastic replacement of heuristics (b) led to faster changes and faster saturation. One more interesting observation is the sudden spike in terminals distribution for the update case (a), which coincides with the previously identified iteration where we observed the sudden shift in the heuristics (Fig. 5a).

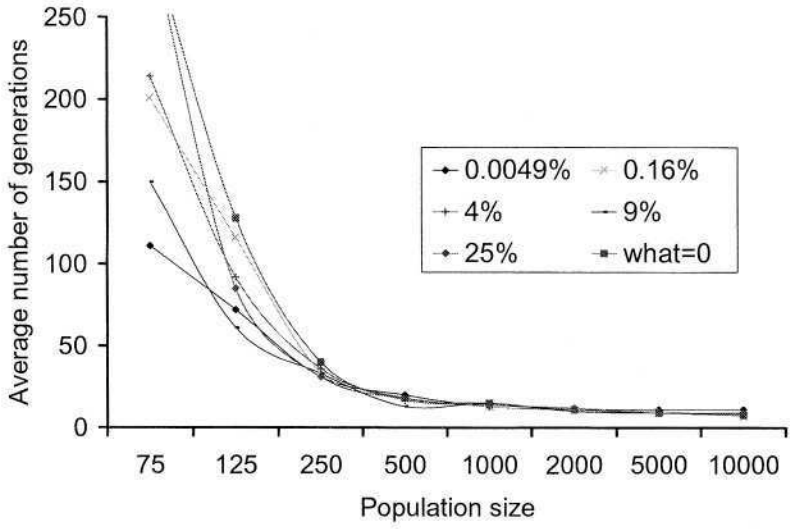


Fig. 8. The number of generations needed to solve for 80% — sampling rates for distributions are the effective rates (what=0 indicated a plain GP run)

4.5 Influence of Population Size, Sampling Rate, and Iteration Length

So far, we have reported experiments with long iterations (25 generations), population of 1000 trees (per population), and using *regrow* at the beginning of each new iteration. The results indicate that ACGP can extract meaningful heuristics, which helps in solving the problem fast and better in subsequent iterations. This is fine if the objective is to extract the heuristics or solve the problem with unlimited time. However, one may wish to make those improvements when solving the problem for the first time as well, while also extracting the heuristics. In this section, we relate population size needed to solve the problem, while attempting to learn and use the heuristics on the first execution of the system. More extensive results are reported in [5]. Fig. 8 presents accumulative result for *iteration*=1, various population sizes, and various effective sampling rates (effective rate refers to the percentage of trees eventually used for distribution with respect to all the trees in all the populations). As seen, ACGP works quite well with short iterations, and in fact beats GP especially for smaller populations and smaller sampling rates. It seems that ACGP allows smaller populations to solve the same problem faster. However, much more studies are needed here.

5 Conclusions

This paper presents the ACGP methodology for automatic extraction of heuristic constraints in genetic programming. It is based on the CGP technology,

which allows efficient processing of such constraints and heuristics. The ACGP algorithm presented here implements a technique based on distribution of local first-order (parent-child) contexts in the population. As illustrated, ACGP is able to extract such heuristics, which not only improve search capabilities but also have meaningful interpretation as compared to previously determined best heuristics for the same problem.

The paper also illustrates the changes in the distributions in the population, and raises a number of questions for the future.

- Extending the technique to the CGP `v2` technology, which allows overloaded functions, and thus extending the heuristics to the context of siblings.
- Linking population size with ACGP performance and problem complexity.
- Determining scalability of ACGP.
- Varying the effect of distribution and the heuristics at deeper tree levels.
- Exchanging chromosomes or separating heuristics between populations.
- Clustering techniques to explore "useful" high-order heuristics (more levels deep). This is similar to ADFs, except that ACGP would learn clusters of deep heuristics rather than abstract functions.
- The resulting trade-off between added capabilities and additional complexity when using deeper heuristics (CGP technology guarantees its low overhead only for the one–level constraints/heuristics).
- Other techniques for the heuristics, such as co–evolution between the heuristics and the solutions.

References

1. Banzhaf, W, et al.: Genetic Programming, and Introduction. Morgan Kaufmann (1998)
2. Koza, J. R.: Genetic Programming. On the Programming of Computers by Means of Natural Selection. Massachusetts Institute of Technology (1994)
3. Janikow, C.Z.: A Methodology for Processing Problem Constraints in Genetic Programming. Computers and Mathematics with Applications, Vol. 32, No. 8 (1996) 97–113
4. Janikow, C.Z. and Deshpande, R.A.: Evolving Representation in Genetic Programming. Proceedings of ANNIE'03 (2003) 45–50
5. Janikow, C.Z.: ACGP: Adaptable Constrained Geneting Programming. Proceedings of GPTP (2004) TBP
6. Montana, D. J.: Strongly Typed Genetic Programming. Evolutionary Computation, Vol. 3, No. 2 (1995)
7. Pelikan, M. and Goldberg, M.: BOA: The Bayesian Optimization Algorithm. Proceedings of GECCO'99 (1999) 525–532
8. Shan, Y., McKay, R. I., Abbass, H. A., and Essam, D.: Program Evolution with Explicit Learning: a New Framework for Program Automatic Synthesis. Technical Report. School of Com-puter Science, Univ. College, Univ. of New South Wales. Submitted Feb. 2003.
9. Whigham, P. A.: Grammatically-based Genetic Programming. In: J. P. Rosca (ed.): Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications (1995) 33–41

A Descriptive Encoding Language for Evolving Modular Neural Networks

Jae-Yoon Jung and James A. Reggia

Department of Computer Science, University of Maryland,
College Park, MD 20742, USA
{jung, reggia}@cs.umd.edu

Abstract. Evolutionary algorithms are a promising approach for the automated design of artificial neural networks, but they require a compact and efficient genetic encoding scheme to represent repetitive and recurrent modules in networks. Here we introduce a problem-independent approach based on a human-readable descriptive encoding using a high-level language. We show that this approach is useful in designing hierarchical structures and modular neural networks, and can be used to describe the search space as well as the final resultant networks.

1 Introduction and Background

Neuroevolution refers to the design of artificial neural networks using evolutionary computation methods. It involves searching through a space of weights and/or architectures without substantial human intervention, trying to obtain an optimal network for a given task. The evaluation (fitness) and improvement of each network is based on its overall behavior (not just on its performance, but also on other properties of the network), and this makes neuroevolution a promising method for solving complex problems involving reinforcement learning [5,20] or designing recurrent networks [14].

An important research issue in neuroevolution is determining how to best encode possible solution networks as developmental "programs" forming the genotype in a compact and efficient manner. In early work in this field, Kitano encoded a set of developmental rules as a *graph generation grammar* having the form of a matrix [9,10]. Such an approach is developmental in nature because the information (e.g., a set of ordered rules and parameters) stored in the genotype describes a way of "growing" the phenotype. Constructing a network connectivity matrix begins with the initial start symbol in a chromosome, and rules are then applied to replace each non-terminal symbol in the chromosome with a 2x2 matrix of symbols, until there are all terminal symbols. Other well known but somewhat different approaches include Gruau's *cellular encoding,* where each rule defines a transformation or modification of a cell and the rules constitute a tree structure such that the order of execution for each rule is specified [3,4]; edge encoding [15]; and geometry-based cellular encoding [11,12].

Also of relevance to the work described here is past research that has proposed layer-based, parametric encoding schemes in which the basic unit of network

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 519–530, 2004.

architecture is a *set* of nodes (i.e., a layer) and network properties such as layer size and learning rates are encoded in the chromosomes as parameters that are altered during the evolutionary process [6,7,16,19].

Many of the encoding schemes above are very difficult to apply for designing large networks due to the scalability problem and their procedural nature, and are based on specific problem-dependent assumptions (e.g., the number of hidden nodes is manually selected [14]). Here we present an encoding scheme which appears to address these limitations. A layer-based, hierarchical architecture representation in our approach enables a high-level specification of multi-modular networks, and we let users incorporate their domain knowledge and restrictions on the types of networks evolved by explicitly choosing an appropriate set of network properties and their legal values. Thus our system does not a priori restrict whether architecture, layer sizes, learning method, etc. form the focus of evolution as many previous approaches have done, but instead allows the user to select which aspects of networks are to evolve. Furthermore, our approach is analogous to the abstraction process used in contemporary programming, in the sense that users write a text file specifying the problem to solve using a given high-level language. Our system then parses this file and searches for a solution within the designated search space, and finally it produces the results as another human readable text file. Thus we believe that our approach facilitates automated design of large scale neural networks covering a wide range of problem domains, not only because of its encoding efficiency, but also because it increases human readability and understandability of the initial environment specification and the final resultant networks.

2 Descriptive Encoding Methodology

Our encoding scheme is an extension of both the grammatical and the parametric encoding methods described above. Modular, hierarchical structure is essential when the size of the resultant neural network is expected to be large, since monolithic networks can behave irregularly as the network size becomes larger. Moreover, there is substantial evidence that a basic underlying neurobiological unit of cognitive function is a region (layer), e.g., in cerebral cortex [8], which strengthens the argument that hierarchical structure of layers should be the base architecture of any functional unit. Parametric encoding can also reduce the complexity of a genotype when there is a regular pattern in the network features, and opens the possibility for users to specify a set of appropriate network features according to given problem requirements.

We refer to our approach as a *descriptive encoding* since it enables users to "describe" the target space of neural networks to be considered in a natural, non-procedural human readable format. A user writes a text file like the ones later in this paper to specify sets of layers with appropriate properties, their legal evolvable values, and inter-layer connectivity. This input description does *not* specify individual connections, nor their weights. The specification of legal values affects the range of valid genetic operations. In other words, a description file

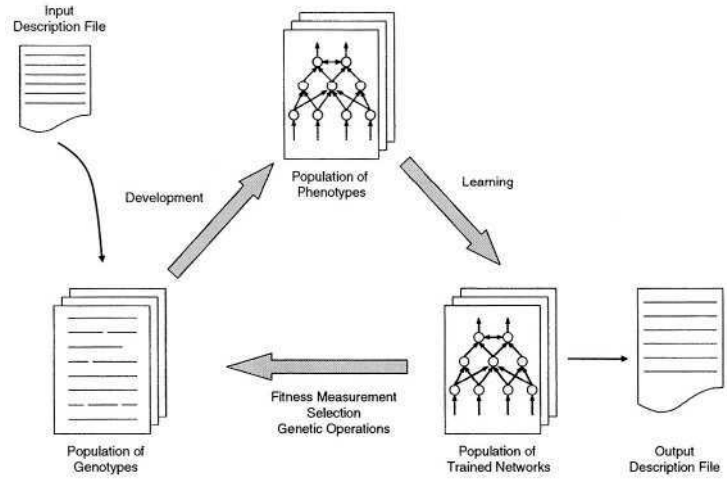


Fig. 1. The development, learning, and evolution procedure used in our system. The input description file (upper left) is a human-written specification of the class of neural networks to be evolved (the space to be searched) by the evolutionary process; the output description file (lower right) is a human-readable specification of the best specific networks obtained.

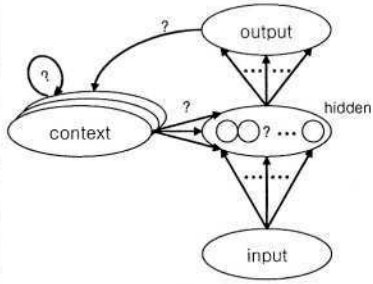
specifies the initial population and environment variables, and confines the search space of genetic operators throughout the evolutionary process. The evolutionary process in our system involves an initialization step plus a repeated cycle of three stages, as in Figure 1. First, the text description file prepared by the user is parsed and an initial random population of chromosomes (genotypes) is created within the search space represented by the description (left part of Figure 1). In the Development stage, a new population of realized networks (phenotypes) is created or "grown" from the genotype population. Each phenotype network keeps actual and specific nodes, connection weights, and biases. The Learning stage involves training each phenotype network if the user specifies one or more learning rules in the description file, making use of an input/output pattern file. Evolutionary computation is often less effective for local, fine tuning tasks [21], so we adopt neural network training methods. After the training stage, each individual network is evaluated according to user-defined fitness criteria and genetic operators are applied to the genotypes. Currently, fitness criteria may reflect both network performance (e.g., mean squared error) and a penalty for a large network (e.g., total number of nodes), or other measures.

2.1 Explicitly Incorporating Domain Knowledge

When searching for an optimal neural network using evolutionary computation methods, a network designer usually wants to restrict the architecture, learning rules, etc. to some proper subset of all possible models. Thus, many problem

```
[SEQUENCE
  [LAYER input  [SIZE 45] [CONNECT hidden]
                [LEARN_RULE [EVOLVE bp rprop]]]
  [LAYER hidden [SIZE [EVOLVE 1 45]] [CONNECT output]
                [LEARN_RULE [EVOLVE bp rprop]]]
  [LAYER output [SIZE 45] [CONNECT [EVOLVE context]]
                [CONNECT_RADIUS 1]]
  [SEQUENCE context
    [LAYER con [NUMBER  [EVOLVE 1 5]] [SIZE 45]
               [CONNECT [EVOLVE con hidden]]]]
]
```

(a)



(b)

Fig. 2. (a) Part of a description file; other information, such as details of the evolutionary process, is not shown, (b) An illustration of the corresponding class of recurrent networks that are described in (a). In effect, the description file specifies the search space to be used by the evolutionary process, while simultaneously providing a tree structure to be used by genetic operators such as crossover and mutation.

specific constraints need to be applied in creating the initial population and maintaining it within a restricted subspace of the space of all neural networks. For example, the range of architectures and valid property values for each individual network in the initial population will depend upon the specific problem being addressed. While such constraints and initialization procedures have been treated implicitly in previous approaches, our encoding scheme permits them to be described in a compact and explicit manner.

Figure 2 illustrates an example description written in our language for evolving a recurrent network, motivated by [17]. Each semantic block, enclosed in brackets [···], starts with a type identifier followed by an optional name and a list of properties (a much simplified grammar for part of our encoding language is given in the Appendix to make this more precise). A network contains other (sub)networks and/or layers recursively, and a network type identifier (SEQUENCE, PARALLEL, or COLLECTION) indicates the arrangement of the subnetworks contained in this network. We define a layer as a set (sometimes one or two dimensional, depending on the problem) of nodes that share the same properties, and it is the basic module of our network representation scheme. For example, the description in Figure 2a indicates that a sequence of four types of layers are to be used: input, hidden, output, and con layers. Properties fill in the details of the network architecture (e.g., layer size and connectivity) and specify other network features including learning rules and activation dynamics.

Most previous neuroevolution research has focused a priori on some limited number of network features (e.g., network weights, number of nodes in the hidden layer) assuming that the other features are fixed, and this situation has prevented neuroevolution models developed by researchers from being used more widely in different environments. To overcome this limitation, we let users decide which properties are necessary to solve their problems, and what factors should

be evolved, from a set of supported properties that include architectures, activation dynamics, and learning rules. Unspecified properties may be replaced with default values and are treated as being fixed after initialization. For example, in Figure 2a, the input layer has a fixed number of 45 nodes and is connected to the hidden layer, while the single hidden layer has a SIZE within the range 1 to 45. The EVOLVE attribute indicates that the hidden layer's size will be randomly selected initially and is to be modified within the specified range during the evolution process. Note that the learning rules to be used for connections originating from both input and hidden layers are also declared as an evolvable property. After processing input patterns from the hidden layer, the output layer propagates its output to the layers in the context network, and the CONNECT_RADIUS property defines one-to-one connectivity in this case. Since the number of layers in the context network may vary from 1 to 5 (i.e., LAYER con has an evolvable NUMBER property), this output connectivity can be linked to any of these layers that were selected in a random manner during the evolution process. Finally, the layers in the context network are arranged as a sequence, and are connected to the hidden layer or themselves. Figure 2b depicts the corresponding search space schematically for the description of Figure 2a, and the details of each individual genotype (shown as question marks in the picture) will be assigned within this space at the initialization step and forced to remain within this space during the evolution process. Since the genotype structure is a tree, genetic operators used in GP [13] can be easily applied to them.

3 Results with the Parallel Multiple XOR Problem

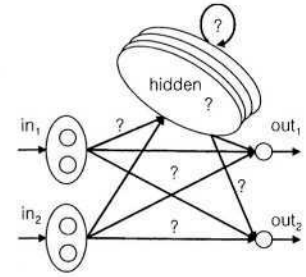
We use a parallel multiple exclusive-or (XOR) problem that currently runs on our system to illustrate the transformation of a description file into the results of an evolutionary process. The parallel multiple XOR problem is defined by a training data set where the correct result of each output node is the XOR of a corresponding separate pair of input nodes, independent of the values of the other output nodes. Of course, the evolutionary process has no a priori knowledge that this is the desired behavior; it is only evident from the training data. Further, this problem is difficult in the sense that there are apparent partial relationships between unrelated input and output nodes that must be identified without a priori domain knowledge. Also, since individual XOR problems are not linearly separable, the evolutionary process must discover that at least one hidden layer is necessary. A minimal network with no hidden layer cannot be the solution, although it is legal.

3.1 Encoding Details

The description file in Figure 3a defines a parallel dual-XOR problem with four input nodes, where layers in_1 and out_1 form one XOR gate, while layer in_2 is paired with out_2 to form another one. This description specifies both the initial networks to be created and the search space. It indicates that the desired

```
[SEQUENCE dual_xor
  [PARALLEL input
    [LAYER in  [NUMBER 2][SIZE 2]
               [CONNECT [EVOLVE hidden output]]]]
  [COLLECTION hidden
    [LAYER hid [NUMBER  [EVOLVE 0 10]]
               [SIZE    [EVOLVE 1  5]]
               [CONNECT [EVOLVE hidden output]]]]
  [PARALLEL output
    [LAYER out [NUMBER 2][SIZE 1]]]]
```

(a)



(b)

Fig. 3. Initial description file (a) and sketch of the space of networks to be searched (b) for a simple parallel dual-XOR problem with four input nodes and two output nodes.

overall structure is a sequential network named dual_XOR consisting of two input layers in parallel, a set (or "COLLECTION") of zero to 10 hidden layers, and two single-node output layers. The NUMBER statements assign the range of how many layers of each type may be created with the same properties in the network. So the description for the input layer is equivalent (except for optional layer names) to specifying this:

```
[LAYER in1 [SIZE 2] [CONNECT [EVOLVE hidden output]] ]
[LAYER in2 [SIZE 2] [CONNECT [EVOLVE hidden output]] ]
```

The CONNECT property in the input layers descriptor indicates that nodes in each input layer may evolve to connect to hidden layers, output layers, neither or both. The COLLECTION description indicates that networks can evolve to have 0 to 10 hidden layers, each with 1 to 5 nodes, and that they can be connected arbitrarily to themselves and to the output layers. The EVOLVE attributes listed here indicate that the connections from input layers to hidden and/or output layers, the number and size of hidden layers, and the connections from hidden layers to other hidden layers and output layers, are all evolvable. These combinations are randomly and independently decided at the initialization step and enforced by genetic operators throughout the evolution process.

Each chromosome created from this description stores a representation of an architecture in the form of a tree, as well as other network features as embedded parameters (properties) in the tree. This hierarchical description of the network architecture has some benefits over a linear list of layers in previous layer-based encoding schemes, since it directly maps the topology of a network into the representation. These benefits are: 1) it enables crossover operation on a set of topologically neighboring layers, which was not possible with point crossover operators; 2) functionally separated blocks can be easily specified and identified in a large scale, multi modular network; and 3) reusable subnetworks can be defined to address the scalability problem (e.g., like ADFs in GP [13]). Figure 4a and b illustrate two example networks automatically generated from the description

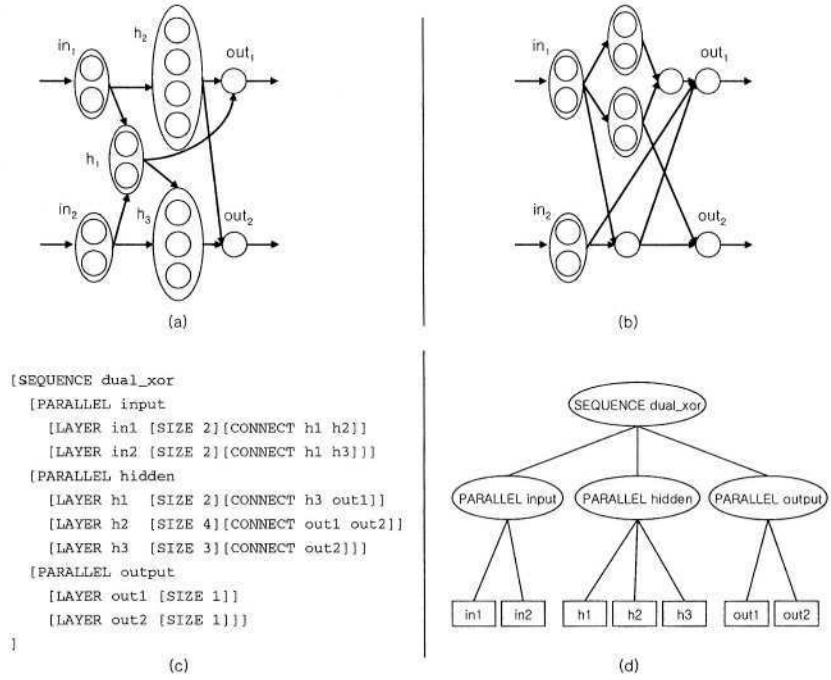


Fig. 4. (a),(b) Examples of neural network architectures randomly created from the description during initialization. Input and output layers are the same, but the number of hidden layers and their connections are quite different and specific now. Arrows indicate sets of connections between layers (i.e., not individual node-to-node connections). (c) The chromosome description of the network illustrated in (a), as it would be written in our descriptive language. This is *not* a description file written by the user, but is automatically generated from that description file. Note that no EVOLVE attributes are present, for example. (d) Top part of the tree-like structure of the genotype in (c), making it directly usable by GP operators. Each rectangle designates a layer.

file of Figure 3a; they show different numbers of layers and topologies. Figure 4c shows the corresponding chromosome or genotype structure of one of these, the network in Figure 4a. Note that the overall structure is the same with the initial description in Figure 3a, but the COLLECTION hidden network has been replaced with a PARALLEL network with three layers and each property has a fixed value (i.e., the EVOLVE attributes are gone). Figure 4d shows the tree like structure of this genotype, making it amenable to standard GP operators.

3.2 The Evolutionary Procedure

The description file created by the user not only specifies the search space, but also sets a variety of parameter values that influence the evolutionary process. Figure 5 shows the description for the learning and evolutionary parameters used to solve the parallel dual XOR problem. This is also a part of the description

```
[TRAINING
  [TRAIN_DATA "./inout_pattern.txt"] [MAX_TRAIN 100]]
[EVOLUTION
  [FITNESS weighted_sum] [ALPHA 0.5] [BETA 0.2] [GAMMA 0.2]
  [SELECTION tournament] [TOURNAMENT_POOL 3] [ELITISM 0]
  [MUTATION_PROB 0.7]    [CROSSOVER_PROB 0.0]
  [MAX_GENERATION 50]    [MAX_POPULATION 50]
]
```

Fig. 5. Training and evolutionary parameters for a parallel XOR problem.

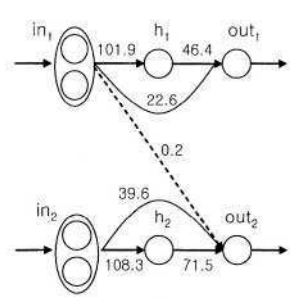
file following the network description, providing users with a systematic way to control the training and evolutionary procedure. In the Learning stage, each phenotype network is trained for 100 epochs with a default (as it is not specified in the description) backpropagation algorithm (RPROP [18]) plus the input/output pattern file specified in the TRAIN_DATA property (recurrent connections are deleted after the mutation operation). Currently, the fitness value of each phenotype network is calculated as a weighted sum of three reciprocally normalized criteria: mean squared error (MSE, e), total number of network nodes (n), and total number of layer-to-layer connections (c) (ultimately, we wish to allow the user to specify more general fitness functions). These three criteria are weighted with coefficients α, β, and γ which the user assigns (Figure 5). MSE reflects the output performance of the network, and the other two measures are adopted as a penalty for larger networks. Currently connections are counted on a layer-to-layer basis, not on a node-to-node basis, since the latter may be correlated or proportional to the total number of nodes. More specifically, the fitness value of a network is:

$$fit = \alpha \cdot \left(\frac{e_{max} - e}{e_{max} - e_{min}} \right) + \beta \cdot \left(\frac{n_{max} - n}{n_{max} - n_{min}} \right) + \gamma \cdot \left(\frac{c_{max} - c}{c_{max} - c_{min}} \right) \quad (1)$$

where $x_{min}(x_{max})$ denotes the minimum (maximum) value of criterion x among the population. Although this simple weighted sum was sufficient for the XOR problem, other multi-objective optimization methods [2] can be easily applied. Tournament selection with tournament size 3 is specified by the user for selecting candidate individuals for the next generation. Selection is based on the trained network fitness values, while genetic operations are done with genotype trees. The fitness value of each network is compared with that of three other randomly selected networks. Then one fittest network is selected and the corresponding chromosome (genotype) is copied to the next generation. No individuals other than the tournament winners are inserted into the new population and no elitism is used. The mutation rate is 0.7 and no crossover is used. Operators can mutate layer size, direction of an existing inter-layer connection, and can add or delete a new layer or connection. The initial description file is implicitly used here to constrain the resultant network population to remain in the search space

```
[SEQUENCE dual_xor
  [PARALLEL input
    [LAYER in1 [SIZE 2][CONNECT [h1 out1 out2]]
    [LAYER in2 [SIZE 2][CONNECT [h2 out2]]]
  [PARALLEL hidden
    [LAYER h1  [SIZE 1][CONNECT out1]]
    [LAYER h2  [SIZE 1][CONNECT out2]]]
  [PARALLEL output
    [LAYER out1 [SIZE 1]]
    [LAYER out2 [SIZE 1]]]]
```

(a)



(b)

Fig. 6. (a) A typical separate channel network evolved for the dual XOR problem. Each input layer uses its own hidden node (separate layer), and a direct connection to the correct matching output node. (b) The average value of absolute weights on connections between each layer is specified. A dotted line between $\mathbf{in_1}$ and $\mathbf{out_2}$ shows persistent, non-optimal connectivity with very small weights.

specified by the user; i.e., mutation only occurs within the legal range specified in the description file. After these genetic operations, a new genotype population is generated and starts another cycle. The total number of generations in a simulation and population size are both 50.

3.3 Results of the Evolutionary Process

Given the above information, our system generated near-optimal networks that both solved the dual XOR problem and had a small number of nodes. One of the best, but commonly found, resultant networks that correctly solve the problem is depicted in Figure 6a, when the ratio of $\alpha = 0.5$, $\beta = 0.2$, and $\gamma = 0.2$ was used. Note that this text description is a part of the output file which is generated by the system, and still a legal description in our language. Since it is a specific network, no EVOLVE attributes are present. The final resultant output also contains various statistical data and network details including trained connection weights, as summarized in Figure 6b. This result clearly demonstrates that our system can identify the partial, internal relationships between input and output patterns present in the dual XOR problem and represent them within a modular architecture, without a priori information about this in the initial network description. Ignoring the dotted line connections which have a near zero weight values shown in Figure 6b, we can easily see that this is a near-optimal network for the dual XOR problem in terms of the number of network nodes and connections.

4 Encoding Properties

Our approach has some important encoding properties. It can represent recurrent network architectures, and is scalable with respect to node/connectivity changes. More specifically, the encoding approach we are using has:

- *Closure :* A representation scheme can be said to be closed if all genotypes produced are mapped into a valid set of phenotype networks [1]. First, every genotype at the initial step is decoded into a valid phenotype since the initial population of genotypes is based on the user-defined description. Next, our approach is closed with respect to mutation operators that change property values in a layer, since we only allow property values to be mutated within the legal ranges defined by users or the system. This is checked at runtime and any illegal mutation result is discarded with replacement by another mutation to keep the population size fixed. Although our encoding scheme is not closed with crossover operators on a grammar level, it can be constrained to be closed on a system level by adjusting invalid property values, according to the description file. For example, if the number of layers in a network becomes too large after a crossover operation, such a network may be deleted (or the whole network structure could be adjusted to maintain legal genotypes).
- *Completeness :* Our encoding scheme can be used to represent any recurrent neural network architecture. This can be easily seen from the fact that if we confine the size of each and every layer to be one node, our encoding scheme is equivalent to a direct encoding which specifies full connectivity on a node-to-node basis. Combined with the closure property, this property ensures that our encoding can safely replace the direct encoding or the equivalent encodings.
- *Scalability:* This property can be defined by how decoding time and genotype space complexity are affected by a single change in a phenotype [1]. Our encoding scheme takes $O(1)$ time and space in a node addition/deletion, since changing the number of nodes means just changing a parameter value in a property in the corresponding genotype, and node addition/deletion does not make substantial changes in time and space requirements during the genotype-phenotype mapping. In a similar way, a node-to-node connection addition/deletion in a phenotype will cost $O(1)$ space in genotype and $O(N+C)$ decoding time, as N denotes the total number of nodes in a network, and C denotes the total number of layer-to-layer connections. If a connection is deleted in a phenotype, it will split the corresponding source and target layers since nodes in these layers do not share connectivity anymore, but this split is equivalent to deleting a node in both layers plus creating two single-node layers, which will cost $O(1)$ space (assuming a constant layer size) and $O(N + C)$ additional decoding time. In general, our scheme is $O(1)$ scalable with respect to nodes and $O(N + C)$ scalable with respect to connectivity.

5 Discussion

In this paper we have introduced a problem-independent evolving neural network system based on an encoding scheme and a high-level description language. Our approach can efficiently represent the hierarchical structure of multi-layer neural networks, which is a desired property for designing large scale networks. The tree structured encoding scheme used in our approach is amenable to GP operators, and we have shown that it is scalable and can be mapped into a valid set of recurrent phenotype networks. We have demonstrated with the parallel XOR problem that our system can identify relationships between input and output patterns and incorporate them into an optimal architecture. The use of a description file provides users with a systematic, non-procedural methodology for specifying the search space and evolution parameters, and the same language used for the network description can be used to produce a human readable final network description.

References

1. K. Balakrishnan and V. Honavar, Properties of genetic representations of neural architectures, *Proc. of the World Congress on Neural Networks,* pp. 807-813, 1995.
2. C. Coello, Evolutionary multi-objective optimization: a critical review, *Evolutionary Optimization,* R. Sarkar et al. (eds.), Kluwer, pp. 117-146, 2002.
3. F. Gruau, Neural network synthesis using cellular encoding and the genetic algorithm, *Ph.D. Thesis,* Laboratoire de l'Informatique du Parallilisme, Ecole Normale Supirieure de Lyon, France, 1994.
4. F. Gruau, Automatic definition of modular neural networks, *Adaptive Behavior,* 3:151-183, 1995.
5. F. Gruau, D. Whitley, and L. Pyeatt, A comparison between cellular encoding and direct encoding for genetic neural networks, *Genetic Programming 1996: Proc. of the First Annual Conference,* MIT Press, pp. 81-89, 1996.
6. S. A. Harp, T. Samad, and A. Guha, Toward the genetic synthesis of neural networks, *Proc. of 3rd International Conference on Genetic Algorithms,* Morgan Kaufmann, pp. 379-384, 1989.
7. T. S. Hussain and R. A. Browse, Network generating attribute grammar encoding, *Proc. of International Joint Conference on Neural Networks (IJCNN '98),* 1:431-436, 1998.
8. E. R. Kandel and J. H. Schwartz, *Principles of Neural Science,* Elsevier, 1983.
9. H. Kitano, Designing neural networks using genetic algorithms with graph generation system, *Complex Systems,* 4:461-476, 1990.
10. H. Kitano, Neurogenetic learning: an integrated method of designing and training neural networks using genetic algorithms. *Physica D,* 75:225-238, 1994.
11. J. Kodjabachian and J. A. Meyer, Evolution and development of control architectures in animats, *Robotics and Autonomous Systems,* 16:161-182, 1995.
12. J. Kodjabachian and J. A. Meyer, Evolution and development of modular control architectures for 1-D locomotion in six-legged animats, *Connection Science,* 10:211-254, 1998.
13. J. R. Koza, *Genetic Programming II,* MIT Press, 1994.

14. K. W. C. Ku, M. W. Mak, and W. C. Siu, Adding learning to cellular genetic algorithms for training recurrent neural networks, *IEEE Trans. on Neural Networks,* 10(2):239-252, 1999.
15. S. Luke and L. Spector, Evolving graphs and networks with edge encoding: preliminary report, *Late Breaking Papers at the Genetic Programming 1996 Conference,* pp. 117-124, 1996.
16. M. Mandischer, Representation and evolution of neural networks, *Artificial Neural Nets and Genetic Algorithms,* R. F. Albrecht, C. R. Reeves, and N. C. Steele (eds.), Springer, pp. 643-649, 1993.
17. M. J. Radio, J. A. Reggia, and R. S. Berndt, Learning word pronunciations using a recurrent neural network, *Proc. of the International Joint Conference on Neural Networks (IJCNN-01),* 1:11-15, 2001.
18. M. Riedmiller and H. Braun, A direct adaptive method for faster backpropagation learning: the RPROP algorithm, *Proc. of the IEEE International Conference on Neural Networks,* 1:586-591, 1993.
19. W. Schiffmann, Encoding feedforward networks for topology optimization by simulated evolution, *Proc. of 4th International Conference on Knowledge-Based Intelligent Engineering Systems and Allied Technologies (KES 2000),* 1:361-364, 2000.
20. K. O. Stanley and R. Miikkulainen, Efficient reinforcement learning through evolving neural network topologies, *Proc. of the Genetic and Evolutionary Computation Conference (GECCO-2002),* Morgan Kaufmann, pp. 569-577, 2002.
21. X. Yao, Evolving artificial neural networks, *Proc. of the IEEE,* 87(9): 1423-1447, 1999.

Appendix – A Simplified and Partial Encoding Grammar

```
<description>  := <network> <training> <evolution>
<network>      := [<net_type> <name> <sub_network>] | <layer>
<training>     := [TRAINING  <tr_prop_list>]
<evolution>    := [EVOLUTION <ev_prop_list>]
<net_type>     := SEQUENCE | PARALLEL | COLLECTION
<sub_network>  := <network> <sub_network> | <network>
<layer>        := [LAYER <name> <ly_prop_list>]
<ly_prop_list> := <ly_property> <ly_prop_list> | <ly_property>
<tr_prop_list> := <tr_property> <tr_prop_list> | <tr_property>
<ev_prop_list> := <ev_property> <ev_prop_list> | <ev_property>
<ly_property>  := [NUMBER     <value>] | [SIZE      <value>] | ...
<tr_property>  := [TRAIN_DATA <path> ] | [MAX_TRAIN <value>] | ...
<ev_property>  := [FITNESS    <name> ] | [SELECTION <name> ] | ...
<value>        := [EVOLVE <range_value>] | [<range_value>] |
                  [EVOLVE <fixed_value>] |  <fixed_value>
<range_value>  := <fixed_value> <range_value> | <fixed_value>
<fixed_value>  := <integer>  | <float> | <literals>
<name>         := <literals>
<path>         := ``<name>''
```

Run Transferable Libraries — Learning Functional Bias in Problem Domains

Maarten Keijzer[1], Conor Ryan[2], and Mike Cattolico[3]

[1] Prognosys, Utrecht, The Netherlands mkeijzer@xs4all.nl
[2] University of Limerick, Ireland conor.ryan@ul.ie
[3] Tiger Mountain Scientific Inc., Kirkland, WA USA mike@tigerscience.com

Abstract. This paper introduces the notion of Run Transferable Libraries, a mechanism to pass knowledge acquired in one GP run to another. We demonstrate that a system using these libraries can solve a selection of standard benchmarks considerably more quickly than GP with ADFs by building knowledge about a problem. Further, we demonstrate that a GP system with these libraries can scale much better than a standard ADF GP system when trained initially on simpler versions of difficult problems.

1 Introduction

This paper introduces the concept of Run Transferable Libraries (RTLs) that overturns the traditional notion of independent runs in GP. Instead of executing a large number of independent runs, in each of which GP attempts to solve a problem or learn a target function from scratch, a system using RTL accumulates information from run to run, learning more about the problem area each time it gets applied. The method is partly inspired by the use of libraries in conventional programming, where they are used to capture common functionality for a certain *domain* of functionality. Similarly, an RTL enabled search method uses libraries that provide functionality that is valid in *problem domains.* However, unlike standard programming libraries which tend to be static, RTL enabled search methods evolve their libraries over time and update them based on past experience. It is thought that by using Run Transferable Libraries it is possible for a search strategy such as GP to become *competent* in solving more difficult problems than can be tackled currently.

This approach can have major implications for the scalability of a system. Not only is it possible for an RTL to learn from run to run on a single problem instance, becoming more adept at solving the problem at hand, it can also be trained on "simple" problem instances, in order to tackle more difficult problems later on. A third use of RTLs is to train a library to become competent in a problem domain where initial runs are unsuccessful, tackling same complexity problems with more and more success as the RTL enabled system becomes more experienced. In effect, by training a library on different problem instances, we hope to learn the underlying bias that makes the set of problems a coherent

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 531–542, 2004.

whole (a problem domain). The bias that an RTL method tries to learn is of a functional nature, a set of functions that help in solving related problems faster. It is hypothesised that to do this an RTL approach needs to have:

- the ability to learn to tackle a difficult problem by using information gathered from previous, possibly unsuccessful attempts;
- the ability to apply information learned from a simple instance of a problem set to a more difficult instance;
- the ability to learn how to efficiently solve problems taken from the same problem domain.

We demonstrate that the current implementation of RTL, our system *Mnemosyne*[1], can, even when applied to a *single* instance of problems such as 4- and 5- parity and a small version of the Lawnmower problem, learn about the problem so quickly that it outperforms GP (with ADFs enabled) in terms of computational effort, especially on larger versions of the problem. It is also demonstrated that we can *unbias* an initial function set used for logical functions by training on randomly generated instances of logical functions. When applying a library trained in such a way to the parity problem, it shows a decrease in computational effort, albeit not so large as when the library is trained on parity problems themselves.

2 Scaling and Problem Domains in GP

Traditionally, AI has been divided into two fundamentally different approaches, *strong* and *weak*. Strong AI usually involves capturing as much prior knowledge as possible about a problem before attempting to solve it, while Weak AI is concerned with the application of more general methods that can be used across a wide spread of problems. This is possible because Weak AI methods do not require prior knowledge of a problem.

GP (and Evolutionary Computation in general) has been described as a *strong-weak* method [2] because of the way in which populations capture pertinent information about a problem and recombine it to produce more useful solutions. The typical dynamic for an EC run is that the population starts out with very poor performing individuals, and slowly accumulates knowledge about the problem until a satisfactory solution is found.

Most applications of EC involve independent runs; that is, all the knowledge accumulated by each run is discarded once it terminates. Furthermore, when one attempts to apply GP to a more difficult problem, even a more difficult instance of a problem already solved, all the effort spent in previous runs is lost, and the system must start from scratch again, which has lead to concerns about the scaling abilities of GP.

Another approach to the scaling issue is to exploit the inherent modularity present in many problems, through the use of Automatically Defined Functions

[1] Mnemosyne is the mother of the Muses. She is the personification of Memory.

(ADFs) [5] or some similar structures that permit GP to decompose the problem. The following are examples are some of the more successful attempts to improve the scaling abilities of GP using one of these methods.

ADFs. Automatically Defined Functions were introduced by Koza [5] ostensibly as a method for decomposing problems. ADFs are essentially local functions that individuals can call, which are also subject to evolution.

Prior to running ADF GP, one typically chooses the number of ADFs each individual has, as well as their arity. Thus, each individual is made up of a *result producing branch* (RPB) which is essentially a main program, and one or more ADFs, each of which can be called by the RPBs.

Koza showed that, on decomposable problems, using ADFs appropriate for the problem significantly improved the performance of GP. Indeed, while the choice of number and structure of the ADFs effects the performance of the system, Koza showed that, on these problems, using *any kind* of ADF yielded better performance than without. Furthermore, it is possible to evolve the number and arity of ADFs, although this can lead to considerably longer evolution times.

GLib. Another approach to the exploitation of modularity is Angeline's [1] Genetic Library Builder (GLiB). Unlike ADFs, GLiB relies upon *subroutines* as its modularity mechanism. Subroutines in GLiB are referred to as *modules.*

Modules are created dynamically during a run, through the use of a *compression* mutation operator. Compression is probabilistically applied to a sub-tree of an individual, and compresses it into a module, which is a node with the same functionality. This has the dual effect of increasing the expressiveness of the language and decreasing the average size of individuals in the population.

A newly created module can then be passed onto later generations in the same way any other node is. The extent to which it is used by descendants depends on a combination of the utility of the module and individuals' successful use of it. Unlike ADFs, however, due to the extraction process, GLiB modules cannot reuse arguments in the modules. It has been argued that this impedes performance in GLiB [4].

GLiB also has an *expansion* mutation operator, which expands a previously compressed node. This was introduced to combat the loss of diversity that the system experienced due to the presence of the modules.

ARL. The Adaptive Representation through Learning(ARL) [8] system is a combination of the GLiB and ADFs. Unlike GLiB, ARL selected individuals to compress based on how well their fitness improved upon that of their parents.

ARL was shown to perform well relative to the other methods, but has been criticised [7] for requiring many control parameters, and has yet to find wide acceptance in the field.

Subtree Encapsulation. Each of the above schemes is concerned with producing modules and functions in parallel with the individuals that will use them. Once a run is finished, all the effort involved in producing these modules must be repeated for the next run. A different approach was taken by [7] with their *Subtree Encapsulation* method. Instead of producing modules on the fly, they harvest completed runs for useful modules which are then subsequently used to augment the terminal set of another run.

To achieve this, a database of all subtrees generated during a run is maintained. The database maintains a count of the usage of each tree, which is then used to decide which subtrees should be added to the terminal set; the most frequently used ones being encapsulated as atoms.

It was shown that their method outperformed standard GP both in terms of the speed to good solutions and to the overall quality of solution generated. Perhaps somewhat surprisingly, they also showed that simply *randomly* choosing the subtrees gave them better performance than the usage-based approach.

Sequential Runs. There have been several attempts to pass information from run to run, although these have all been with a view to solving a *single* problem, rather than training a system that can be applied to successively more difficult problems. The most well known of these techniques is to simply seed a population with the *best-of-run* individual from the previous run. This has been used with mixed success by many researchers, as some have found that this can lead to very premature convergence on that individual.

Beasley described the use of *derating functions* for the discovery of all peaks in a multi-modal function [3]. His system attempted to discover one or two peaks with each run, and to then modify the fitness function of subsequent runs to penalise individuals that revisited that part of the search space. He discovered that his algorithms substantially outperformed standard niching techniques on these problems, mainly because they made the problem easier. Each run was presented with fewer and fewer peaks until they were all discovered.

A similar approach was taken by Streeter et all [9] with a system that iteratively refined a circuit across several runs. This work was concerned with generating an electric circuit which produces an output that is an approximation to the *error function* of an existing circuit, which may or may not have been generated using GP. The generated circuit can be added to the existing one to produce a new one which performs its task with greater accuracy. These iterations can be repeated as many times as necessary, or until the benefit of the improvements gained is outweighed by the cost of an extra run.

3 Mnemosyne

The Mnemosyne system implements RTL by keeping a library that is divided into segments, each segment containing functions of a certain arity. A segment has a function and terminal set defined for it. Typically the function set for a segment contains the primitive functions defined for the problem (domain), while

```
InitLibrary;
for each library iteration do
    RunGP;
    UpdateLibrary;
end
```

Algorithm 1: Overall Algorithm

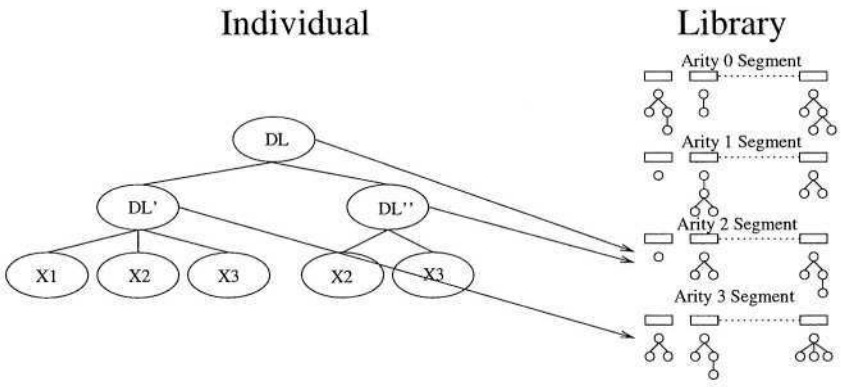


Fig. 1. An overview of Mnemosyne. Each DL-node consists of a floating point label which will be used to is used to look up the appropriate library segment, according to the arity of the function.

the terminal set contains the arguments for the function. Functions in segments have a floating point value associated with them, these are called *tags*. The tags are the main means for *linking* to library elements. Therefore the library elements are referred to as Tag Addressable Functions (TAFs).

A general overview of the Mnemosyne training algorithm is provided in Algorithm 1. It defines an individual GP run **(RunGP)** as a basic sub-routine that is used to collect statistics about the use of library elements. After each individual run, the library is updated using these statistics. Details on the update mechanism are provided in Section 3.1.

Individual runs use the library by virtue of the presence of a special function in the function set: a Dynamically Linked Node (DL-node). Such a node is fully specified by an arity and a *tag* value. The arity defines which segment the node links to, while the tag value defines which particular function is used, by searching for the nearest tag value in the segment. An overview of the contents of the library and how an individual links to the library is depicted in Figure 1. In the experiments below, the function set in the individual runs is limited to DL-nodes only, this in order to force the run to use the library and identify important elements, and to avoid using the bias inherently present in this set of primitive functions.

At the initialization phase of the individual runs, the DL-nodes are initialized with a uniformly chosen arity, and a tag value that is chosen uniformly from the

available tags. Thus, each TAF has an equal chance of being present in the initial population. During a run, a tag-mutation operator is used, which will mutate the tags in the DL-nodes, leading to the node being linked to (possibly) a different TAF. This mutation operator operates in conjunction with a regular subtree crossover and reproduction operator.

3.1 Updating the Library

TAFs consist of a tuple $\langle t, w, v, f \rangle$, where t denotes the tree (usually a function with one or more arguments), w the multiple run worth, v the floating point tag value and f the per run usage (frequency). A library consists of multiple TAFs, and is usually sorted on the tag value for quick *O(logN)* lookup. The library is initialized with randomly generated TAFs using a library specific function and terminal set using the ramped-half-and-half method. Initially, after loading or creating the library, the per-run frequency count f is set to 0. During the run, whenever TAF i is linked to a tree of size s, the frequency is updated by:

$$f_i \leftarrow f_i + 1/s$$

Normalisation by the size of the calling tree is performed to avoid a situation where the larger individuals in the population have the deciding vote on which tafs will propagate. After the run, the frequencies are normalised to sum to one, and added to the long-term worth of the library element:

$$w_i \leftarrow (w_i + f_i / \sum_j f_j) \times d$$

The library update mechanism attempts to ensure that the frequency of occurrence of library elements is proportional to their long-term usage information. First the mean usage $\bar{w}$ for library elements is calculated. Then, each library element for which $w_i \geq 2\bar{w}$, is selected for reproduction/variation. To keep a constant library size, an individual j to replace is chosen that has the lowest worth w. The chosen TAF i will either be reproduced (50% of the time) or crossed over with a randomly chosen TAF and copied over individual j. Subsequently, the w values of both i and j will be shared between the parent and the offspring, keeping the sum of w constant.

$$\begin{aligned} w_j &\leftarrow \frac{w_j + w_i}{2} \\ w_i &\leftarrow w_j \end{aligned}$$

And the tag value of the offspring is mutated by adding a normal distributed random number

4 Experiments

We tested the system on two classes of problems often associated with ADFs, namely the Boolean *even n-parity* and the *Lawnmower* problems. The even-n-paxity problem takes n Boolean arguments and returns **true** if an even number of the arguments are **true,** and otherwise returns **false.** The Mnemosyne system was implemented using ECJ [6].

The difficulty of these parity problems is easily scaled by increasing the number of inputs, and have proved a very tough benchmark for standard GP. Koza used these problems to illustrate the utility of employing ADFs, and showed that ADF GP could solve a selection of them with far less computational cost and a much smaller population (4,000 vs. 16,000). Koza examined problems up to and including the even-11-parity problem, and reported that the sheer time involved in running these experiments prohibited him from going any further, as the number of test cases increase exponentially with the number of inputs.

The Lawnmower problem was specifically designed as a test bed for ADFs [5]. It is a difficult enough problem to exhibit hierarchical decompositions and exploitable regularities, the characteristics upon which ADFs thrive. The Lawnmower problem is also such that it can be scaled with much more granularity than the Boolean even-k-parity problems.

In the Lawnmower problem, the aim is to produce a program that controls the movement of a lawn mower such that the mower cuts all the grass on a particular lawn. The lawn is laid out as a toroidal grid, and the lawn mower can occupy one grid location at a time. Koza examined problems from 32 up to 96 squares and demonstrated that, while ADFs aren't necessary to solve the problem, they do so much more quickly, and scale much better.

To tackle these problems, the usual syntactic distinction between terminals (leaf nodes) and functions (internal nodes) is abandoned and a more functional distinction of variables and functions is used: variables are considered to be problem dependent entities that are not transferable between different problem instances, while functions are a set of operations that are valid throughout a problem domain. This distinction allows for functions of arity 0 (e.g., sensors) to be considered different entities from variables (also of arity 0) as found in the parity problems. In the experiments, functions will be placed in the library, while variables (that are not supposed to be transferable) are used in the individual GP runs. The library will only use functions, while the evolving population will be constrained to use the library and the variables.

All experiments were done with the same set of parameters: libraries and individual runs are initialised using the ramped-half-and-half method (ramp varying between 2 and 6), population and library segment size is set to 500, the individual runs run for 50 generations, the decay factor in the library is set to 0.9, crossover probability in the library equals 0.5 (meaning that half of the variation events inside the library are straight reproduction events), the individual runs use 10% reproduction, 45% crossover and 45% tag mutation. The parameters have been set using a few initial runs, and are possibly not very optimal.

To compare the results of the library against results taken from the literature, the concept of *conditional effort* is introduced. The calculation of the conditional effort is the same as the normal effort, the difference being that it is conditional on the library iterations that have already been performed. After training a library, it will be tested on a new set of problems; and only this new set of problems is used to calculate the effort. To recognise the effort that has already been put in the library, the number of iterations the library has been trained is reported as well. This was necessary as the normal *number of individuals processed* statistic has not enough granularity to make distinctions between solving problems of varying difficulty. Also, in practice, we view the creation of a library not as an event that needs to be done for every problem we encounter, but advocate re-use of libraries to tackle new problems. Therefore, we view the effort to create the library to be on the same footing as changing other parameters, particularly tuning function sets to a particular domain.

4.1 Boolean Even-n-parity Experiments

The central thesis of this paper is that RTLs permit GP runs to transfer knowledge in the form of functions, not only from run to run, but also to transfer knowledge across increasingly difficult problems in the same domain. Thus, our baseline experiments involve comparing Mnemosyne to ADF GP on a single problem instance. A second set of experiments was designed to test the ability of the system to transfer knowledge from run to run. To this end, an RTL was generated ("trained") on one instance of the problem, and subsequently applied to problems of increasing difficulty. Two sets of experiments were carried out in this case; one for an RTL initially trained on the even-4-parity problem, and subsequently applied to 5- onto 10-parity, and another in which the RTL was initially trained on the even-5-parity problem, and subsequently applied to 6- to 10-parity problems.

For the boolean parity problems, the function set {AND,OR,NAND,NOR} is used. It is known that this function set is very biased *against* parity problems. Overcoming this bias and converging to the optimal set of either XOR or EQ, combined with negation could be a winning strategy. The library consists of 3 segments, for functions of arities 1,2 and 3. The primitive function set that is used inside the library consists of the aforementioned four primitive logical functions. The solutions evolving in the separate runs use the DL-nodes that refer to the elements of the library in the function set; the terminal set consists of the problem-dependent variables. In this setup, it is impossible to encode solutions for individual problems in the library. The library needs to evolve in such a way that it helps the individual runs to solve the problems.

4.2 Lawnmower Problems

A similar experimental set up was used for the Lawnmower problem. An initial library was trained on an 8x8 lawn, and then tested on progressively larger lawns, up to 8x12, the largest size reported on by Koza in [5]. The library consisted

of the the standard functions for this problem, that is { LEFT,MOW,V8A, PROGN2, FROG } while the evolving population was made up of DL–nodes. Also for this problem, the library consists of three segments, here of arity 0,1, and 2. The library and GP parameters were the same as for the parity problem.

5 Results

Figure 2 shows the results for training the library on the parity 4 and 5 problems and subsequently testing them on higher parities. Fifty independent libraries were trained on the lower parity problems for 95 iterations. The end–of–run libraries thus consisted of feedback obtained from 95 *dependent* runs, i.e., runs that use updated libraries. The libraries produced after every fifth iteration were saved and tested against the higher parity problems. Libraries with 0 iterations are randomly initialised. Thus, a test–run at library iteration 15, tests a library that was trained using 15 different runs of the smaller parity problem. Figure 2 shows a clear improvement, where the conditional effort of solving the parity problems decreases with library iterations. It is also clear that the library is capable of generalising: libraries trained on parity 4 have a better expected performance on the higher parities than untrained libraries. When comparing the libraries trained on parity 4 with those trained on 5, it is interesting to note that the libraries trained on the simpler problem apparently generalise better. It seems that the simpler problem allows the library to focus on the characteristics of this problem domain better. After 95 iterations of training on parity 4, applying this library on a parity 10 problem apparently seems to outperform standard GP using ADFs on parity 4 itself! Considering that parity 4 problems consist of a factor 64 less fitness cases than parity 10, it seems that the Mnemosyne RTL approach provides good scaling characteristics on this type of problem.

A cursory inspection of the content of the libraries for the segments for arity 2 shows that the library tends to converge to a majority of either the XOR or the EQ function, accompanied with low but persistent numbers of auxiliary functions. The library thus learned to remove the initial bias induced by the four primitives, and induced a bias more appropriate to this type of problem.

Figure 3 shows the computational effort needed to solve Lawnmower problems of increasing difficulty. Also here it can be seen that the computational effort decreases after longer periods of training, while the more difficult (larger) problems benefit from this increased training effort. Again, the benchmarks are easily surpassed. The second graph in Figure 3 shows a comparison of effort between a library *trained* on a lawn of 64 cells, and *tested* on a lawn of 256 cells, against a library *trained* on such a lawn of 256 cells. Although there is a fair amount of variability in the graph (this is caused by using 50 runs for calculating the effort), it is clear that at the very least the library that is trained on a smaller lawn does not have a worse performance than the library trained on the bigger lawn. Starting out with training on small and fast instances of a problem, seems to be helpful in solving the more difficult problem.

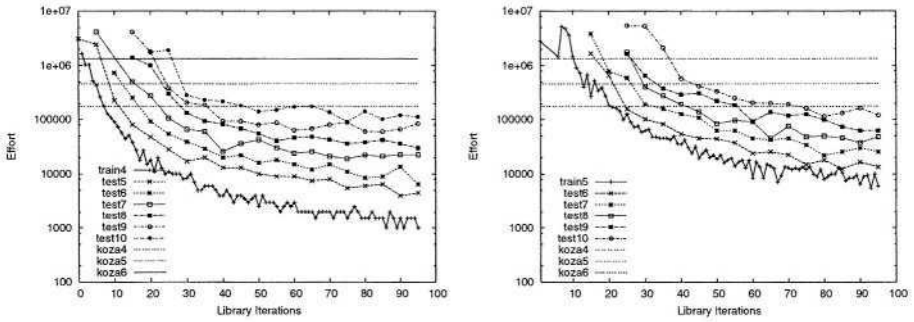


Fig. 2. Number of individuals needed to be processed to obtain a solution with 99% probability on the parity problems. Training was done using 50 independently trained libraries on 95 iterations, while testing was done on each library taken from every fifth iteration. The graphs show the results for libraries trained on parity 4 and 5. Also shown are the computational efforts as published by Koza using ADFs [5]. Runs that did not produce any solutions are not plotted. Note the log scale.

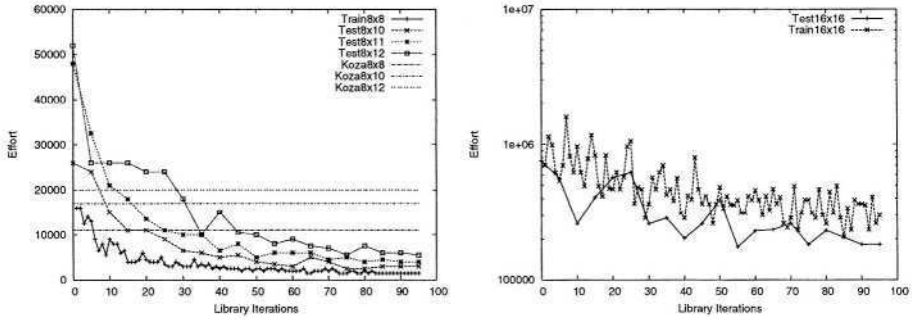


Fig. 3. Number of individuals needed to be processed to obtain a solution with 99% probability on the parity problems. Training was done using 50 independently trained libraries on 95 iterations, while testing was done on each library taken from every fifth iteration. The graphs show the results for libraries trained on lawns of size 8x8. Also shown are the computational efforts as published by Koza using ADFs [5].

Finally, a set of runs are performed where libraries are trained on random boolean problems of 5 inputs and an equal number of ones and zeros in the target output. The libraries were subsequently tested on the parity problems from 5 upwards to 10 inputs. The problem domain thus consists of a large subset of *all* logical problems, and the best the library should be able to do is to find an as *unbiased* set of functions as possible. Surprisingly, even in this setting, the library approach was able to get good performance: the conditional effort for parity 5 after training is 21,000 individuals; for parity 6, 42,000; parity 7, 111,000; parity 8, 240,000; parity 9, 388,000; and, finally, parity 10 1,056,000. Also here the effort is significantly lower than when trying to overcome the initial bias from scratch as is done using ADFs.

The results strongly suggest that the concept of Run Transferable Libraries is sound: it is possible to capture significant functionality from one or more training problems in a library and use it to solve related and/or more difficult problems more easily. The results also show that the approach of iteratively re-solving the problem significantly improves GP's ability to scale on these problems. On both problems there is an indication that solving simple 'toy' versions of a more difficult problem helps better in solving the larger problem than to try more difficult versions. It seems that the feedback received in these simpler versions leads more directly to the induction of the right functional bias.

6 Future Work

The current research is still in an early stage, yet a myriad of questions and possible avenues present themselves. What learning regime do we need to use to obtain a library that is optimal for a certain problem domain? Does the system reach a stage where it overfits, i.e., when is a library too focused on a problem instance that the library is no longer transferable to different problems and, if so, when does this happen? How can this be overcome? Does the locality enforced in Mnemosyne help in evolving a library? Does it make sense to cross/mutate libraries?

It is envisioned that there is ample opportunity for using Run Transferable Libraries to solve problems: in most circumstances, a practitioner of GP is interested in solving a set of related, yet different problems. By training on simpler instances and learning the underlying functional bias of the problem domain, it is expected that the performance of the basic search can be enhanced. As an example, consider the subject of image classification: although every problem instance will need to find an appropriate way to classify the particular set of images at hand, detecting particular features in an image is a process that transcends problem instances. By evolving feature detectors on various problems using an RTL, a general purpose image classification program could be build up and used for new problem instances.

The definition of Run Transferable Libraries thus presents a possibility to give an operational (yet circular) definition of a problem domain: *A problem domain is a set of problems such that the necessary abstractions to tackle problem instances can be encoded in the form of a library of functions.* The circle is completed by noting that an RTL is defined as a system that helps in tackling problem domains.

7 Conclusion

The concept of Run Transferable Libraries is introduced and its worth is tested using a concrete implementation on two well-known benchmark functions for testing functional abstraction. It is shown that transferring an evolving library of functions from one run to the next helps not only in solving the same problem faster, but enables scaling in at least two ways: the ability to solve difficult

problems faster by first solving simpler but related problems; and the ability to solve similar problems faster. It is hypothesised that a library is a good method to learn and encode the *functional bias* of a problem domain, i.e., that set of functionality that helps in solving problem instances taken from that domain faster.

Acknowledgements. The authors would like to thank Clearwire Corporation for the use of their servers for some of the runs.

References

1. P. J. Angeline and J. B. Pollack. The evolutionary induction of subroutines. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society,* Bloomington, Indiana, USA, 1992. Lawrence Erlbaum.
2. Peter John Angeline. *Evolutionary Algorithms and Emergent Intelligence.* PhD thesis, Ohio State University, 1993.
3. David Beasley, David R. Bull, and Ralph R. Martin. A sequential niche technique for multimodal function optimization. *Evolutionary Computation,* 1(2):101–125, 1993.
4. Kenneth E. Kinnear, Jr. Alternatives in automatic function definition: A comparison of performance. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming,* chapter 6, pages 119–141. MIT Press, 1994.
5. John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs.* MIT Press, Cambridge Massachusetts, May 1994.
6. Sean Luke. ECJ 10 : An Evolutionary Computation research system in Java. Available at http://cs.gmu.edu/ eclab/projects/ecj/, 2003.
7. Simon C. Roberts, Daniel Howard, and John R. Koza. Evolving modules in genetic programming by subtree encapsulation. In Julian F. Miller, Marco Tomassini, Pier Luca Lanzi, Conor Ryan, Andrea G. B. Tettamanzi, and William B. Langdon, editors, *Genetic Programming, Proceedings of EuroGP'2001,* volume 2038 of *LNCS,* pages 160–175, Lake Como, Italy, 18–20 April 2001. Springer-Verlag.
8. Justinian P. Rosca and Dana H. Ballard. Discovery of subroutines in genetic programming. In Peter J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2,* chapter 9, pages 177–202. MIT Press, Cambridge, MA, USA, 1996.
9. Matthew J. Streeter, Martin A. Keane, and John R. Koza. Iterative refinement of computational circuits using genetic programming. In W. B. Langdon, E. Cantú–Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference,* pages 877–884, New York, 9-13 July 2002. Morgan Kaufmann Publishers.

Using Genetic Programming to Obtain a Closed-Form Approximation to a Recursive Function

Evan Kirshenbaum and Henri J. Suermondt

HP Labs, Palo Alto, CA
{Evan.Kirshenbaum, Jaap.Suermondt}@hp.com

Abstract. We demonstrate a fully automated method for obtaining a closed-form approximation of a recursive function. This method resulted from a real-world problem in which we had a detector that monitors a time series and where we needed an indication of the total number of false positives expected over a fixed amount of time. The problem, because of the constraints on the available measurements on the detector, was formulated as a recursion, and conventional methods for solving the recursion failed to yield a closed form or a closed-form approximation. We demonstrate the use of genetic programming to rapidly obtain a high-accuracy approximation with minimal assumptions about the expected solution and without a need to specify problem-specific parameterizations. We analyze both the solution and the evolutionary process. This novel application shows a promising way of using genetic programming to solve recurrences in practical settings.

1 Introduction

In monitoring applications, the false positive rate of a detector is an important statistic used to characterize the practical applicability of the detector. False alarms are not just annoying, but they also tend to make those responsible for responding to alarms less likely to respond to a real alarm and are a frequent reason for people to turn alarms (or even the underlying detector) off. Even in unsupervised environments, false alarms lead to large event logs, fruitless investigations, and unnecessary cost. Therefore, when developing novel detectors for use in time-series monitoring, we needed (among other things) a good way to estimate the expected number of false positives in a negative sequence of a given length, as a core performance statistic by which we judged the detector.

Unfortunately, the performance data available on the detector does not lend itself to easy characterization of the expected number of false positives in a sequence of arbitrary length. Instead, we find ourselves with a multi-parameter nonlinear recurrence relation for which conventional methods do not yield a closed form or even a closed-form approximation.

The recurrence relation results from the following problem definition. Since the detector is stateful and the elements of the series are not independent, we cannot simply assume an independent probability of getting a false positive at each new element. The detector will be run on the sequence until it signals a detection and then

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 543–556, 2004.

will be retrained for a fixed number of points T. In our particular case, the data is assumed to follow an ARIMA model [7], and the detector uses Holt-Winters exponential smoothing [14]. In an earlier stage of the investigation, we had tested detectors on randomly generated negative sequences of arbitrary fixed length w. In this setup, all the sequences were negative, so any time a detector signaled an event, it should be considered a false positive. We had derived functions that would estimate, for any parameterization of the data stream and detector, the expected false positive rate FPR, viewed as the proportion of these randomly generated negative sequences for which the detector would spuriously signal detection, and the expected penetration p into a sequence before a false positive occurs, averaged over those sequences for which a false positive occurs.

So the basic notion is that $1 - FPR$ of the time, the detector will get through a negative sequence of length w without signaling detection, and the remaining FPR of the time it will get on average p in before signaling, retrain for T, and start over.

We make two minor yet practical simplifying assumptions. First, in a sequence that is longer than w (the length of our test sequences), the behavior of the detector after successfully scanning a window of size w is the same as its behavior when looking from scratch at another sequence generated from the same model (in other words, the detector does not take advantage of its knowledge of the first w data points). Second, after the retraining (recalibration) period of length T, the detector also behaves as it does when looking from scratch at a sequence generated from the same model. If we define

$$d = p + T \tag{1}$$

as the expected number of points consumed by a false positive, these assumptions yield as a first-order approximation for the expected number of false positives the recurrence relation

$$NFP_{len} = (1 - FPR)NFP_{len-w} + FPR(1 + NFP_{len-d}) \ . \tag{2}$$

Unfortunately, for our purposes, we needed to be able to determine this quantity a large number of times for different parameterizations of the detector, and a recursive definition was not efficient enough, so we needed to find a closed form solution, or at least a good approximation. When $d = w$ (i.e., the case where our test sequences happen to be the same length as the expected number of points consumed by a false positive), this trivially reduces to

$$\begin{aligned} NFP_{len} &= NFP_{len-w} + FPR \\ &= \frac{FPR \cdot len}{w} \end{aligned} \ , \tag{3}$$

but when $d \neq w$ (as it does, since d is a function of w), things rapidly get messy.

There are many techniques for solving recurrences [5, 6, 10, 13]. Most such techniques work only for recurrences of a particular form, typically they involve trial-and-error and guesswork, and many of them become unworkable when the element being determined depends on the value of an element far away, as they do in our problem, in which NFP_{len} depends on NFP_{len-w}. Symbolic math packages can

automatically solve specific forms of recurrence relations. *Mathematica* [22], for example, can solve linear recurrence relations for a sequence in which s_n is defined in terms of expressions in s_{n+i} or $s_{q^i n}$, for constant values of i and where there are a fixed number of base cases. In the problem we are addressing, the referenced subscript offsets are non-constant and the number of base cases depend on the parameter w.

Finding a closed-form solution to (2) is exactly the sort of non-linear functional regression problem that genetic programming [1, 12] is asserted to be good at solving, and this gave us an opportunity to test whether it could provide a solution without any special problem knowledge or tweaking of the GP parameters.

2 Experimental Setup

The training set consisted of 1,000 cases randomly drawn over

$$\begin{aligned} fpr &\in [0,1] \\ w &\in [100,200] \\ d &\in [20,150] \\ len &\in [300,1000] \end{aligned} .$$

For each case, we computed NFP_{len}, where

$$NFP_i = \begin{cases} 0 & \text{if } i < d \\ fpr\,(1 + NFP_{i-d}) & \text{if } d \le i < w \\ (1 - fpr)\,NFP_{i-w} + fpr\,(1 + NFP_{i-d}) & \text{if } i \ge w \end{cases} . \quad (4)$$

The range of values for NFP_{len} was from 0.0031 to 37.8361.

2.1 GP Parameters

For the genetic programming component, we used GPLab, a genetic programming tool developed at HP Labs [9] and used both for investigating and extending genetic programming and for solving real-world problems. We decided that it would be particularly valuable to establish how well GPLab (and genetic programming in general) worked "out of the box", so we left the many parameters that control the run set to default values established as ones that have seemed to work well on problems in the past.

In particular, we ran with a population of 10,000 candidates in each generation. The initial population consisted of expression trees between four and seven levels deep over an operator set consisting of the variables *fpr, w, d,* and *len,* real-valued addition, subtraction, multiplication, and division, integer constants between –10 and 10, and real constants between –10 and 10, with a resolution of 0.001. Division was defined such that division by zero resulted in a value of one. We were frankly skeptical that a good solution would be found with such a restricted set of operators, but it seemed the most honest way to get a baseline.

Half the cases (500) were randomly selected for use as training cases. The other half were split into two validation sets to be used in monitoring the progress of the run and choosing the "winner". A separate set of 1,000 cases was generated for testing the chosen function after the run. Each candidate function was presented with a set of 50 training cases, biased toward those adjudged "more difficult" by a reinforcement learning mechanism. The fitness pressure was toward a lower mean relative error. Ties were broken by the number of nodes in the expression tree and then by the mean absolute error in the prediction. Candidates which attempted to evaluate more than 100 operators were judged "infinitely bad".

In each generation, a randomly chosen 10% of the top 25% of the population (250 candidates overall) were tested on the entire set of training and validation cases, and the ones that performed better than the current best on the training set, either of the two validation sets, or the entire set were preserved for later analysis. The overall winner would be the final leader over all 1,000 cases.

Reproduction occurred generationally, with parents chosen by tournament selection with a tournament size of three. In each generation the best candidate on the tests encountered was preserved. For the remainder, 77% (10/13) were a product of crossover, 8% (1/13) were copied from the parent generation, 8% (1/13) were modified by point mutation (the replacement of the operator of a node by another operator with a compatible signature), and the remaining 8% (1/13) were modified by applying a Gaussian drift to a constant. "Classical" genetic programming mutation, in which a new subtree is grown, was not used.

The experiment ran for 1,000 generations.

3 Other Methods

Before discussing the results obtained through genetic programming, it is worthwhile to consider, as a baseline, the performance that can be obtained via other machine learning methods. Note that the results presented in this section do not represent a serious investigation into the optimal parameterization of any of the models used, nor are we experts in the use of most of them. Even so, the results indicate the level of difficulty that this problem presents for traditional machine learning techniques.

The experiments in this section were performed using the WEKA toolkit (version 3.4) [20]. For purposes of comparison, all negative predicted values were clipped to zero. All predictors were trained on the same 1,000 training cases and evaluated on the same 1,000 testing cases used in the genetic programming experiments.

Linear regression resulted in the following model:

$$E[nfp] = 8.0691\,fpr - 0.0056\,w - 0.0275\,d + 0.0053\,len - 1.0497\ , \qquad (5)$$

which has a mean relative error of 264.2%, a median of 32.158%, a maximum of 60,550%, and a 95^{th} percentile of 550.2%. Some of this error is due to predicting negative numbers, which are physically impossible. When negative predictions are clipped to zero, the model has a mean relative error of 64.03%, a median of 32.16%, a maximum of 4,637%, and a 95^{th} percentile of 121.19%. The correlation with the actual values is 0.855. Clearly, such a model would be useless as a predictor.

Other methods tried (and their respective mean relative errors) include Radial Basis Function Networks (544.92%), Conjunctive Rules (305.34%), Decision Stump (one-level decision trees, 305.20%), Locally Weighted Learning [2] (247.53%), Neural Network [16] (113.34%), K* [4] (90.34%), Decision Table (70.65%), *k*-Nearest Neighbor [1] (68.41% with $k = 1$ and $k = 5$), Pace Regression [21] (64.03%), Simple (one-term) Linear Regression (53.73%), Sequential Minimal Optimization [17] (47.18%), Least Median Squared Linear Regression [15] (43.16%), REP Trees (35.20%), M5 [8] (15.55%), and M5 Prime (11.22%). None of these methods was able to get an expected error of less than 10%, and only three of them (K*, M5, and M5 prime) were even able to get the median error below 10. The best of the techniques, M5 prime had a median relative error of 5.23%, maximum of 303.45%, a 95th percentile of 41.68%, and a correlation with the actual values of 0.987.

4 Results

In the initial random population of our genetic programming run, the best solution was equivalent to

$$fpr + \frac{fpr(len - d)}{w} \quad . \tag{6}$$

This had a mean relative error of 17.87% over all of the cases. The median relative error was 12.80%, the maximum was 83.87%, and the 95th percentile value was 50.51%. The correlation with the actual values was 0.822. This can be used as an estimate of how well one can expect to do by guessing a function 10,000 times. By contrast, the best linear regression model has a mean relative error of 286.50%, a median relative error of 34.333%, a maximum relative error of 63,469.47%, and a 95th percentile error of 565.25%, but its correlation is slightly better, at 0.832.

Looking at Fig. 1, in which the x-axis represents the cases sorted by the value of the target function, we see that the best random function (medium gray triangles) is a nice match for the target function (black diamonds) when the target is small, but for approximately the last third of the range it begins to seriously underestimate the value. By contrast, the linear model (light gray squares) is closer for the larger values, but underestimates the smaller ones. This leads to its significantly high relative error, even though the correlation is slightly higher.

As the run progressed, the "overall best so far individual" rapidly improved. The mean relative error was below 15% by generation 14, below the 11.22% of the best other method by generation 22, below 10% by generation 30, and below 5% by generation 144. The experiment took place on a 2.0 GHz Pentium 4 processor, and each generation took roughly 8.5 seconds. The entire run of 1,000 generations took two hours and 23 minutes. The progress can be seen in Fig. 2. The heavy black line tracks the mean relative error, the medium gray line at the bottom tracks the median relative error, and the light gray line at the top tracks the 95th percentile relative error. The maximum error (not shown on the chart) begins at 84% and only exceeds 100%

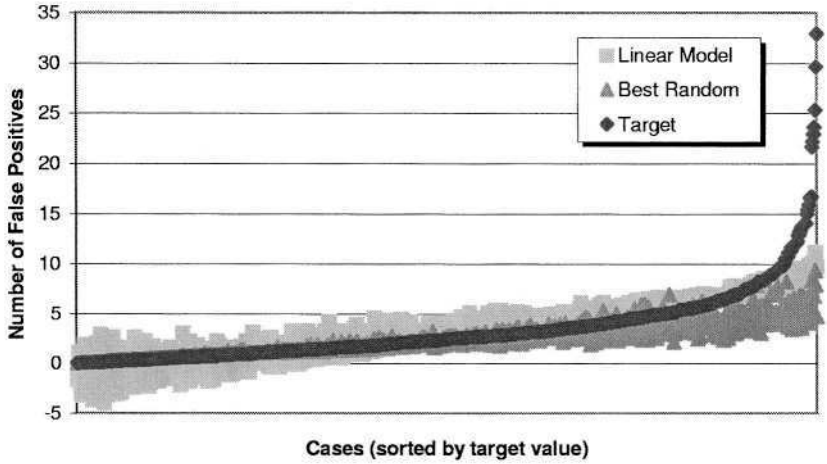


Fig. 1. Target function, best-fit linear model, and best overall candidate from initial random generation. The data points are sorted by target value

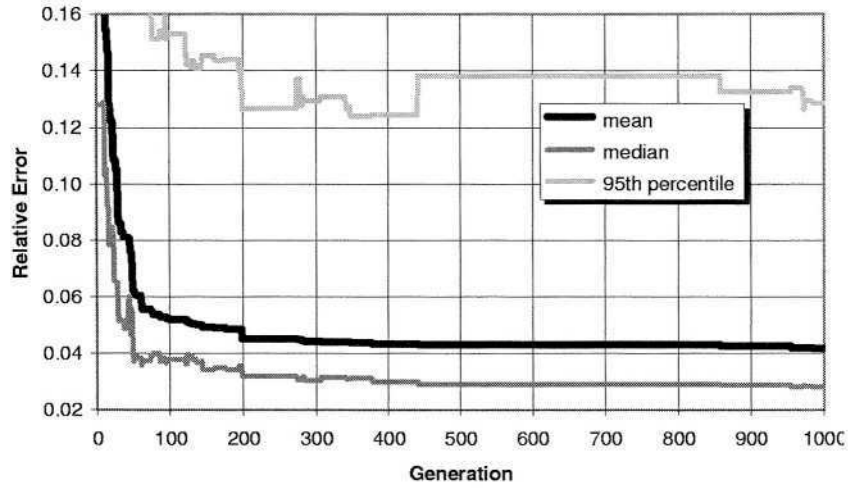


Fig. 2. Statistics for "overall best-so-far" candidate

for two generations (one "best-so-far" candidate). After generation 45, it is only greater than 60% occasionally between generations 280 and 440. After generation 200, not only are nearly all of the errors less than 14% (and half of them less than 3.2%), but the absolute worst case is almost always less than 60%.

The overall best candidate appeared in generation 990, after two hours and 22 minutes of unattended experiment time. This candidate had an overall mean relative error of 4.183%. The median relative error was 2.823%, the 95^{th} percentile relative error was 12.862%, and the maximum relative error was 45.353%. Looking at

absolute error (which was *not* a primary fitness pressure), the mean error was 0.147, the median error was 0.051, the 95th percentile error was 0.581, and the maximum error was 8.992. The overall best candidate did only slightly better (4.041% vs. 4.253%) on cases in the training set than it did on cases outside of the training set. When tested on a new set of 1,000 training cases, the overall best candidate did nearly identically. The mean relative error was 4.268%, the median was 3.036%, the 95th percentile was 13.395%, and the maximum was 26.72%. The correlation with the target was 0.993.

The response curve for the overall best candidate can be seen in Fig. 3. Here we see that the solution (gray circles) is a near-perfect fit to the target (black diamonds). The overall best candidate for the run had 95 nodes in its tree, and when simplified by *Mathematica* was equivalent to

$$\frac{fpr\left(\frac{w\,fpr^2}{10}+fpr+d\right)(len-d+\alpha\,fpr+0.4561w-\beta)}{w(d-0.07823\,fpr^2\,w)} \quad (7)$$

where

$$\alpha = 1.851479 + len\left(\frac{fpr}{fpr-6.47463}+\frac{len}{len^2-0.2127\,d}-\frac{d}{w}+1\right)$$
$$\beta = \frac{0.0502266\,fpr\cdot w(d\cdot len\cdot w-1)}{(d+9\,fpr)len\cdot d} \quad (8)$$

It is relatively easy to see this as a refinement of the

$$fpr+\frac{fpr(len-d)}{w} \quad (9)$$

found in the initial generation. As with many evolved solutions, this one involves a fair number of "magic numbers", and they should always be looked upon with some suspicion. Constants drift slowly during a run, and often there may be a better number in the near vicinity. Very small numbers often indicate that a spurious term is being driven toward zero. But too much should not be made of this skepticism. The suspiciously small constant, 0.07823, multiplied by the square of a variable that averages 0.5, looks to indicate that the term is useless. But if the denominator is replaced by $w \cdot d$, the solution actually becomes significantly worse, having a mean relative error of 8.269%, a median relative error of 5.631%, and a correlation of 0.968. On the other hand, 0.07823, while close, does not appear to be optimal. Investigating by hand within *R* [19] shows that the minimum mean relative error is 4.244% (from 4.268%) when the constant in that position is 0.082496. Such a difference is probably not worth a tremendous amount of effort, but it does show that evolved solutions may not be optimal, especially for a novel data set.

5 Overfitting

One of the prime fears when learning a function from data is that one will find a solution that works well on the training cases but which does not generalize to other

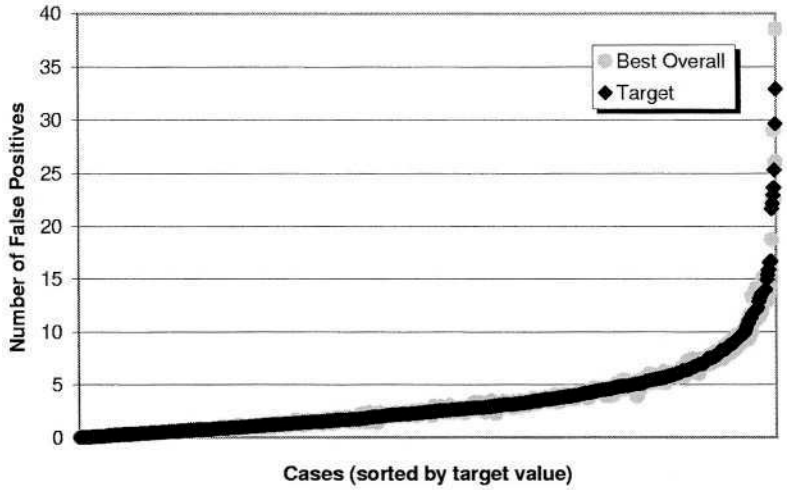


Fig. 3. Target function and best overall candidate, found in generation 986. The data points are sorted by target value

cases. This is known as *overfitting* the training data. Many learning methods, including genetic methods, are susceptible to this problem, but in our experience, genetic programming appears to be less likely to overfit even a reasonably small number of training cases than other methods.

Clearly the overall best candidate did not overfit the data, as shown by its performance on a set of test cases that were completely different from the ones used during its evolution. But ten million candidate programs were evaluated, and 72 were selected as "overall best so far", so if the experiment had been stopped earlier a different candidate would have been chosen as the overall best, and it is possible that that candidate would not have been as general.

One way to get confidence of the generality of a solution is to watch how well the solution does on out-of-set cases. Fig. 4 shows the performance of the best-so-far candidate, selected based on overall performance, evaluated over all the cases (black line) as well as over the training cases (medium gray line) and validation cases (light gray line) separately. If a candidate is overfitting the data, we should see the training error decrease while the out-of-set error remains high. Plainly, this is not happening. Indeed, for a plateau from generation 440 through generation 856, the performance on validation cases was better than the performance on training cases.

It should be stressed that genetic programming is not immune to overfitting, and it is even the case that a single run may alternate general and overfit solutions vying for the "top spot". But in our experience, the technique appears to be much more likely to find a general solution than most other techniques, perhaps because overfitting results in more complicated functions than general solutions, and simpler functions tend to be found first. It certainly seems to be the case that if there *is* as reasonably simple exact solution (that is, one that can be exactly described in an expression containing less than, say, 50 or so operators) it is much more likely to be found than a solution which just happens to fit the data.

6 Repeatability

Having analyzed the results of the run, the question arises as to whether we were merely lucky. Genetic programming is a highly nondeterministic process, and it might be the case that this particular run was anomalous, and that there's no particular reason to believe that other runs will do as well. To test this, 25 more runs were performed with identical parameters and training cases.

As can be seen in Fig. 5, the run described was actually the among the *worst* of the runs. It resulted in a solution with a mean relative error of 4.133% on the training and validation sets, which was worse than all but one of the subsequent runs. On the testing set, the primary run solution's mean relative error of 4.268% was worse than all but two of the 25 subsequent runs.

The mean relative error over the 25 runs averaged 3.500% on the training and validation sets and 3.675% on the testing set. On the testing set, the expected median was 2.473%, the maximum was 34.575%, and the 95th percentile was 11.566%. For the best of the runs, the mean relative error on the testing set was 3.242%, for the worst it was 4.429%, and for the median run it was 3.504%.

The primary run took 986 generations to reach its in-set level of 4.183%, while the other 25 runs only required, on average, 187 generations to surpass that level, with 13 doing so in less than 100 generations, nine in less than fifty generations, and one requiring only 24 generations. By generation seventeen, the expected solution performed better than all of the models produced by the other methods we tried. Some of this is due simply to the increased expressiveness of the genetic programming representation. In the initial random population, the expected mean relative error (equivalent to guessing 10,000 times) was 27.126% and the minimum over all 25 runs (equivalent to guessing 250,000 times) was 20.855%.

Looking at Fig. 5, it is apparent that several of the runs did strikingly better than the others. When the best-so-far candidates are examined, it becomes apparent that, probably due to initial random variation, these two runs refined a different strategy

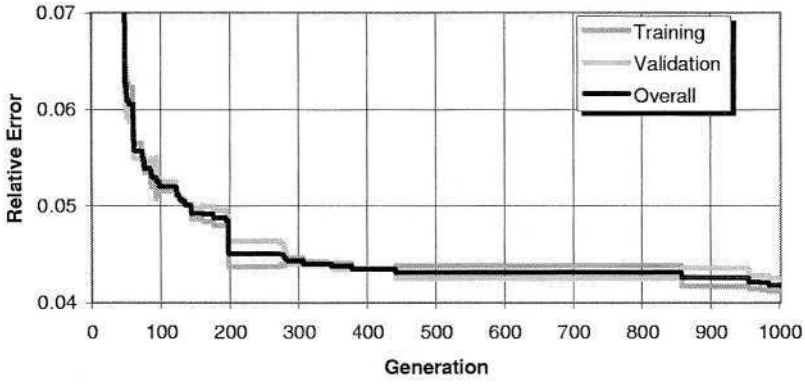


Fig. 4. Mean relative error overall compared with mean relative error on training cases and validation cases

from the other three runs. In runs that converge slowly, the top level of the recorded trees tend to be multiplications by *fpr*. That is, the population discovered early on that the problem could be thought of as finding an appropriate multiplier for the false positive rate, and the evolutionary process discovered better and better approximations to that multiplier. In runs that converge quickly, by contrast, the root of the tree for most of the run is a division operator, and in many cases the numerator is simply *len* or something that contains *len* as a principal component.

In all of the runs, however, both strategies are seen early on, so it appears that the strategy of treating the problem as essentially a division works better *provided* that it does well enough soon enough. Otherwise, it appears that that strategy gets bred out of the population by a somewhat more resilient strategy of multiplying by *fpr,* which improves more slowly and never seems to get to the levels of the division strategy.

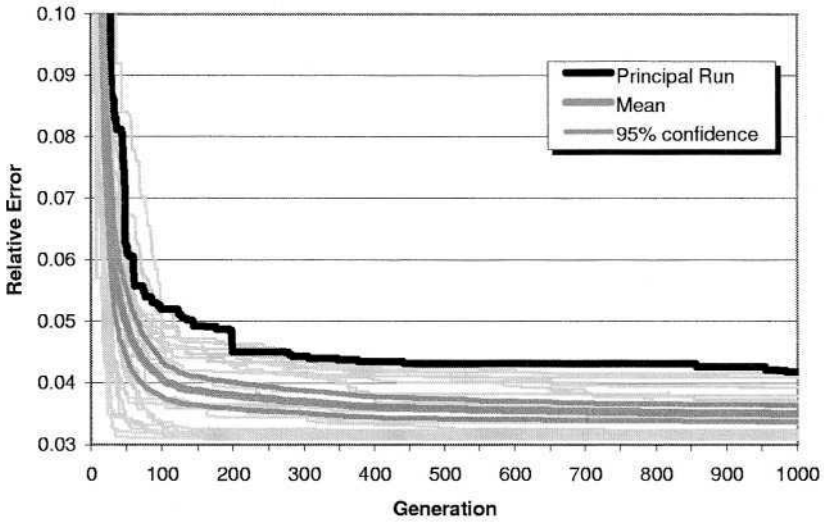


Fig. 5. Mean relative error on overall best-so-far candidates for 25 runs

There are more advanced techniques which can be used to counter—or even take advantage of—such tendencies. On the one hand, there is *deme-based* evolution [18], in which there are multiple (possibly asynchronously) evolving populations which, every so often, send good representatives to one another. This can keep a population from getting stuck in a rut and can permit the combination of mature strategies. On the other hand, there are methods for combining the results of independent runs to get solutions that are more stable than can be expected from any given run—and may, in fact, be better than any of the individual solutions. The latter technique is especially useful when it is feared that an individual solution has a small chance of being unusably bad but there is no way of determining this before it is used. Combining multiple runs can result in a solution that is much less likely to perform poorly, and, indeed can, in some cases, result in a solution that outperforms any individual solution. In the current experiment, sampling 1,000 five-solution "teams" from the 25 solutions generated and considering the prediction of each team to be the mean of the predictions of its members drops the expected mean relative error from 3.675% to

3.366%. Dropping the high and low predictions before taking the mean drops it further to 3.317%.

7 Solution Complexity

Having shown that genetic programming can produce useful results without worrying about customizing the parameters, we now look at what happens when you change one of them.

The solutions discovered all bumped up against the evaluation budget of 100 operations used as a run parameter. Since there were no loops, conditionals, or automatically defined functions in the allowed operator set, this means that the resulting expression trees all had nearly 100 nodes. This is a commonly-seen property of longer genetic programming runs, in which the population will experiment with strategies having different sizes and shapes for the first few dozen generations until it settles on the best strategies having at or near the maximal number of operators, spending the rest of the run refining this strategy. With nearly a hundred nodes, the expressions can get, as we have seen, quite complex, and it is often extremely difficult to understand them without a fair bit of interaction with a tool such as *Mathematica.* Of course, often we really don't *need* to understand the result, we merely need a result we can have confidence in. But if understandability is important, it can be useful to investigate whether a useful result can be obtained with a smaller allowable tree size.

When this problem was run with an evaluation budget of fifty operations, the mean relative error on the testing set over 25 runs was 3.716%, only slightly worse than the 3.675% with a budget of 100 operations. The expected median was 2.480%, the expected maximum was 29.339%, and the expected 95^{th} percentile was 12.046%. The best run had a mean relative error of 3.195%, the worst run 5.292%, and the median run 3.309%.

Pushing further, an evaluation budget of only 25 operations appears to be sufficient. The first such run yielded a result which simplified to

$$\frac{fpr\,(len - d + 0.273\,w\,(2 - fpr))}{w\,(1 - fpr) + fpr \cdot d} \tag{10}$$

in generation 301. This solution had a mean relative error of 3.238% (3.340% out-of-set), a median of 2.054%, a 95^{th} percentile of 10.377%, and a maximum of 24.492%. The second run yielded a result which simplified to

$$fpr\left(\frac{len}{w\,(1 - fpr) + fpr\,(d + 2.7491)} + \frac{0.862299\,w - 1.8623\,d}{d + w}\right) \tag{11}$$

in generation 556. This solution had a mean relative error of 3.379% (3.3095% out-of-set), a median of 2.224%, a 95^{th} percentile of 11.241%, and a maximum of 25.643%.

When 25 runs were performed with an evaluation budget of 25 operations, the mean relative error (on the testing set) of the solutions ranged from 3.304% to 9.600%, with a median of 3.744% and a mean of 4.824%.

There does appear to be a lower limit, however. The first run with an evaluation budget of 15 operations was unable to produce an individual better than one equivalent to

$$\frac{fpr \cdot d}{w(1-fpr)+fpr \cdot d} \tag{12}$$

found in generation 44. This candidate had a mean relative error of 7.112% (7.0114% out-of-set), a median of 5.208%, a 95^{th} percentile of 20.533%, and a maximum of 48.520%. When 25 runs were performed with an evaluation budget of 15 operations, the best solution found did no better than a 4.505% mean relative error on the testing set, and the worst run was 11.651%, the median run 5.382%, and the mean was 7.426%.

While clearly these solutions are not as good as the ones obtained when the candidates are allowed to be a bit larger, the fact that several runs investigated hit on similar functions indicates that this is probably to some extent a "strongest component". Unfortunately further runs based on both 15-node and 25-node solutions were unable to find any signal in the residuals.

8 Population Size

Another parameter worth looking at is population size. To investigate this, 25 runs were performed at each of the following population sizes: 20,000, 5,000, 1,000, 500, and 100. All other parameters were kept the same.

As discussed above, over 25 runs of 1,000 generations each with a population size of 10,000 candidates, the solutions ranged from a mean relative error (on the testing set) of 3.242% to 4.429% with a median of 3.504% and a mean of 3.657%.

When the population size is raised to 20,000, the solutions ranged from a mean relative error of 3.193% to 4.300%, with a median of 3.322% and a mean of 3.509%. At a population size of 5,000, the solutions ranged from a mean relative error of 3.207% to 4.969%, with a median of 3.913% and a mean of 3.943%. At a population size of 1,000, the solutions ranged from a mean relative error of 3.397% to 8.181%, with a median of 4.696% and a mean of 5.004%. At a population size of 500, the solutions ranged from a mean relative error of 3.374% to 8.467%, with a median of 5.636% and a mean of 5.694%. At a population size of 100, the solutions ranged from a mean relative error of 3.908% to 37.855%, with a median of 9.842% and a mean of 13.174%.

9 Discussion

Although many techniques for solving or approximating recurrences are known, all of them are somewhat painful for the non-mathematician researcher who has a practical need for solving such recurrences. In this paper, we demonstrate the use of genetic programming as a fully automated, out-of-the-box solver for such a problem.

The recursive function took one argument and had three constant parameters, yielding a solution that is a function of four variables. The problem was real, and the solution was not known ahead of time. In fact, when we presented our result to a number of statisticians and operations-research experts, they were pleased with the quality of the solution and the simplicity of its form, and most of all they were surprised by the fact that we were able to find a closed-form approximation at all for the recurrence we had formulated.

In order to investigate just how well genetic programming would work on such a problem without requiring any special expertise on the part of the experimenter, we deliberately ran the experiment with an "out-of-the-box" configuration and what we assumed would be an overly restrictive set of operators. To our surprise, it worked very well. The solutions derived were not merely good enough, they were good—although they were not the perfect solutions we would have liked—and they were much better than the solutions derived using the out-of-the-box configurations of any of a suite of other machine learning techniques, none of which produced a usable result. The method showed absolutely no tendency to overfit, nor did it appear to optimize the single statistic of interest at the expense of others, and multiple runs demonstrated that the initial satisfactory result was a likely outcome of the method, and, indeed, that even better solutions could often be expected.

It is true that genetic programming does take far longer than many other techniques, but it took less than three hours from writing the scripts to generate the data set to having an answer, and for a real-world problem whose solution is to be used in other systems, three hours is perfectly reasonable—indeed for many such problems three *days* would be perfectly reasonable.

One question, of course, is whether we were simply lucky with this particular recurrence. Future work is to characterize the performance of this technique on many other classes of recursive functions, including those on which well-known techniques are known to yield closed forms (as we would like this to work reasonably well without any knowledge or guess as to what the class of solution should be). Early experiments regressing to the Fibonacci function, which has a known-good closed-form approximation, are promising.

References

1. Aha, D.W., Kibler, D., Albert, M.K.: Instance-Based Learning Algorithms. Machine Learning 6 (January, 1991) 37–66
2. Atkeson, C., Moore, A., Schaal, S.: Locally Weighted Learning. Artificial Intelligence Review 11 (April, 1997) 11–73
3. Banzhaf, W., Nordin, P., Keller, R.E., Francone, F.D.: Genetic Programming: An Introduction. Morgan Kaufmann (1998)
4. Cleary, J.G., Trigg, L.E.: K*: An Instance-Based Learner Using an Entropic Distance Measure. Proc. 12th Int. Conf. on Machine Learning, (1995) 108–114
5. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. MIT Press (1990)
6. Graham, R.L., Knuth, D.E., Patashnik, O.: Concrete Mathematics. Addison-Wesley (1989)
7. Hamilton, J.D.: Time Series Analysis. Princeton University Press (1994)
8. Holmes, G., Hall, M., Frank, E.: Generating Rule Sets from Model Trees. Australian Joint Conf. on Artificial Intelligence (1999) 1–12

9. Kirshenbaum, E.: GPLab: A Flexible Genetic Programming Framework. Tech Report HPL-2004-12, Hewlett-Packard Laboratories, Palo Alto, CA (2004)
10. Knuth, D.E.: The Art of Computer Programming, Vol. 1: Fundamental Algorithms. 3rd edn. Addison-Wesley (1997)
11. Kohavi, R.: The Power of Decision Tables. Proc. Euro. Conf. on Machine Learning. Lecture Notes in Artificial Intelligence, Vol. 914. Springer-Verlag (1995) 174–189
12. Koza, J.R.: Genetic Programming. MIT Press (1992)
13. Lueker, G.S.: Some Techniques for Solving Recurrences. ACM Comp. Surv. 12(4) (1980)
14. Makridakis, S, Wheelwright, S.C., Hundman, R.J.: Forecasting: Methods and Applications 3rd edn. John Wiley & Sons (1998)
15. Rousseeuw, P.J., Leroy, A.M.: Robust Regression and Outlier Detection. John Wiley & Sons (1987)
16. Rumelhart, D.E., McClelland, J.L., PDP Research Group: Parallel Distributed Processing. MIT Press (1986)
17. Smola, A.J., Scholkopf, B.: A Tutorial on Support Vector Regression. NeuroCOLT2 Technical Report Series NC2-TR-1998-030, 1998.
18. Tackett, W.A., Carmi, A.: The Donut Problem: Scalability, Generalization and Breeding Policies in Genetic Programming. In: Kinnear, K.E. (ed.), Advances in Genetic Programming. MIT Press (1994)
19. Venables, W.N., Smith, D.M., R Development Core Team: An Introduction to R. http://cran.r-project.org/doc/manuals/R-intro.pdf
20. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools with Java Implementations. Morgan Kaufmann (2000)
21. Wang, Y., Witten, I.H.: Modeling for Optimal Probability Predictions. Proc. 19th Int. Conf. of Machine Learning (2002)
22. Wolfram, S.: The Mathematica Book. 4th edn. Wolfram Media, Inc. and Cambridge University Press (1999)

Comparison of Selection Strategies for Evolutionary Quantum Circuit Design

André Leier[1] and Wolfgang Banzhaf[2]

[1] Dept. of Computer Science, University of Dortmund, 44221 Dortmund, Germany
andre.leier@cs.uni-dortmund.de
[2] Dept. of Computer Science, Memorial University of Newfoundland, St. John's, NL, A1C 5S7 Canada
banzhaf@cs.mun.ca

Abstract. Evolution of quantum circuits faces two major challenges: complex and huge search spaces and the high costs of simulating quantum circuits on conventional computers. In this paper we analyze different selection strategies, which are applied to the Deutsch-Jozsa problem and the 1-SAT problem using our GP system. Furthermore, we show the effects of adding randomness to the selection mechanism of a (1,10) selection strategy. It can be demonstrated that this boosts the evolution of quantum algorithms on particular problems.

1 Introduction

Quantum Computing results from the link between quantum mechanics, computer science and classical information theory. It uses quantum mechanical effects, especially superposition, interference and entanglement, to perform new types of computation which promise to be more efficient than classical computation. However, up to now only a very narrow class of problems is known to be sped up by quantum algorithms, including factoring, (Shor's algorithm) [1], unstructured search (Grover's algorithm) [2] and also certain structured combinatorial search problems (Hogg's algorithm) [3] plus a few other number-theory problems [4,5,6]. Unfortunately, progress is still slow, yet the development of quantum algorithms seems to be crucial for future prospects of quantum computing.

Automatic quantum circuit design was motivated by the difficulties in manual quantum circuit design, because quantum algorithms are highly non-intuitive and practical quantum computer hardware is not yet available. Thus, quantum computers have to be simulated on classical hardware which naturally entails an exponential growth of computational costs and allows only to simulate small quantum systems (i. e. with only few qubits). Huge search spaces in even the simplest problems render evolutionary approaches nearly unable to achieve breakthrough solutions in the development of *new* quantum algorithms. At present we must be content to *evolve* essentially already existing quantum algorithms and to analyze the search space of these quantum circuits in order to improve

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 557–568, 2004.

the efficiency of evolutionary search. Since the simulation of quantum circuits cannot be sped up, the only way to novel, complex quantum circuits leads via accelerated evolution.

This is our motivation for examining and comparing different selection strategies with respect to their effectiveness. Using our linear GP system, that comprises a quantum simulator for fitness evaluation of quantum programs, we implemented the (μ, λ) and $(\mu + \lambda)$ ES selection as part of a generational GP approach and tournament selection as part of a steady state GP. Moreover, we (i) combined the (1,10) selection strategy with additional randomness and, (ii) tested independently a step size adaptation. Our experiments were exemplarily performed on the Deutsch-Jozsa problem and the 1-SAT problem. Both problems were made suitable by choosing small instances.

This paper is organized as follows: An overview of related work on automatic quantum circuit design is given in Section 2. Our GP system, with the exception of the selection algorithms, is described in Section 3. Included are some basics of quantum computing, as far as they concern the understanding of this paper. The selection algorithms are separately treated in Section 4, while Section 5 explains the test problems. Section 6 presents experiments and their empirical results, before conclusions are drawn in the last section.

2 Related Work

The idea of using genetic programming to design quantum circuits was discussed first by Williams and Gray in [7]. They demonstrated a GP-based search heuristic which finds a correct decomposition of a given unitary matrix U into a sequence of elementary quantum gate operations. In contrast to subsequent GP schemes for the evolution of quantum circuits, a unitary operator solving a given problem had to be known in advance. Extensive investigations concerning the evolution of quantum algorithms were subsequently done by Spector et al. [8,9,10,11]. In [8] the authors presented three different GP schemes for quantum circuit evolution: standard tree-based GP and both, stack-based and stackless linear genome GP. These were applied to evolve algorithms for Deutsch's two-bit problem, the scaling majority-on problem, the quantum four-item database search problem, and the two-bit-AND-OR problem. Better-than-classical algorithms could be evolved for all but the scaling majority-on problem. In [12] a linear-tree GP scheme was successfully applied to evolve a scalable quantum algorithm for 1-SAT, analogous to Hogg's algorithm. It was also found, that the mixing matrix can be implemented more efficiently by using simple Rx-Gates. In [13] we analyzed the structure of mutation landscapes of small instances of the Deutsch-Jozsa problem using autocorrelation functions and information measures for characterizing their behavior.

3 The GP System

3.1 The Quantum Computer Simulator

An integral component of our GP system is a quantum computer simulator. It simulates quantum algorithms, based on the idealized, noiseless (decoherence-free) quantum circuit model. Briefly speaking, quantum circuits are sequences of unitary matrices, so-called quantum gates, which are applied (i. e. multiplied) to an initial vector (or state) – usually a basis vector of the complex Hilbert space. This operation describes the transformation of a quantum mechanical, physical system – resulting in a final vector.

The dimension of the vector space grows exponentially in the number of quantum bits, qubits for short. A qubit is the basic unit of information in quantum computing. A basic set of quantum gates is sufficient to perform any arbitrary computations, i. e. it is, in the computational sense, universal. Some of the most elementary quantum gates are the Hadamard gate *H,* the single qubit rotation gates $Rx[\phi]$, $Ry[\phi]$, $Rz[\phi]$, the *NOT* and the controlled-*NOT* gate *CNOT*. The rotation gates need an angle parameter. In our system the resolution of the angle parameters was restricted to four bits allowing rotations as multiples of $1/8\pi$. Inputs to quantum algorithms are provided by certain unitary matrices *(INP),* sometimes known as oracles. They may change from instance to instance of a given problem, while the "surrounding" quantum algorithm remains unchanged. Our experiments were conducted using the universal gate set {*H, NOT, CNOT,* $Rx[\phi]$, $Ry[\phi]$, *INP*}. The initial vector is the first of the standard basis vectors.

At the end of a quantum computation always stands a measurement, otherwise we could not gather any information about the system. This can be seen just as a projection onto a basis vector. The probabilities for obtaining a certain basis vector are defined by and calculated from the vector coefficients.

To get a deeper insight into quantum computing and quantum algorithms [14,15,16] might be of interest.

3.2 Individual Structure and Genetic Operators

Because quantum circuits have a natural linear structure[1], it seems obvious to consider a linear genotypic representation for quantum circuits. The length or size of a quantum circuit, i. e. the number of quantum gates, should be limited to a reasonable value. It should be mentioned, that genotype and phenotype correspond directly to each other. The GP algorithm, generational or steady-state GP, decisively depend on the selection strategy that is described in detail in the next section.

In our GP system we exclusively considered mutation operators as evolutionary operators. These operators consist of random *deletion, insertion* and *replacement* of a single quantum gate, random *alteration* of parameters of a single gate and random *swap* of two neighboring quantum gates. Due to our experience, we refrained from using a crossover operator. Our observation showed

[1] Intermediate measurements providing a linear-tree structure [12] are disregarded.

that the efficiency of the search improves. This experience is confirmed by an analysis of the autocorrelation functions of time series obtained from random walks on the mutation [13] and crossover landscapes [17] of certain instances of the Deutsch-Jozsa problem[2]. For both, mutation and crossover landscapes the correlations are rather low. However, for crossover landscapes it is substantially lower (Fig. 1).

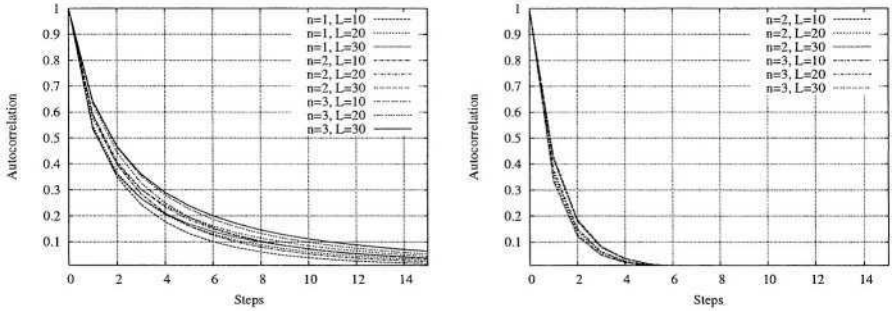


Fig. 1. Autocorrelation functions of mutation (left) and crossover landscapes (right) for the Deutsch-Jozsa problem with different values of n, the number of input bits of some Boolean function f (cf. Section 5.1), and L, the maximum length of a quantum circuit.

4 Selection Strategies

In general, selection is the result of a competition between individuals in a population. The better the individual's fitness in comparison with all other individuals in the population, the higher is its selection probability.

Some popular generational selection algorithms are the $(\mu + \lambda)$ and (μ, λ) selection as well as tournament selection. The (μ, λ) selection was originally used in ES-algorithms [18]. Here, μ parents are allowed to breed λ offspring, out of which the best μ are used as parents for the next generation. A variant of that method is the $(\mu + \lambda)$ selection [19], where offspring and parents participate in the selection. An algorithmic description of ES selection of type $(\mu + \lambda)$ and (μ, λ) follows:

[2] A detailed landscape analysis for landscapes regarding the 1-SAT problem has yet to be done.

1. Generate the initial population; population size is:
   - $\mu + \lambda$ (for $(\mu + \lambda)$ selection)
   - λ (for (μ, λ) selection)
2. Evaluate the fitness of each individual;
3. Select the winners:
   - the best μ individuals in the entire population (for $(\mu + \lambda)$ selection)
   - the best μ newly created individuals or offspring respectively (for (μ, λ) selection)
4. Perform one-point mutation on each winner to generate λ offspring (about λ/μ offspring per winner);
5. Evaluate the fitness of the offspring;
6. Go to step 3 unless termination criteria are met.

For our experiments we extended the (1,10) selection by adding randomness. After fitness evaluation of the offspring, instead of the best offspring, individuals were chosen randomly with a given probability p. In this case step 2 of the algorithm above looks as follows:

2. Let be $u = \text{unif}([0,1])$ uniformly distributed; if $u < p$, then choose the winner randomly from the set of offspring and go ahead to step 4, otherwise evaluate the fitness of each individual;

Furthermore, we used (1,10) selection with self-adaptation [18] of the step-sizes. The self-adaptation routine is due to [20] and looks as follows:

$u = \text{unif}([0,1])$
if $u < 0.5$
then $m* = 1.3$
else $m/ = 1.3$
$n_{mut} = \text{geo}(u, m)$

The parameter n_{mut} determines the step size, i. e. the number of mutations. It is drawn from a geometric distribution. In more detail,

$$\text{geo}(u, m) = \left\lfloor \frac{\ln(1-u)}{\ln(1-p)} \right\rfloor, \quad \text{where} \quad p = 1 - \frac{m}{1 + \sqrt{1+m^2}}.$$

The initial value of m has to be chosen appropriately.

Another important selection mechanism, *tournament selection,* does not belong to generational selection algorithms. Instead, it is based on competition within only a (usually small) subset of the population. A number of individuals taking part in the tournament is selected randomly according to the *tournament size.* In the smallest possible tournament, two individuals compete. The better individuals (the winners) are subject to genetic operators and replace the losers of the tournament. A higher selection pressure can be adjusted by a larger tournament size.

Independent of the selection strategy and the evolutionary process our GP system always stores the fitness of the best individual found so far. These data are the basis of all empirical results.

5 The Test Problems

5.1 The Deutsch-Jozsa Problem

Given a function $f : \{0,1\}^n \to \{0,1\}$ (as a black box) promised to be either *constant,* $f(x) = c, \forall x \in \{0,1\}^n$ and $c \in \{0,1\}$, or *balanced,* i. e. as the result of f 0 occurs as many times as 1, the Deutsch-Jozsa problem is to determine which of the two properties f has. For $n = 1$ the task is also known as Deutsch's problem.

In classical computing f has to be evaluated $2^{n-1}+1$ times in the worst case, in quantum computing a single application of the corresponding input matrix, which is a Boolean function matrix defining f, is sufficient [21] (cf. [15,13] for a detailed description of the input matrix). Using this matrix representation of a Boolean function, $n+1$ qubits are necessary to solve the problem on a quantum computer. The number of Boolean functions being either constant or balanced amounts to $2+\binom{2^n}{2^{n-1}}$. The general quantum algorithm solving the Deutsch-Jozsa problem is discussed in detail in [14,15].

The fitness of a quantum program is measured by its ability to identify the property of a given function f. To obtain the fitness value of a quantum circuit, it is evaluated for each input matrix. Afterwards, the resulting vectors corresponding to balanced functions are compared with those corresponding to the two constant functions. On the basis of measurement probabilities for every base state the determinability of a correct classification is quantified. Misclassifications are penalized. A proper quantum algorithm, solving the Deutsch-Jozsa problem, classifies all functions correctly.

5.2 The 1-SAT Problem

The satisfiability problem (SAT) consists of a logical formula in n variables $v_1, \ldots, v_n$ and the requirement to find an assignment $a = (a_1, \ldots, a_n) \in \{0,1\}^n$ for the variables that makes the formula true. For k-SAT the formula is given as a conjunction of m clauses, where each clause is a disjunction of k literals v_i or $\bar{v_i}$ respectively with $i \in \{1 \ldots n\}$. Clauses which become false for a given assignment are called *conflicts.* The 1-SAT problem for n variables, solved by classical heuristics in $O(n)$ steps, can be solved even faster on a quantum computer. Hogg's quantum algorithm, presented in [22,3], finds a solution in a single search step, using an input matrix, which encodes the number of conflicts for each assignment.

The number of fitness cases (the number of formulas) is $\sum_{k=1}^{n} \binom{n}{k} 2^k$ in total. Each fitness case consists of an input matrix for the formula and the desired

output. The fitness value is determined by the probabilities, that the final measurement will lead to a basis vector which corresponds to an assignment, making the formula true.

It applies to the fitness calculation of both problems, that the fitness function is also standardized and normalized. Moreover, quantum circuits with more than one input gate are strongly penalized.

6 Experiments and Empirical Results

We did experiments for the Deutsch-Joza problem with n=2 (3 qubits) and n=3 (4 qubits) and for the 1-SAT problem with n=3 (3 qubits) and n=4 (4 qubits). In this paper we present only plots of the problem instances with four qubits.

For these problem instances 20 evolutionary runs were performed for each selection method. The results were averaged over the runs. A GP run terminated when the number of single individual (quantum program) evaluations exceeded $1e + 07$ or the fitness of a new best individual under-ran a given threshold, which is close to zero. The length of the quantum circuits was limited to 15 gates. The best quantum circuits for the Deutsch-Jozsa (n=3) and the 1-SAT problem problem (n=4) need about nine gates each using the same gate set. The initial population was randomly created.

The selection strategies and parameter settings used in the GP system are:

- Comma- and Plus-strategies: (1,10), (5,20), (1+10), (5+20);
- (1,10) with step-size adaptation. The value of n_{mut} (the step size) was bounded to 5 mutations. The start value of m is set to 3. This might seem arbitrary, but experiments with larger values showed no or virtually no effect. A more detailed analysis is future work.
- (1,10), combined with additional random selection.
- Tournament selection with different population sizes (100, 500, 1000, 1500, 2000). Tournament size is 2.

Figure 2a shows four graphs illustrating the course of the evolutionary runs for quantum circuits of the Deutsch-Jozsa problem using pure comma- and plus-selection strategies. The (1,10) strategy achieved best results. However, for this strategy 25% of the runs did not lead to an acceptable result. For the other strategies the failure rate was even larger.

Tournament selection can outperform these runs when applied with a suitable population size of about 1500 to 2000 individuals, as shown in Figure 2b. For larger populations the performance decreases more and more. Yet even for tournament selection not every run was successfull within the required number of evaluations.

Performance of the (1,10) selection can be visibly boosted using step size adaptation, which was, all in all, the best selection strategy. In Figure 2c the plot essentially runs below the plots relating to all other selection strategies.

Couriously enough, comparable good performance was achieved by using the (1,10) strategy combined with 10% random selection. Further experiments

demonstrated that even higher percentages of random selection predominate over the pure comma- and plus-strategies. In all runs the resulting quantum circuit was a solution of the test problem. Thus, considering the number of successful runs, this strategy behaves even better than tournament selection on the given problem. Figure 2c illustrates the performance of all tested kinds of selection strategies.

This result confirms our analysis of the mutation landscape of the Deutsch-Joza problem. Because of the high ruggedness, the high modality and only a few paths to global optima the evolution is often caught in local optima. Larger steps or - what seems to help as much - additional randomness appears to compensate for this.

Applied to the 1-SAT problem instance we obtained a completely different result. Here, the best strategies are tournament selection with a population size of about 500 individuals, as shown in Figure 3a, and (1+10) selection, as shown in Figure 3b. Furthermore, Figure 3b illustrates, that plus-strategies perform far better than the comma-strategies. Step size adaptation does not improve evolution and additional randomness rather deteriorates the evolutionary search (Figure 3a). Moreover, for each of the tested selection strategies all 20 evolutionary runs were successful on this problem. This might be a hint that the fitness landscapes of the Deutsch-Jozsa and the 1-SAT problem clearly differ. The mutation landscape for 1-SAT should therefore be smoother and less complex, allowing evolution with comma- or plus-strategy to be more effective and independent of additional randomness and step-size adaptation. It remains to be seen, whether the fitness landscape analysis for the 1-SAT problem will confirm this prediction.

7 Conclusions

In this work we presented a comparison between different selection strategies for evolutionary quantum circuit design: tournament selection for different population sizes, simple comma- and plus-strategies, the (1,10) selection strategy combined with random selection or with step size adaptation. Our results were based on two test problems, the Deutsch-Jozsa and the 1-SAT problem, for which quantum algorithms already exist.

Depending on the population size, tournament selection can be very effective on either test problem. Self-adaptation is similarly effective and enables comma-selection to compete against tournament selection. Other parameter settings may even improve the search speed for the 1-SAT problem.

The comma-strategy with random selection seems to be useful on complex problems with rugged fitness landscapes, difficult for standard evolutionary selection mechanisms. Under those circumstances it can be effective or for smaller populations even better than tournament selection.

Further experiments on the influence of adding randomness for very difficult fitness landscapes would help to judge, whether this is more than just a pleasant side effect. Unfortunately there is only a small set of test problems available

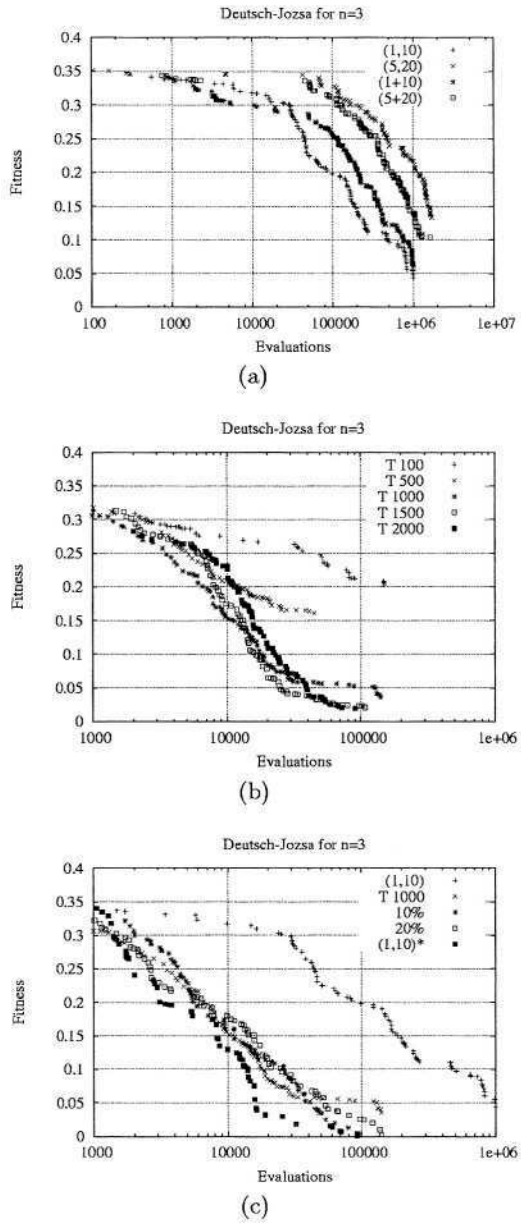
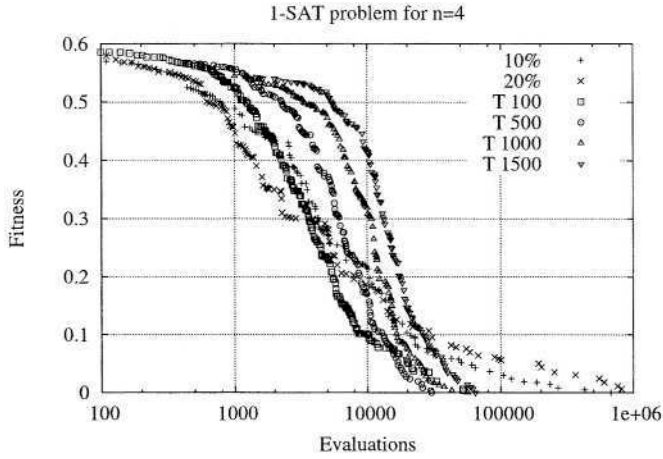
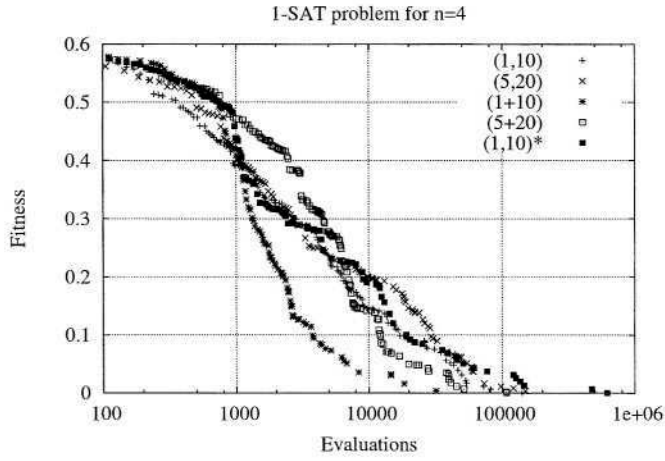


Fig. 2. Different selection strategies for the Deutsch-Jozsa problem with n=3: (a) comma- and plus-strategies; (b) tournament selection for different population sizes; (c) tournament selection, pure (1,10) selection, (1,10) selection with step size adaptation (marked with *) and (1,10) selection plus 10% or 20% random selection respectively. Note the logarithmic scaling on the x-axis.



(a)



(b)

Fig. 3. Different selection strategies for the 1-SAT problem with n=4: (a) Tournament selection for different population sizes and (1,10) selection strategy with 10% or 20% randomness, respectively, (b) (1+10), (5+20), (1,10) and (5,20) ES selection plus (1,10) selection with step size adaptation (marked with *). Note the logarithmic scaling on the x-axis.

and evolution of quantum circuits on larger state spaces is prohibitively time consuming.

Acknowledgement. This work is supported by a grant from the Deutsche Forschungsgemeinschaft (DFG) within GK 726 "Materials and Concepts for Quantum Information Processing". We thank M. Emmerich and C. Richter for numerous discussions and helpful comments.

References

[1] Shor, P.: Algorithms for quantum computation: discrete logarithm and factoring. In IEEE, ed.: Proceedings of the 35th Annual IEEE Symp. on Foundations of Computer Science (FOCS), Silver Spring, MD, USA, IEEE Computer Society Press (1994) 124–134

[2] Grover, L.: A fast quantum mechanical algorithm for database search. In ACM, ed.: Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC), New York, ACM Press (1996) 212–219

[3] Hogg, T.: Solving highly constrained search problems with quantum computers. J. Artificial Intelligence Res. **10** (1999) 39–66

[4] van Dam, W., Hallgren, S.: Efficient quantum algorithms for shifted quadratic character problems (2000)

[5] Hallgren, S.: Polynomial-time quantum algorithms for pell's equation and the principal ideal problem. In ACM, ed.: Proceedings of the 34rd Annual ACM Symposium on Theory of Computing (STOC), New York, ACM Press (2002)

[6] van Dam, W., Seroussi, G.: Efficient quantum algorithms for estimating Gauss sums (2002)

[7] Williams, C., Gray, A. In: Automated Design of Quantum Circuits. Springer, New York (1997) 113–125

[8] Spector, L., Barnum, H., Bernstein, H., Swamy, N. In: Quantum Computing Applications of Genetic Programming. Volume 3. MIT Press, Cambridge, MA, USA (1999) 135–160

[9] Spector, L., Barnum, H., Bernstein, H., Swamy, N.: Finding a better-than-classical quantum AND/OR algorithm using genetic programming. In Angeline, P., Michalewicz, Z., Schoenauer, M., Yao, X., Zalzala, A., eds.: Proceedings of the 1999 Congress on Evolutionary Computation, Silver Spring, MD, USA, IEEE Computer Society Press (1999) 2239–2246

[10] Barnum, H., Bernstein, H., Spector, L.: Better-than-classical circuits for OR and AND/OR found using genetic programming (1999)

[11] Barnum, H., Bernstein, H., Spector, L.: Quantum circuits for OR and AND of ORs. J. Phys. A: Math. Gen. **33** (2000) 8047–8057

[12] Leier, A., Banzhaf, W.: Evolving hogg's quantum algorithm using linear-tree gp. In Cantú-Paz, E., Wilson, S., Harman, M., Wegener, J., Dasgupta, D., Potter, M., Schultz, A., Jonoska, N., Dowsland, K., Miller, J., Foster, J., Deb, K., Lawrence, D., Roy, R., O'Reilly, U.M., Beyer, H.G., Standish, R., Kendall, G., eds.: GECCO-03: Proceedings of the Genetic and Evolutionary Computation Conference, Part I. Volume 2723 of LNCS., Springer (2003) 390–400

[13] Leier, A., Banzhaf, W.: Exploring the search space of quantum programs. In Sarker, R., Reynolds, R., Abbass, H., Tan, K., McKay, B., Essam, D., Gedeon, T., eds.: Proceedings of the 2003 Congress on Evolutionary Computation. Volume I., Piscataway, NJ, USA, IEEE Computer Society Press (2003) 170–177
[14] Gruska, J.: Quantum Computing. McGraw-Hill, London (1999)
[15] Nielsen, M., Chuang, I.: Quantum Computation and Quantum Information. Cambridge University Press, Cambridge, UK (2000)
[16] Hirvensalo, M.: Quantum Computing. Natural Computing Series. Springer, Berlin (2001)
[17] Wagner, G., Stadler, P.: Complex adaptations and the structure of recombination spaces. Technical Report 97-03-029, Santa Fe Institute (1998)
[18] Schwefel, H.P.: Evolution and Optimum Seeking. Sixth-Generation Computer Technology Series. John-Wiley & Sons, New York, USA (1995)
[19] Rechenberg, I.: Evolutionsstrategie '93. Frommann Verlag, Stuttgart, Germany (1994)
[20] Rudolph, G.: An evolutionary algorithm for integer programming. In Davidor, Y., Schwefel, H.P., Männer, R., eds.: Parallel Problem Solving from Nature – PPSN III, International Conference on Evolutionary Computation. Volume 866 of Lecture Notes in Computer Science., Berlin, Springer (1994) 139–148
[21] Cleve, R., Ekert, A., Macchiavello, C., Mosca, M.: Quantum algorithms revisited. Proc. R. Soc. London A **454** (1998) 339–354
[22] Hogg, T.: Highly structured searches with quantum computers. Phys. Rev. Lett. **80** (1998) 2473–2476

Evolving Quantum Circuits and Programs Through Genetic Programming

Paul Massey, John A. Clark, and Susan Stepney

Department of Computer Science, University of York,
Heslington, York, YO10 5DD, UK.
{psm111,jac,susan}@cs.york.ac.uk

Abstract. Spector *et al.* have shown [1],[2],[3] that genetic programming can be used to evolve quantum circuits. In this paper, we present new results in this field, introducing probabilistic and deterministic quantum circuits that have not been previously published. We compare our techniques with those of Spector *et al,* and point out some differences in perspective between our two approaches. Finally, we show how, by using sets of functions rather than precise quantum states as fitness cases, our basic technique can be extended to evolve true quantum algorithms.

Keywords: Quantum computing, genetic programming

1 Introduction

Quantum computing [4],[5] is a radical new paradigm that has the potential to bring a new class of previously intractable problems within the reach of computer science. Harnessing the phenomena of *superposition* and *entanglement,* a quantum computer can perform certain operations exponentially faster than classical (non-quantum) computers. However, devising algorithms to harness the power of a quantum computer has proved extraordinarily difficult. Eighteen years after the publication of the first quantum algorithm in 1985 [4], almost all known quantum programs are based on two fundamental techniques: Shor's Quantum Fourier Transform [6],[7] and Grover's quantum search algorithm [8].

Spector *et al.* [1],[2],[3] show how genetic programming (GP) might be used to evolve quantum programs and circuits. The key features of their approach are:

- A *second order encoding:* individuals in the GP population are LISP programs that generate quantum circuits. The LISP *alleles* available to the GP software include functions to allow iteration and arithmetic, as well as functions to generate quantum gates.
- An emphasis on finding *probabilistic* solutions: quantum programs give the correct answer with a probability of at least 50% for every fitness case tested.
- A fitness function made up of three components: *hits, correctness* and *efficiency.* This function is described later, in the context of our own work.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 569–580, 2004.

Spector *et al* evolved a quantum circuit that solves Deutsch's Problem [4] on three qubits, with an error probability of less than 0.3, and a quantum circuit that solves the database search problem on five qubits, for the special case of four marked states (effectively an implementation of a special case of Grover's algorithm [8]).

Our work also involes using GP to evolve quantum circuits, but with a different emphasis from Spector *et al.* We use the following terminology in this paper:

Quantum circuit (or 'quantum gate array'): a collection of quantum logic gates that can be applied in sequence to a specified quantum system.

Quantum program: a set of instructions that, when executed, generates one or more quantum circuits. The program may include constructs such as iteration and branching functions as well as functions to generate particular quantum gates. In the language of GP, a quantum program is the *genotype* that can be decoded to produce a quantum circuit *phenotype.*

Quantum algorithm: a *parameterisable* quantum program that, as the value of the parameter(s) are altered, generates quantum circuits to solve a large number of different problem instances, perhaps across different sizes of quantum system.

Our terminology differs from normal non-quantum usage, where, for example, a (compiled) 'program' is software and executed on real circuitry, and where an 'algorithm' is generally a machine independent recipe that would be implemented by a program. (Both algorithm and program might be capable of handling various system instances). We use a general purpose circuit generating language, abstracting over the implementation of the circuit in hardware, to express our 'programs'. Since we have at present no general purpose quantum computer to target, this seems appropriate. Also, the ability to express parametrisable programs in the same language, allowing abstraction over different system instances, is motivated by our real goal: the use of GP-related methods to discover new ways of solving problems by quantum means.

2 Evolving Quantum Circuits

2.1 The Software

Our research has been conducted using three successive iterations of a tool called *Q-PACE* (*Q*uantum *P*rograms *A*nd *C*ircuits through *E*volution). The original Q-PACE suite is now obsolete. Q-PACE II and Q-PACE III are genetic programming suites written in C++, incorporating a number of classes and functions from Wall's GALib library [11].

Q-PACE II uses a *first order encoding:* each individual is a quantum circuit, not a quantum program (as per the definitions earlier). In practice, each individual is a tree of quantum gates; the tree is traversed to generate the sequence the quantum gates.

Q-PACE III uses a *second order encoding* similar to that of Spector *et al:* each individual is a quantum program that generates one or more actual quantum

circuit(s). In practice, each individual is a tree of statements, with each statement being either a function (including functions to allow iteration and functions to generate quantum gates) or a terminal symbol (a number, representing the qubit to be operated on, the number of iterations a loop should run for, etc.)

Both Q-PACE II and Q-PACE III use tournament selection and a "subtree swap" crossover operator. They both incorporate various mutation operators, including subtree insertion, subtree deletion, and subtree replacement. In Q-PACE III, constraints are imposed on the mutation operators to ensure that only syntactically valid quantum programs are produced as GP individuals.

2.2 Fitness Functions

By default, both Q-PACE II and Q-PACE III evaluate the fitness of candidate quantum programs and circuits using the following method:

- Initialisation:
  - Create a set V_I of input state vectors that span the space of all possible inputs. Each member of V_I acts as a *fitness case* for the problem under test.
  - Create a set V_T of *target vectors,* the desired results for each fitness case.
- Evaluation:
  - Apply the candidate individual to each fitness case, to produce a set of *result vectors* V_R.
  - Compare each member of V_R with the corresponding member of V_T. The chosen means of comparison defines the specific *fitness function* for the particular problem under test.

When evolving *deterministic* quantum programs or circuits (those that give the correct answer with probability 1, 100% of the time), we use the sum of the magnitudes of the differences of the probability amplitudes:

$$f = \sum_i ||V_{T_i} - V_{R_i}|| \tag{1}$$

When trying to evolve *probabilistic* quantum programs or circuits, we use:

$$f = hits + correctness + efficiency \tag{2}$$

This follows the lead of Spector *et al.* [1],[2],[3]. The *hits* component is the total number of fitness cases used minus the number of fitness cases where the program produces the correct answer with a probability of more than 0.52 (following Spector, chosen to be far enough away from 0.5 to be sure it is not due to rounding errors). The *correctness* component is defined as:

$$correctness = \frac{\sum_{i=1}^{n} \max(0, error_i - 0.48)}{\max(hits, 1)} \tag{3}$$

Because it is desirable for the fitness function to focus on attaining probabilistically correct answers to all fitness cases, rather than simply improving the probability of success in those fitness cases where it is already good enough (e.g. from a 55% success rate to a 60% success rate), errors smaller than 0.48 are ignored. Also, it is desirable that reasonably fit programs are compared primarily with respect to the number of fitness cases they produce a (probabilistically) correct answer for, and only secondarily with respect to the magnitudes of the errors of the incorrect cases, the 'pure' *correctness* term is divided by *hits* (unless *hits* < 1) before being used in the fitness function.

The *efficiency* is the number of quantum gates in the final solution, divided by a large constant. Therefore, efficiency has a very small effect on the overall fitness of the solution, until programs are evolved that solve all fitness cases, at which point the other two terms become zero and the efficiency dominates. The overall effect is that the search initially concentrates on finding probabilistic solutions to the problem, and then tries to make those solutions more efficient, in terms of the number of quantum gates used. No effort is wasted on trying to make the solutions more accurate (*i.e.* increase the probability of them correctly giving the answer).

2.3 The Evolution of Deterministic Quantum Circuits

Q-PACE II evolved a deterministic full adder circuit using simple and controlled versions of the *N* and *H* gates (see Appendix A), and the non-unitary zeroing gate *Z*. Q-PACE II found 2 distinct solutions to the problem; the more efficient of which is identical to that of Gossett [15]. We also applied Q-PACE II to a number of more challenging problems, including multiplication modulo n and exponentiation modulo n. We also attempted to evolve simple circuits for quantum arithmetic using only very basic quantum transformations as alleles. However, despite many variations of problem specification and fitness function, large population sizes, and runs of thousands of generations, we were unable to evolve exact solutions to these problems.

GP is intrinsically suited to finding good *approximate* solutions, as opposed to *exact* deterministic solutions. We modified Q-PACE II to use a two stage search strategy: GP to evolve candidate solutions with a very good, but not perfect, fitness score, then hill-climbing on these good solutions to look for a nearby exact answer. However, deterministic solutions to these harder problems continued to be elusive. We believe that this is due to the discontinuity of the search space: exact solutions lie some distance from good approximate solutions, so changing a single quantum gate in a good solution is of no use in converging to an exact solution. Therefore, we changed the form of the problem to be solved to one we believe more suited to GP.

2.4 Probabilistic Quantum Circuits

Using the probabilistic fitness function(3), Q-PACE II can find probabilistic solutions to problems for which it was unable to find a deterministic solution. It

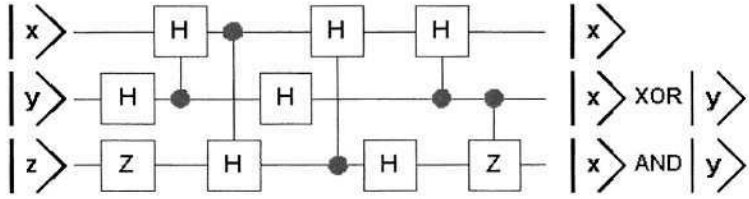


Fig. 1. A probabilistic half-adder

found a probabilistic half-adder on 3 qubits using only the H gate and the non-unitary zeroing gate Z (together with their controlled equivalents). The problem is defined as $|x, y, z\rangle \rightarrow |x, x\,\mathrm{XOR}\,y, x\,\mathrm{AND}\,y\rangle$, where $|x\,\mathrm{XOR}\,y\rangle$ is the sum bit and $|x\,\mathrm{AND}\,y\rangle$ the carry bit.

Q-PACE II evolved the circuit shown in figure 1. It has the following probabilistic solution to the problem:

initial state		correct answer	prob of ending in state							
			$\|000\rangle$	$\|001\rangle$	$\|010\rangle$	$\|011\rangle$	$\|100\rangle$	$\|101\rangle$	$\|110\rangle$	$\|111\rangle$
$\|000\rangle$	$\rightarrow$	$\|000\rangle$	**0.53**	0.22	0	0	0	0.09	0.14	0
$\|001\rangle$	$\rightarrow$	$\|000\rangle$	**0.53**	0.22	0	0	0	0.09	0.14	0
$\|010\rangle$	$\rightarrow$	$\|010\rangle$	0	0.05	**0.61**	0	0	0.09	0.24	0
$\|011\rangle$	$\rightarrow$	$\|010\rangle$	0	0.05	**0.61**	0	0	0.09	0.24	0
$\|100\rangle$	$\rightarrow$	$\|110\rangle$	0.09	0.04	0.03	0	0	0.02	**0.82**	0
$\|101\rangle$	$\rightarrow$	$\|110\rangle$	0.09	0.04	0.03	0	0	0.02	**0.82**	0
$\|110\rangle$	$\rightarrow$	$\|101\rangle$	0	0.30	0.10	0	0.02	**0.53**	0.04	0
$\|111\rangle$	$\rightarrow$	$\|101\rangle$	0	0.30	0.10	0	0.02	**0.53**	0.04	0

The most appropriate use of probabilistic circuits remains to be determined. It is possible to exploit probabilistic biases by repeated application of a program to a particular problem instance. We can also imagine their use in more 'soft' systems, where a good deal of noise in results is expected as the computation evolves. This is an open research issue.

3 Evolving Q-MAX Algorithms Using 'Functional Fitness Cases'

To evolve more advanced quantum programs (and ultimately quantum algorithms), it has proved necessary to replace the basic technique of calculating the fitness, as described above, with a different approach.

To illustrate this approach, consider that we are given a set of functions on x, where x is an integer variable constrained to a certain domain and range (e.g. [0..3] $\rightarrow$ [0..3]), such as the following:

$f_1(x) = 7x \bmod 3$
$f_2(x) =$ the permutation $\{0,1,2,3\} \rightarrow \{2,1,3,0\}$
$f_3(x) = 3 - x$
$f_4(x) = 0$ for all x (i.e. $0 \rightarrow 0, 1 \rightarrow 0, 2 \rightarrow 0, 3 \rightarrow 0$)

For any of these functions, it would be a reasonable challenge to seek a quantum program that finds the *maximum value* for that function (in other words, "which input value x gives the largest output value y"?). For the function $f_3(x)$, for example, the maximum value is 3, which occurs when $x = 0$. A more difficult but much more useful challenge would be to evolve a quantum algorithm for finding the maximum value of *any* [0..3] $\rightarrow$ [0..3] function. Such an algorithm could be given any of the functions listed above and would return the correct maximum value for that function.

3.1 The Specific Case

Before progressing to the general case, first consider how we might solve the following basic problem:

Given a permutation function f(x) which operates over the integer range [0..3], *use a suitable encoding to evolve a quantum program U that returns the value of x that gives the maximum value of f(x).*

Before we can attempt to evolve a candidate *U,* we first need to create a quantum state to encode $f(x)$. As both x and $f(x)$ range from [0..3], we can encode $f(x)$ in a 4 qubit quantum state, with the first 2 qubits encoding x and the remaining 2 qubits encoding $f(x)$. For example, the state vector Y (fig 6) encodes the function $f_2(x)$ defined above.

$$Y = \begin{array}{c} x\ f(x) \\ 00\ 00 \\ 00\ 01 \\ 00\ 10 \\ 00\ 11 \\ 01\ 00 \\ 01\ 01 \\ 01\ 10 \\ 01\ 11 \\ 10\ 00 \\ 10\ 01 \\ 10\ 10 \\ 10\ 11 \\ 11\ 00 \\ 11\ 01 \\ 11\ 10 \\ 11\ 11 \end{array} \begin{pmatrix} 0 \\ 0 \\ a \\ 0 \\ 0 \\ b \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ c \\ d \\ 0 \\ 0 \\ 0 \end{pmatrix} \qquad R = \begin{array}{c} x\ f(x) \\ 00\ 00 \\ 00\ 01 \\ 00\ 10 \\ 00\ 11 \\ 01\ 00 \\ 01\ 01 \\ 01\ 10 \\ 01\ 11 \\ 10\ 00 \\ 10\ 01 \\ 10\ 10 \\ 10\ 11 \\ 11\ 00 \\ 11\ 01 \\ 11\ 10 \\ 11\ 11 \end{array} \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \\ i \\ j \\ k \\ l \\ m \\ n \\ o \\ p \end{pmatrix}$$

The non-zero probability amplitudes a, b, c and d represent the desired transitions $f(0) = 2$, $f(1) = 1$, $f(2) = 3$ and $f(3) = 0$ respectively. All other transitions

are disallowed in $f_2(x)$, and so are given an associated probability amplitude of zero in the state vector. To be a valid quantum state, $|a|^2 + |b|^2 + |c|^2 + |d|^2 = 1$. For simplicity, we take these amplitudes as equal: $a = b = c = d = \frac{1}{\sqrt{4}} = \frac{1}{2}$

Having created this state vector Y to act as our initial quantum state, we then apply our candidate quantum program U to it, thus creating a state vector $R = UY$. At the end of this computation, we simulate measuring the first two qubits of the quantum system to read out the 'answer', the value of x that corresponds to the maximum value of $f(x)$. In the general case, we end up with a state vector R shown in fig 6.

Observing the full state causes it to collapse to one of the 16 eigenstates. We see that the first two qubits are observed with particular values with probability equal to the sum of probabilities of the eigenstates consistent with the values. Thus, a state with $x = 00$ is observed with probability $q^2 = |a|^2 + |b|^2 + |c|^2 + |d|^2$; a state with $x = 01$, with probability $r^2 = |e|^2 + |f|^2 + |g|^2 + |h|^2$; a state with $x = 10$, with probability $s^2 = |i|^2 + |j|^2 + |k|^2 + |l|^2$; and a state with $x = 11$, with probability $t^2 = |m|^2 + |n|^2 + |o|^2 + |p|^2$.

We then compute the probabilities $|q|^2$, $|r|^2$, $|s|^2$ and $|t|^2$, which represent the probability of measuring x to be 0, 1, 2 and 3 respectively. Suppose, for one particular candidate U, these probabilities are found to be 0.2, 0.3, 0.5 and 0. By looking at these probabilities, we can see that this candidate U gives the correct answer ($x = 2$) 50% of the time. Our goal is to find a candidate program U which somehow amplifies the probability of measuring the correct value of x, and decreases the probability of measuring an incorrect value of x.

Given a state vector R, how should we actually assess the fitness of a candidate U? The most obvious fitness measure is simply the probability of the correct answer, in this case, $|s|^2$ (in practice, using $1 - |s|^2$ would be more useful, as a fitness of 0 would then correspond to an exact solution). A slightly more advanced fitness measure would allow U to be a probabilistic program (i.e. a U that gave $|s|^2 > 0.5$ would be regarded as a solution to the problem under test). Alternative fitness functions could be used that gave some credit for having "near miss" answers (if we are trying to evolve a MAX algorithm, a candidate solution which could consistently return a value of x that gives a large but not maximal value of $f(x)$ might actually be sufficient).

3.2 The General Case

So far we have shown only how we can evolve a quantum program U to solve one specific problem instance. How might we generalise this to evolve a true quantum MAX algorithm, capable of returning the value of x that gives the maximum value of $f(x)$ for any function $f(x)$ that is supplied as input?

To achieve this goal, we create a large number of functions to act as fitness cases. Each candidate U operates on all fitness cases, and its overall fitness is the sum of the fitness scores it is awarded for each f.

3.3 Evolution of a Probabilistic MAX Algorithm for Permutation Functions

Using this strategy, we have been able to evolve (using Q-PACE III) a number of probabilistic quantum programs which, when given a number of suitably encoded $[0..3] \rightarrow [0..3]$ permutation functions, returned for every one of these permutation functions (with a probability > 0.5) the value of x that gave the maximum value of $f(x)$ for that function. We refer to this problem as the "PF MAX" problem for short. Ultimately, we evolved a program that solved the problem for all 24 possible $[0..3] \rightarrow [0..3]$ permutation functions, as is shown below.

We consider only $[0..3] \rightarrow [0..3]$ permutation functions because they have the nice properties of being discrete, one-to-one, easy to generate, and capable of being encoded by a 4 qubit system.

There are 4! = 24 different $[0..3] \rightarrow [0..3]$ permutation functions. Our default approach was to give Q-PACE III a subset of these 24 functions to act as fitness cases. If Q-PACE III evolved a MAX program that worked for all the fitness cases, we would then test it on the other functions in the set to determine the generality of the evolved solution.

3.4 The "PF MAX 1" Program

The first useful program generated by Q-PACE III (which we call PF MAX 1) was evolved using the following 8 fitness cases (expressed as permutations): {(3,1,0,2), (0,2,3,1), (3,0,1,2), (1,2,3,0), (3,2,0,1), (2,3,0,1), (2,0,1,3), (2,1,3,0)}. PF MAX 1 does the following:

fitness case	correct answer	prob of ending in state $\|00\rangle$	$\|01\rangle$	$\|10\rangle$	$\|11\rangle$
(3,1,0,2)	$\|00\rangle$	**0.53**	0.22	0.03	0.22
(0, 2, 3, 1)	$\|10\rangle$	0.03	0.22	**0.53**	0.22
(3,0,1,2)	$\|00\rangle$	**0.53**	0.22	0.03	0.22
(1,2,3,0)	$\|10\rangle$	0.03	0.22	**0.53**	0.22
(3, 2, 0, 1)	$\|00\rangle$	**0.53**	0.22	0.03	0.22
(2, 3, 0, 1)	$\|01\rangle$	0.22	**0.53**	0.22	0.03
(2,0,1,3)	$\|11\rangle$	0.22	0.03	0.22	**0.53**
(2, 1, 3, 0)	$\|10\rangle$	0.03	0.22	**0.56**	0.19

PF MAX 1 is a probabilistic solution to all 8 of the fitness cases used; what effect does PF MAX 1 have on the 16 other permutation functions in the set? For 20 out of the 24 possible permutation functions, PF MAX 1 gives the correct answer with a probability of more than 0.5; for the other 4 fitness cases, PF MAX 1 gives the correct answer with a higher probability than any given incorrect answer. Thus PF MAX 1 is a true MAX algorithm for $[0..3] \rightarrow [0..3]$ permutation functions, that "works" on all 24 of these functions. Although evolved from only 8 fitness cases, the resulting PF MAX 1 is much more general.

The circuit generated by PF MAX 1 (after removing by hand 5 gates that have no effect) is shown in figure 2.

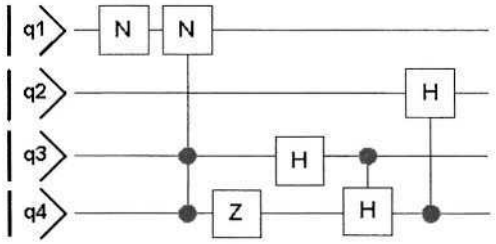


Fig. 2. PF MAX 1 circuit

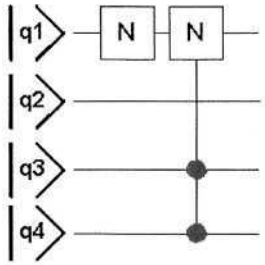


Fig. 3. PF MAX 3 circuit

3.5 Changing the Acceptance Criterion: "PF MAX 3"

The existence of PF MAX 1 suggests a quantum program might exist that would give the correct solution with a probability of > 0.5 for all 24 fitness cases. However, repeated experiments failed to evolve a program that met these parameters. Yet circuits that give the correct answer to certain fitness cases with a probability of *exactly* 0.5 were commonly produced.

So we attempted to solve the PF MAX problem for all 24 possible fitness cases with a relaxed acceptance criterion of > 0.4. When Q-PACE III was run with this relaxed acceptance criterion, it evolved a quantum program which generated a single quantum circuit that, for each of the 24 fitness cases, has a probability of 0.5 of returning the correct answer (the probabilities of returning incorrect answers are 0.25 or zero). So the quantum circuit implements a probabilistic MAX function that has twice the probability of "guessing".

The circuit generated by PF MAX 3 (after removing by hand several gates that have no effect) is shown in figure 3.

This seems remarkably simple. What is happening here? After consideration we can see that the system is actually exploiting the initial set-up very efficiently. Suppose for example, that the maximum occurs at $x = 00$. Then $|0011\rangle$ has amplitude $\frac{1}{2}$ (corresponding to probability $\frac{1}{4}$), and $|0000\rangle$, $|0001\rangle$ and $|0010\rangle$ all have amplitude of 0. Now consider $x = 10$. We must have $f(10) = 00$, $f(10) = 01$, or $f(10) = 10$ since the maximum is already reached uniquely by $f(00) = 11$.

Suppose $f(10) = 00$. Then the state $|1000\rangle$ has amplitude $\frac{1}{2}$, while $|1001\rangle$, $|1010\rangle$ and $|1011\rangle$ all have amplitudes of 0. The application of the CCNOT operation transforms $|1000\rangle$ to $|0000\rangle$ with amplitude $\frac{1}{2}$ whilst $|0011\rangle$ remains unaltered with amplitude $\frac{1}{2}$. We now have two eigenstates with $x = 00$ and amplitude $\frac{1}{2}$: $|0000\rangle$ and $|0011\rangle$. So the probability of now observing one of these eigenstates is $\frac{1}{4} + \frac{1}{4} = \frac{1}{2}$. This is a better than classical algorithm. More generally, if $f(x) = 11$ then we can consider the states $|x\ 11\rangle$ and $|x'\ f(x')\rangle$ (where x' is obtained from x by flipping the first bit) to obtain a similar result. Furthermore, there would appear to be an obvious generalisation to n qubits: let the second negation on qubit 1 be controlled by all the qubits of $f(x)$.

4 Conclusions

The programs and circuits presented in this paper have become increasingly capable, with PF MAX 3 in particular solving quite a general problem, with its better-than-classical result generalising to n qubits. Using GP to produce probabilistic circuits, that give the right answer with high probability for all fitness cases, seems a more practical approach than requiring fully correct deterministic circuits.

In future work, we will build on our results to evolve parameterisable quantum programs, which, when given different parameters, will be able to solve problems such as the PF MAX problem for different classes of function and across different system sizes. If such things can be evolved, they will prove that genetic programming can have a part to play in discovering new, useful quantum algorithms, and help break the bottleneck that currently exists in quantum algorithm discovery.

Citations "*quant-ph/yymmxxx*" are available on the Internet from the Los Alamos National Laboratory pre-print server at http://www.arXiv.org

References

[1] L. Spector, H. Barnum, H. Bernstein, "Genetic Programming for Quantum Computers", in *Genetic Programming 1998,* Morgan Kaufmann, 1998.

[2] L. Spector, H. Barnum, H. Bernstein, N. Swamy, "Quantum Computing Applications of Genetic Programming", in *Advances in Genetic Programming 3,* MIT Press, 1999

[3] L. Spector, H. Barnum, H. Bernstein, N. Swamy, "Finding a Better-than-Classical Quantum AND/OR Algorithm using Genetic Programming", in *Congress on Evolutionary Computation,* 1999.

[4] D. Deutsch, "Quantum Theory, the Church-Turing Thesis, and the Universal Quantum Computer", *Proc. Royal Society of London,* series A, vol. 400, p97 1985.

[5] E. Rieffel and W. Polak, "An Introduction to Quantum Computing for non-Physicists", 1998. *quant-ph/9809016.*

[6] P. W. Shor, "Algorithms for Quantum Computation : Discrete Logarithms and Factoring", *Proc. 35^{th} IEEE Symposium on the Foundations of Computer Science,* p124, 1994.
[7] P. W. Shor, "Polynomial Time Algorithms for Prime-Factorisation and Discrete Logarithms on a Quantum Computer", SIAM Journal of Computing, **26,** p1484, 1997
[8] L. Grover, "A Fast Quantum Mechanical Algorithm for Database Search", *Proceedings of the 28^{th} ACM STOC,* p212, 1996.
[9] J. R. Koza, *Genetic Programming,* MIT Press, 1992.
[10] J. R. Koza, *Genetic Programming II,* MIT Press, 1994.
[11] M. Wall, "GALib, a C++ Library for Genetic Algorithms", available from http://lancet.mit.edu/ga/
[12] A. Ekert, P. Hayden & H. Inamori, "Basic Concepts in Quantum Computation", 2000. *quant-ph/0011013.*
[13] T. Toffoli, "Reversible Computing", in *Automata, Languages and Programming,* LNCS 84, Springer, 1980.
[14] E. Fredkin & T. Toffoli, "Conservative Logic", *Intl. J. Theoretical Phys,* **21,** p219, 1982.
[15] P. Gossett, "Quantum Carry-Save Arithmetic", 1998. *quant-ph/9808061.*

A Quantum Gates Used

In this paper, we use only three basic types of quantum gate:

symbol	name	specification
$N(x)$	NOT gate	$\|0\rangle \rightarrow \|1\rangle, \|1\rangle \rightarrow \|0\rangle$
$H(x)$	Hadamard gate	$\|0\rangle \rightarrow \frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$ $\|1\rangle \rightarrow \frac{1}{\sqrt{2}}(\|0\rangle - \|1\rangle)$
$Z(x)$	Non-unitary Zeroing gate	$\|0\rangle \rightarrow \sqrt{(\|\|0\rangle\|^2 + \|\|1\rangle\|^2)}$ $\|1\rangle \rightarrow 0$

These are all single-qubit gates, and take a single parameter x representing the target qubit (e.g. $N(2)$ would represent a NOT gate being applied to the second qubit in a system). However, all three gates can have *controlled* equivalents. Thus, the controlled not applies N to a target qubit only if the control qubit is set. Thus, with the first qubit being the control and the second qubit being the target $|00\rangle$ and $|01\rangle$ remain unchanged but $|10\rangle$ and $|11\rangle$ are transformed to $|11\rangle$ and $|10\rangle$ respectively. This is typically written C_{NOT} (or C_N). C_H and C_Z gates are similar: for example, the gate $C_H(2,4)$ would be a controlled Hadamard gate with the second qubit in a quantum system acting as the control qubit and the fourth qubit acting as the target qubit.

Sometimes, it is useful to use gates that have more than one control bit. These gates only perform their action if all control bits are in state $|1\rangle$, otherwise they have no effect. For example, the "controlled controlled NOT" gate $CCN(1, 2, 3)$ negates its target qubit (qubit 3) if both control qubits (1 and 2) are in state $|1\rangle$. Controlled controlled NOT gates are usually called *Toffoli gates,* after their inventor [13],[14].

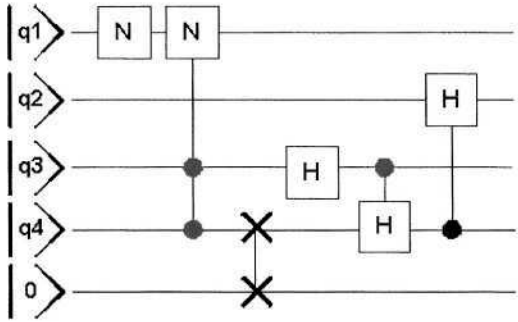


Fig. 4. A circuit equivalent to that of figure 2, with the zeroing gate replaced by an ancilla qubit and with x—x representing a "swap" gate

B Comments on the Use of the Zeroing Gate

Here we comment on the use of the non-unitary zeroing gate Z in the various circuits in this paper. The GP suite was given the Z gate to allow qubits to be initialised to the state $|0\rangle$, as occurs in many papers in the literature. However, the GP suite generalised beyond this and made use of the Z gate in unexpected places, such as in mid-circuit (figure 2).

What does it *mean* to force a qubit to zero in the middle (or at the end) of a circuit? The Z gate acts rather like a standard measurement gate; the key difference is that a measurement gate collapses the quantum state vector into one of its component quantum states at random, whereas the Z gate forces the qubit into the state $|0\rangle$. However, both types of gates have the same side-effect: reducing the number of possible positions in the state vector with a non-zero probability amplitude.

How to implement Z? One way would be to perform the measurement, and proceed only if it were zero. This would reduce the probabilities of correct answers unacceptably. An alternative way is to regard it as a swap operation with an ancilla zeroed qubit. So the circuit illustrated in figure 2 is equivalent to the circuit shown in figure 4.

On Multi-class Classification by Way of Niching

A.R. McIntyre and M.I. Heywood

Faculty of Computer Science, Dalhousie University
Halifax, Nova Scotia, Canada B3H 1W5
{armcnty, mheywood}@cs.dal.ca

Abstract In recent literature, the niche enabling effects of crowding and the sharing algorithms have been systematically investigated in the context of Genetic Algorithms and are now established evolutionary methods for identifying optima in multi-modal problem domains. In this work, the niching metaphor is methodically explored in the context of a simultaneous multi-population GP classifier in order to investigate which (if any) properties of traditional sharing and crowding algorithms may be portable in arriving at a naturally motivated niching GP. For this study, the niching mechanisms are implemented in Grammatical Evolution to provide multi-category solutions from the same population in the same trial. Each member of the population belongs to a different niche in the GE search space corresponding to the data classes. The set of best individuals from each niche are combined hierarchically and used for multi-class classification on the familiar multi-class UCI data sets of Iris and Wine. A distinct preference for Sharing as opposed to Crowding is demonstrated with respect to population diversity during evolution and niche classification accuracy.

1 Introduction

Genetic Programming is increasingly being applied to data mining tasks [1], with advances proposed for encouraging simple solutions [2] and the provision of confidence intervals as well as class labels [3]. One of the major impediments to the wide spread utilization of GP methods in real world data mining applications, however, is still associated with scalability. That is, given the computational overheads associated with the inner loop of GP, how may the algorithm be efficiently applied to solve multi-class problems or problems based on exceptionally large datasets. In this work we are interested in the case of the multi-class problem. That is, how to evolve multi-class classifiers from single GP trials, where the standard approach is to evolve separate binary classifiers for each class [4]. Dividing this job across a Beowulf cluster, for example, provides a brute force method to arrive at a classifier for an arbitrary number of classes on the same dataset.

In this work we investigate the provision of multi category classifiers in a single GP trial through the use of niching metaphors, using algorithms derived from the Genetic Algorithm literature [5-8]. That is, each GP run provides a set of binary classifiers solving the multi-class problem as a group. To do so, this paper investigates the feasibility of a crowding / sharing hybrid algorithm that naturally extends the basic GP framework to a multi-class classifier. A comparative analysis of the niching properties of the GP model with the GA crowding experiments of Mahfoud [5] is provided

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 581–592, 2004.

along with an extension to a hybrid model using deterministic crowding and dynamic sharing [8]. We demonstrate that such a scheme is indeed capable of yielding multi-category classifiers with properties broadly in line with those originally observed by Mahfoud.

The remainder of this paper is organized as follows: Section 2 provides the background information and previous work in this area, Section 3 describes the basic objective and the methodology of this research. Section 4 summarizes the results and provides an analysis of the study and section 5 concludes the paper.

2 Background

Genetic Programs typically identify one single 'super individual' representing a single peak in the search space. Moreover, the standard Canonical Genetic Algorithm (GA) will converge to a single peak even in multimodal domains characterized by peaks of equivalent fitness. This effect is known as genetic drift. Mechanisms that allow GAs to evolve solutions that do not only converge to a single peak are known here as 'diversity mechanisms'.

The multi-modal problem has been of interest within the context of optimization problems for a considerable period of time, with modifications proposed for both Evolutionary Programming [9] and Genetic Algorithms [5-8]. In the case of this work, multimodal GAs provide a natural path for incorporating diversity mechanisms into the GP multi-category classification context. Traditional mechanisms considered as the basis for this work include Deterministic crowding [7], and Sharing [6, 8].

Crowding algorithms were originally defined in terms of a form of tournament. However, rather than tournaments taking place to define individuals to reproduce they were used to define the individuals to be replaced; the number of individuals taking place in such a tournament being denoted the crowding factor [5]. The concept of crowing was thoroughly reviewed and revised by Mahfoud, when a series of experiments were performed on a set of basic multimodal problems, resulting in the definition of a 'Deterministic Crowding' algorithm [7]. These experiments form the basis for the first part of the study conducted here with multi-category classification problems, Section 3 algorithms A1 to A5. The method does however have some shortcomings. Specifically, population drift on more taxing problems is still observed [8]. As a consequence, we extend our review to include the concept of sharing, and demonstrate this within the context of GP, Section 3 algorithm A6.

2.1 Grammatical Evolution

The following study is performed using Grammatical Evolution (although results are not specific to the type of Genetic Programming employed). Grammatical Evolution (GE) permits automatic and language independent evolution of programs of arbitrary complexity [10]. To some extent, there are obvious similarities between GE and GP. However GE does not operate directly on the programs themselves (as in traditional GP), rather the programs are stored as a series of Backus-Naur form (BNF) grammar rule selectors, which are in turn indirectly represented by a fixed length binary string individual. In this sense, GE is similar to GA and consequently permits the use of

simple GA search operators. Based on the findings of Kishore *et al.* regarding their success with minimal functional sets [4], we constrain the expression of our GE individuals to the following simple BNF grammar (for an *i*-input classification problem):

```
code : exp
exp  : var | const | exp op var | exp op exp
op   : + | - | * | /
var  : x1 | x2 | … | xi
const: 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

The fitness of an individual for a given class is calculated by proportionally incrementing the total fitness score (starting from zero) for each correct training set classification (binary class of 1 or 0, where the increment is weighted to equate the value of in and out of class exemplars when they are unequal in number). The details of GE and further comparisons with traditional evolutionary computing methods have been considered elsewhere [10] and will not be re-examined in this paper.

2.2 Classification and GP

A distinction is typically made between binary classifiers and multi-class (or multi-category) classifiers. A multi-class classifier predicts a single label for each example from a set of many mutually exclusive candidate labels, while a binary classifier predicts each example as either in or out of class (in other words, the prediction is binary).

GP has been successfully applied to binary and, more recently, to multi-class classifiers. However, research has only recently begun to address the multi-class nature of the problem directly [11]. To do so, fitness is explicitly evaluated with respect to each class and the individual assigned to the fittest category. Although providing a deterministic solution to the problem for the dataset considered, no investigation is made of the robustness of the algorithm (e.g. sensitivity to drift, dynamics of niche maintenance). The objective of this work is to investigate the multi-class problem from a wider perspective of a niche metaphor as developed by the Genetic Algorithm literature.

3 Methodology

The approach taken for *k*-class classification is to formulate the problem as *k* binary classification problems and hierarchically process the examples through each classifier. Using this approach allows us to separately evolve the *k* binary classifiers where each classifier can distinguish a given example as in or out of its particular class. The cumulative effect of hierarchically combining the *k* classifiers is to discriminately place each example into the class of the highest accepting classifier, while those examples that are deemed out of class for all classifiers are placed into a collection to be reclassified or labeled as indeterminate or other.

In these experiments, we present a set of necessarily modified versions of Mahfoud's crowding algorithms, A1-A5 that have been re-designed for multi-

population GP (specifically a GE multi-class classifier). A0 has been omitted due to its reliance on genotypic similarity metrics, which has no obvious, meaningful GP counterpart. Finally, algorithm A6 is introduced to investigate the potential for sharing based maintenance of niches where this has the potential to provide more robust niche maintenance [8].

3.1 Establishing Phenotypic Similarity

The main modification for these experiments was in defining the notion of phenotypic similarity between individuals. For Mahfoud's experiments, phenotypic similarity takes the form of a distance calculation between pairs of real-valued, decoded parameters (the expression of phenotype in a canonical GA). In the case of this work, we are interested in comparing the effectiveness of programs at solving a series of problems (each of which correspond to a niche, or pattern class, where an individual may acquire fitness).

Our design choice was to establish the concept of phenotypic similarity by evaluating each individual for each problem (niche), so that an individual effectively has k fitnesses (corresponding to a k niche problem). Accordingly, each individual in the population can be considered a point in k dimensional space. This approach was chosen because it seems to provide a straightforward extension of Mahfoud's phenotypic similarity measure, where we are simply extending the concept of phenotype into problem-specific fitness, giving multiple dimensions (one for each problem to be solved by GP). Finally we apply a multi-dimensional Euclidean norm to establish distance between pairs.

3.2 Niching Parameters

All parameters used in our experimental runs are taken directly from those established by Mahfoud [7] or have been changed as little as possible in an attempt to recreate similar incremental testing conditions. All algorithms use phenotypic comparison (as described in section 3.1), crossover rate of 0.9 (in the case of A1) or 1.0 (A2-A6) and no mutation. Each algorithm is run for the equivalent of 20,000 evaluations and results are averaged over 100 runs. To maintain consistency with population sizes, we have scaled population sizes as presented in Mahfoud (5 peak multi-modal functions were allowed 100 individuals [7]) down to 80 in order to fit the number of classes in the problems considered.

3.3 Performance Criteria

For the purposes of comparative analysis, this study is primarily concerned with measures of performance that are more general in nature than the sensitivity and specificity figures (defined in terms of true positive (tp), true negative (tn), false positive (fp) and false negative (fn)) that are traditionally associated with classification:

$$sensitivity = \frac{tp}{tp + fn} \tag{1}$$

$$specificity = \frac{tn}{tn + fp} \tag{2}$$

Although these measures will be established class wise for the test sets of the problems under consideration, here the focus will be on the degree of success of the niching algorithm in question, particularly as it relates to preserving population diversity, maintaining multiple niche optima, and minimizing effects of genetic drift in the GE. These criteria are assessed via the niche population distributions, number of peaks maintained and number of replacement errors made.

Population distributions are simply defined as the counts of population members whose best fitness performance corresponds to the given niche. As defined by Mahfoud, a peak is considered maintained at any iteration if its corresponding subpopulation contains at least one member whose fitness is 80% of the peak's optimal value. Finally, a replacement error refers to the event where a child belonging to niche N_i replaces a parent belonging to niche N_j where $i \neq j$.

3.4 Data Sets and Partitioning

In the following experiments, data sets were partitioned into training and test sets. Partitioning is performed by randomly assigning patterns without replacement to training or test such that 50% of patterns appear in training and 50% appear in test. In and out of class data are stratified in order to achieve proportional representation from each class within the two partitions. Note that the test partition represents a disjoint set ('unseen' exemplars) from the classifier's perspective (the training set). This property of the partitioning affords analysis of the classifier's generalization ability.

Two widely utilized multi-category classification benchmarks from the UCI repository [12] are considered,

1. Iris: Fisher's iris plant database. 150 exemplars each of four continuous inputs and one of three output classes (Iris Setosa, Iris Versicolor, or Iris Verginica);
2. Wine: Wine recognition database; using chemical analysis to determine origin of wine. 178 exemplars each of 13 continuous inputs and one of three output classes.

3.5 Crowding Algorithms as Adapted for GE

3.5.1 Algorithm A1 (equivalent to A0 from [7], with phenotypic similarity): This approach is representative of the original crowding algorithm [5]. Generation gap, *G,* defines the percentage of the population (size POP_SIZE) that is chosen for crossover via fitness proportionate selection. Following the application of crossover (or copy of

parents in the event of failed test for crossover), G × POP_SIZE individuals are chosen randomly for replacement. For each child, a random sample of CF (crowding factor) individuals is chosen without replacement. From this sample, the child replaces the most similar, phenotypically. For algorithm A1 G = 0.1, PC (Probability of application of crossover) is 0.9 and CF = 3:

```
1.  FOR( POP_SIZE )
        a.  Randomly initialize individual
        b.  Evaluate raw fitness (one for each niche) and best fitness
            (best case of raw fitnesses)
2.  FOR( GENS )
        a.  Proportionally select G = 0.1 ( PAIRS ) of the population
            for reproduction
        b.  FOR( PAIR )
                i.  Test / Apply crossover ( PC = 0.9 )
               ii.  Evaluate children (c1, c2) (raw fitnesses, best fit-
                    ness)
              iii.  FOR (child)
                        1.  Randomly choose CF( = 3 ) individuals of
                            population for possible replacement and cal-
                            culate their distance (dci) by phenotype (raw
                            fitnesses) from c
                        2.  Replace the nearest of the CF individuals
                            with c
```

3.5.2 Algorithm A2: Here the algorithm maintains the basic procedure of A1, with the following simplifications: Crossover is applied deterministically (PC = 1.0) and the generation gap (G) is reduced to the minimum to allow for selection of a single pair. The original algorithm is therefore modified by changing step 2 as follows:

```
2.  FOR( GENS )
        a.  Proportionally select 1 PAIR ( Gmin% ) of the population for
            reproduction
        b.  FOR( PAIR )
                i.  Test / Apply crossover ( PC = 1.0 )
```

This simplification was originally chosen in order to rule out parameter settings (G, PC) that may otherwise detract from the thorough analysis of results [7]. The motivation for this also simplification holds here.

3.5.3 Algorithm A3: This algorithm is the same as A2, but here the CF is increased to that of POP_SIZE. Although this is a more costly algorithm in terms of overall time complexity, the hypothesis is that it will correspond to 100% correct replacements. Step 2.b.iii.1 of the algorithm now becomes:

```
1.  Randomly choose CF( = POP_SIZE ) individuals of population for
    possible replacement and calculate their distance (dci) by phe-
    notype (raw fitnesses) from c
```

3.5.4 Algorithm A4: The concept of a crowding factor is removed, instead we proceed by replacing parents at each generation with the children such that the sum of similarity between child-parent replacement pairing is maximized [7]. The hypothesis at this stage is that children will typically be most similar to their parents, thus removing the computationally expensive CP=POP_SIZE. Step 2.b.iii of the algorithm is now replaced and extended with:

```
iii.  Compare children and parent distances (dij) by phenotype (raw
      fitnesses)
 iv.  IF( d11 + d22 < d12 + d21 )
            r1 = p1; r2 = p2
  v.  ELSE
            r1 = p2; r2 = p1
 vi.  replace r1 with c1
vii.  replace r2 with c2
```

3.5.5 Algorithm A5: Also known as Deterministic Crowding [7], all details remain the same as A4 except that generation gap and fitness proportionate selection are dropped. Instead, the population is paired up randomly (without replacement) at each generation. This algorithm also differs from A4 in that a child can only replace the chosen parent if the child's best fitness is an improvement on that of the parent. Step 2 is now:

```
2.  FOR( GENS )
       a.  Pair up individuals without replacement (PAIRS)
       b.  FOR( PAIR )
              i.  Test / Apply crossover ( PC = 1.0 )
             ii.  Evaluate children ( c1, c2 ) (raw fitnesses, best
                  fitness)
            iii.  Compare children  and parent distances (dij) by
                  phenotype (raw fitnesses)
             iv.  IF( d11 + d22 < d12 + d21 )
                        r1 = p1; r2 = p2
              v.  ELSE
                        r1 = p2; r2 = p1
             vi.  IF( best fitness( c1 ) > best fitness ( r1 ))
                        replace r1 with c1
            vii.  IF( best fitness( c2 ) > best fitness ( r2 )) re-
                  place r2 with c2
```

3.5.6 Algorithm A6: Adding the element of sharing to the deterministic crowding algorithm (A5) takes the form of calculating the sharing parameters following each new generation and replacing the chosen parent only if the child has a larger value of dynamic shared fitness (DSF) [8]. Section 2.b.vi and vii become:

```
 vi.  IF( DSF( best fitness( c1 ) ) > DSF( best fitness ( r1 ) ))
      replace r1 with c1
vii.  IF( DSF( best fitness( c2 ) ) > DSF( best fitness ( r2 ) ))
      replace r2 with c2
```

Table 1. Mean niching properties for algorithms A1 – A6 on Wine and Iris Training Data.

				FNS		
		TRE	TPM	C1	C2	C3
IRIS	A1	4626.6	1.29	47.33	17.96	14.71
	A2	5803.26	1.46	45.34	17.74	16.92
	A3	188.44	2.1	42.95	19.04	18.01
	A4	712.46	0.97	75.8	2.63	1.57
	A5	135.11	1.89	76.89	0.03	3.08
	A6	295.16	2.44	36.89	16.14	26.97
WINE	A1	4779.69	1.24	24.58	14.3	29.12
	A2	6134.01	1.52	29.33	19.7	30.97
	A3	125.24	2.6	22.58	20.3	36.32
	A4	1689.03	0.57	19.3	3.89	56.81
	A5	224.04	2.18	5.43	1.97	72.6
	A6	343.29	3	27.12	25.63	27.25

4 Results and Analysis

Results for the GE algorithms (A1-A6) are summarized in Tables 1 and 2 as mean performance over 100 runs. These results report on performance criteria in terms of both classifier quality (sensitivity, specificity, denoted SENS and SPEC respectively, over each problem and class, C1-C3) as well as the relative success of each niching algorithm on a per problem basis as characterized by total replacement errors (TRE), total peaks maintained (TPM), and final niche sizes (FNS, over classes C1-C3).

Table 1 indicates a general decreasing trend to the replacement error over algorithms A1 to A6. However, in terms of the average number of niches maintained, A3 and A6 always retain the correct niche count, where this is also reflected in the final niche sizes for each class. It is interesting to note that of the various niching algorithms investigated, Deterministic Crowding (A5) did not provide particularly strong niche identification.

From Table 2, it is clear that algorithm A3 does not result in a particularly effective classifier in spite of the comparatively good niche characterization from Table 1. In effect A3 would consistently become stuck in a set of suboptimal niches. Algorithm A6, the sharing algorithm, provides the best overall performance, with Deterministic Crowding (A5) also very effective. Classifier results for separate binary runs (SB) have been included for comparative purposes, where the population size was reduced by a factor of $1/k$ that of the niching runs in order to balance computational costs; k (=3) being the number of classes. From Table 2, it appears that the reduced population size was detrimental to the success of the separate binary runs, establishing the value of the niching approach over separate runs under similar resource constraints.

Table 2. Mean classification results for algorithms A1 - A6: Wine and Iris Test Data

		SENS			SPEC		
		C1	C2	C3	C1	C2	C3
IRIS	A1	0.98	0.76	0.81	0.95	0.93	0.93
	A2	0.99	0.87	0.82	0.99	0.91	0.95
	A3	0.97	0.71	0.70	0.93	0.88	0.96
	A4	0.95	0.88	0.85	0.98	0.93	0.97
	A5	1.00	0.85	0.83	0.99	0.92	0.97
	A6	0.99	0.85	0.85	0.99	0.93	0.96
	SB	0.71	0.53	0.76	0.93	0.68	0.67
WINE	A1	0.82	0.73	0.93	0.92	0.94	0.89
	A2	0.76	0.74	0.90	0.93	0.91	0.89
	A3	0.46	0.50	0.73	0.88	0.93	0.66
	A4	0.90	0.82	0.96	0.95	0.97	0.93
	A5	0.79	0.76	0.87	0.93	0.97	0.89
	A6	0.82	0.80	0.97	0.96	0.96	0.93
	SB	0.48	0.46	0.70	0.77	0.75	0.64

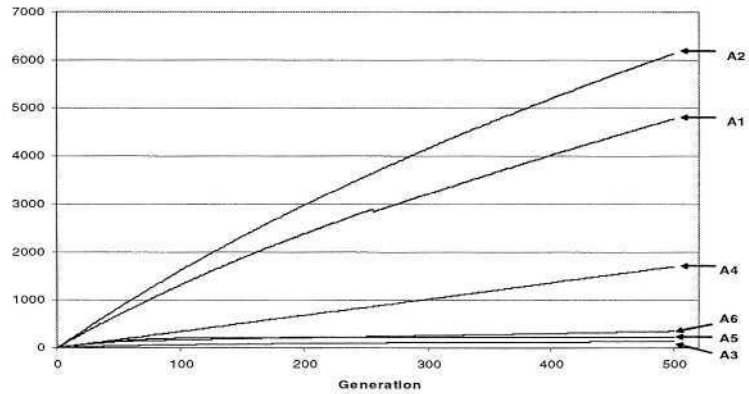


Fig. 1. Mean Cumulative Replacement Errors, A1 to A6 on Wine Training Dataset

4.1 Analysis of Niche Maintenance

This analysis concentrates on the Wine results, although similar properties were demonstrated for the case of Iris. The cumulative replacement error profile illustrates some important differences with respect to the original GA multimodal optimization context [7]. Firstly, 100% crossover resulted in A2 making substantially more replacement errors than the 90% crossover of A1. Secondly, algorithm A4 in which parents were always over-written with the most similar child actually resulted in an

increase in the replacement error, emphasizing the difficulty in comparing programs in GP. In the case of niche maintenance profile, Figure 2, algorithm A4 is again highlighted, thus reinforcing the observation that children of a GP parent often perform much worse. Algorithms A2, A3 and A6 all provide incremental advantages over their respective predecessor. Performance of Deterministic Crowding, A5, relative to Dynamic Niche Sharing, A6, demonstrates a tendency of Deterministic Crowding to drift after initially identifying the correct number of niches. This latter observation is further demonstrated by Figures 3 and 4 that summarize the evolution of each niche under A5 and A6 respectively.

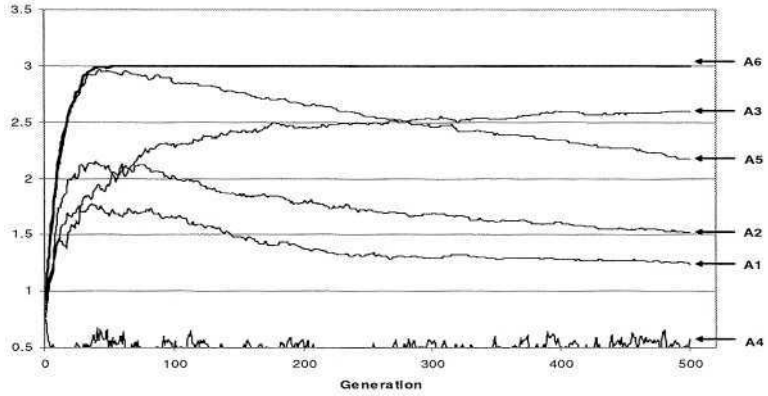


Fig. 2. Mean Niches Maintained, A1 to A6 on Wine Training Dataset

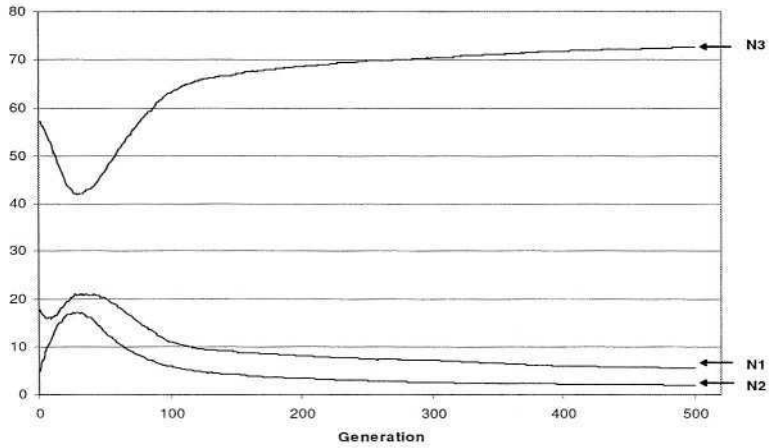


Fig. 3. Mean Niche Count with A5, Deterministic Crowding on Wine Training Dataset

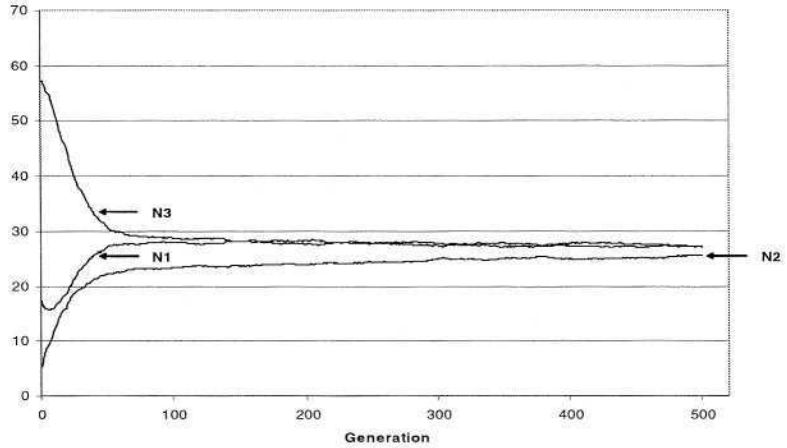


Fig. 4. Mean Niche Count with A6, Dynamic Niche Sharing on Wine Training Dataset

5 Conclusions

The utility of crowding and sharing algorithms as developed for multi-objective optimization with GAs has been demonstrated under GP multi-category classification. Dynamic Niche Sharing appears to provide the best support for the GP context. Deterministic Crowding appears to suffer from a combination of poor child fitness and genetic drift, the combination of which results in the correct number of niches first being located, but thereafter the largest niche continues to consume the majority of population diversity.

Acknowledgements. The authors gratefully acknowledge the support of NSHRF and NSERC Research Grants and a CFI New Opportunities Infrastructure Grant.

References

1. Ghosh A., Freitas A.A.: Data Mining and Knowledge Discovery with Evolutionary Algorithms: Guest Editorial, IEEE Transactions on Evolutionary Computation. 7(6) (2003) 517-518
2. Zhou C., Xiao W., Tirpak T.M., Nelson P.C.: Evolving Accurate and Compact Classification Rules with Gene Expression Programming, IEEE Transactions on Evolutionary Computation. 7(6) (2003) 519-531
3. Au W.-H., Chan K.C.C., Yao X.: A Novel Evolutionary Data Mining Algorithm with Applications to Churn Prediction, IEEE Transactions on Evolutionary Computation. 7(6) (2003) 532-545

4. Kishore, J. K., Patnaik, L. M., Mani, V., Agrawal, V. K.: Application of Genetic Programming for Multicategory Pattern Classification. IEEE Transactions on Evolutionary Computation, Vol. 4, No. 3 (2000) 242-258
5. De Jong, K. A.: An Analysis of the Behaviour of a Class of Genetic Adaptive Systems. Dissertation Abstracts International, Vol 36 No 10 (1975)
6. Deb, K., Goldberg, D. E.: An Investigation of Niche and Species Formation in Genetic Function Optimization. In: Schaffer, J. D. (ed.): Proceedings of Third International Conference of Genetic Algorithms, Morgan Kaufmann, San Mateo, CA (1989) 42-50
7. Mahfoud, S. W.: Crowding and Preselection Revisited. In: Manner, R. and Manderick, B (eds.): Parallel Problem Solving from Nature, 2, Amsterdam, Elsevier Science (1992) 27-36
8. Miller B.L., Shaw M.J.: Genetic Algorithms with dynamic Niche Sharing for Multimodal Function optimization. Uni. Of Illinois at Urbana-Champaign, Dept. General Engineering, IlliGAL Report 95010 (1995) 11 pages
9. Yao X., Liu Y., Lin G.: Evoutionary Programming Made Faster. IEEE Transactions on Evolutionary Computation. 3(2), (1999) 82-102
10. O'Neill, M., Ryan, C.: Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language. Kluwer Academic Publishers. (2003)
11. Bojarczuk, C. C., Lopes, H. S., Freitas, A. A.: An Innovative Application of a Constrained Syntax Genetic Programming System to the Problem of Predicting Survival of Patients. In C. Ryan et al. (eds.) EuroGP 2003, Lecture Notes in Computer Science, Vol. 2610. Springer-Verlag, Berlin Heidelberg (2003) 11-21
12. Blake, C. L., Merz, C. J.: UCI Repository of Machine Learning Databases. University of California, Irvine, Dept. of Information Computer Sciences, http://www.ics.uci.edu/~mlearn (1998)

On the Strength of Size Limits in Linear Genetic Programming

Nicholas Freitag McPhee, Alex Jarvis, and Ellery Fussell Crane

University of Minnesota, Morris, Morris MN 56267, USA
{mcphee,jarv0063,cran0117}@mrs.umn.edu,
http://www.mrs.umn.edu/~mcphee

Abstract. Bloat is a common and well studied problem in genetic programming. Size and depth limits are often used to combat bloat, but to date there has been little detailed exploration of the effects and biases of such limits. In this paper we present empirical and theoretical analyses of the effect of size limits on variable length linear structures. Specifically, we examine the relationship between size limits and the average size of individuals in a population and define the notion of size limit strength. When a size limit is strong, the average size of a population converges to a relatively stable value. When a size limit is weak, no such convergence occurs. The average size instead appears to perform a random walk within a bounded region. We use schema theory to show this is likely a result of uneven sampling of the search space.

1 Introduction

The causes and effects of code growth in genetic programming (GP) have been extensively researched [19,5,4,6]. In order to avoid the negative repercussions of bloat, a variety of corrective measures are commonly employed to keep program sizes in check [13,18,7,3]. One frequently used method is to employ a fixed limit on program size by restricting either the depth or the size of syntax trees.

While these limits have the desired effect of keeping the sizes down, little is known about what other impacts such limits might have on the dynamics of GP. Previous research has shown that decisions such as these can have significant effects on the behavior of runs [2] and on important structural features such as the size and shape distributions of populations [8,10,9]. It would therefore be useful to better understand what structural effects size limits might have, especially given their widespread use.

To shed some light on this issue we collected a large body of data on the impact of size limits on a test problem with variable length linear structures; this is described in Section 2. We present and analyze the empirical results in Section 3. The data shows a definite difference in behavior among size limits, leading us to define the notions of *strong* and *weak* size limits. Strong size limits cause the average size of a population to converge to a relatively stable value. Weak size limits, however, achieve no such convergence. Instead weak limits cause the population's average size to perform what we believe to be a bounded

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 593–604, 2004.

random walk over a large area. In Section 4 we perform a theoretical analysis of our results using the theory of holes [8] based on exact schema theory for GP [16]. This analysis shows that there is a balance between the set of individuals exceeding the size limit and the set of short, unfit individuals. Section 4 also addresses how these results on variable length linear structures might generalize to N-ary trees. We discuss future work in Section 5 and provide conclusions in Section 6.

2 Terms and Concepts

2.1 Average Population Size

In this work the number of individuals in a population remains constant across the entirety of the runs. To simplify the discussion we will use phrases like "the average size of a population" to describe the average size over all the individuals in the population.

2.2 Convergent Average Size

When there is bloat pressure, the average size of individuals in any population increases rapidly during the early generations of a run. In runs using a strong size limit the average size will eventually reach a stable value from which it will seldom deviate significantly (see Fig. 2). In such cases, we define the population's *convergent average size* to be the mean of the average size over time. A key goal of Section 3 is to better understand how size limits affect convergent average size.

2.3 Size Limit Strength

One of the important observations of Section 3 is that run dynamics are governed in a significant way by the choice of size limit. We say that a size limit is *strong* for a run if that run has a convergent average size. Otherwise we say the size limit is *weak.*

2.4 Crossover on Variable Length Linear Structures

The work reported here is all on GP with variable length linear structures (similar to those used in systems like [11]). We used linear structures because the theoretical analysis is more manageable and the computations are more tractable. This has yielded a number of important results for the linear case, and preliminary results suggest that many of the key ideas here are also applicable (at least in broad terms) to the non-linear tree structures typically used in GP (see Section 4.2).

Because our primary interest is the effect of size limits on code growth, we will focus exclusively on the standard subtree-swapping GP crossover operator

which is known to induce bloat. This operator acts by removing a non-empty suffix of an individual and replacing it with a new suffix taken from another individual.

More formally, in linear structure GP where $\mathcal{F}$ is the set of non-terminal nodes and $\mathcal{T}$ is the set of terminal nodes, individuals can be seen as sequences of symbols $c_0 c_1 \ldots c_{N-1}$ where $c_i \in \mathcal{F}$ for $i < N-1$ and $c_{N-1} \in \mathcal{T}$.

Crossover, then, acts on two parent individuals by removing a non-empty suffix from one parent and replacing it with a non-empty suffix from the other parent. The removed suffix is $c_j c_{j+1} \ldots c_{N-1}$ where j is chosen uniformly such that $0 \leq j < N$. The inserted suffix is $d_{j'} d_{j'+1} \ldots d_{N'-1}$ where j' (which could differ from j) is chosen uniformly such that $0 \leq j' < N'$.

2.5 The One-Then-Zeros Problem

We chose to study the impact of size limits using the well-studied *one-then-zeros* problem. In this problem we have $\mathcal{F} = \{0, 1\}$ and $\mathcal{T} = \{0\}$, where both 0 and 1 are unary "operators". This gives us a problem that is essentially equivalent to studying variable length strings of 0's and 1's, with the constraint that the strings always end in a 0. Fitness in this problem will be 1 if the string starts with a 1 and has zeros elsewhere, i.e., the string has the form $1(0)^a$ where $a > 0$; fitness will be 0 otherwise.

One of the reasons for studying this problem is that under selection and crossover this problem induces bloat [8], which ensures that size limits will have an impact. Another advantage is that this problem is amenable to schema theory analysis [15,8,10,17,9].

3 Empirical Results

3.1 Experimental Setup

All the runs presented in this paper use the same parameters except where noted.

Number of generations. All runs in Fig. 1 were for 1,000 generations, and all runs in Fig. 2 were for 2,000 generations.

Control strategy. We use a non-elitist generational control strategy.

Selection mechanism. Since all individuals have either fitness 0 or 1, we used uniform random selection from the set of individuals with fitness 1.

Initialization. The populations were initialized entirely with fit individuals having size 10, i.e., strings of the form $1(0)^9$.

Size limits. These were implemented so that an otherwise fit individual received a fitness of 0 if it's size was *strictly greater* than the size limit. A variety of size limits were used; see below for details.

Operators. We exclusively use crossover in these experiments, so every individual is constructed by choosing two fit parents and performing linear structure crossover as described above. There is no mutation or copying of individuals from one generation to the next.

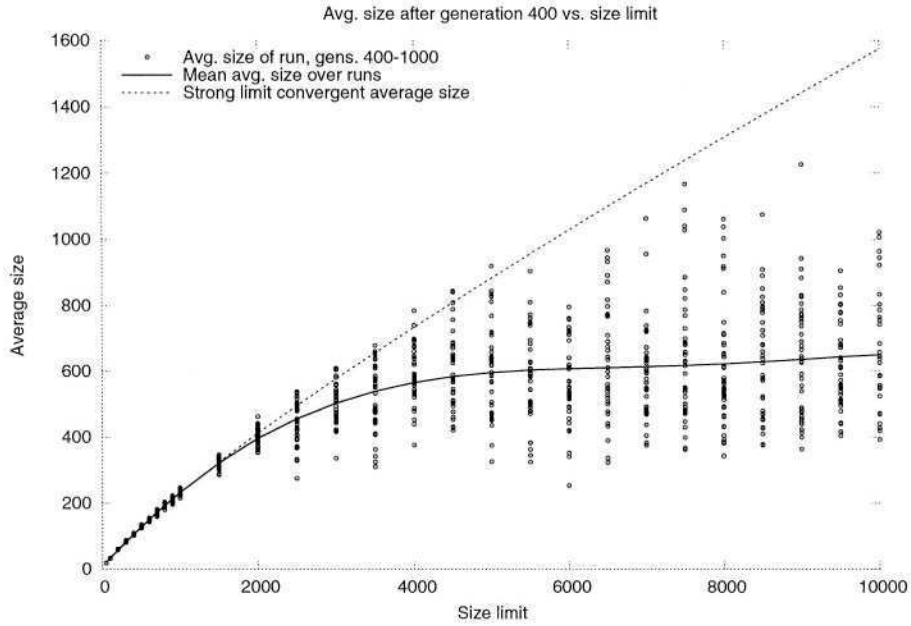


Fig. 1. The average sizes of runs from generation 400 to generation 1,000 versus the size limits used in the runs. The solid line connects the mean values of the runs at each size limit. The dotted line shows the results of a regression over the data for size limits 50 to 1,000. All data was gathered from populations of size 1,000.

In each run the average size of the population was calculated by taking the mean value of the population averages in the range of generations 400 to 1000. This region was selected because in the case of strong size limits the population had converged by generation 400.

3.2 Convergent Average Sizes and Size Limits

The graph in Fig. 1 displays the distribution of run average sizes for a variety of size limits. The average sizes remain very clustered in the lower size limits and gradually become more variable as the size limits grow. It also appears that after a certain point this variance does not increase, instead remaining fairly stable. It was this qualitative change in behavior that led us to propose the notion of size limit strength (see Section 2.3).

Fig. 1 also shows a clear change in the magnitude of the average sizes as we move from strong to weak size limits. The solid line in the figure connects the mean value of the average sizes for all runs at each size. After climbing steadily over the strong size limits, the mean of the average sizes clearly flattens for the

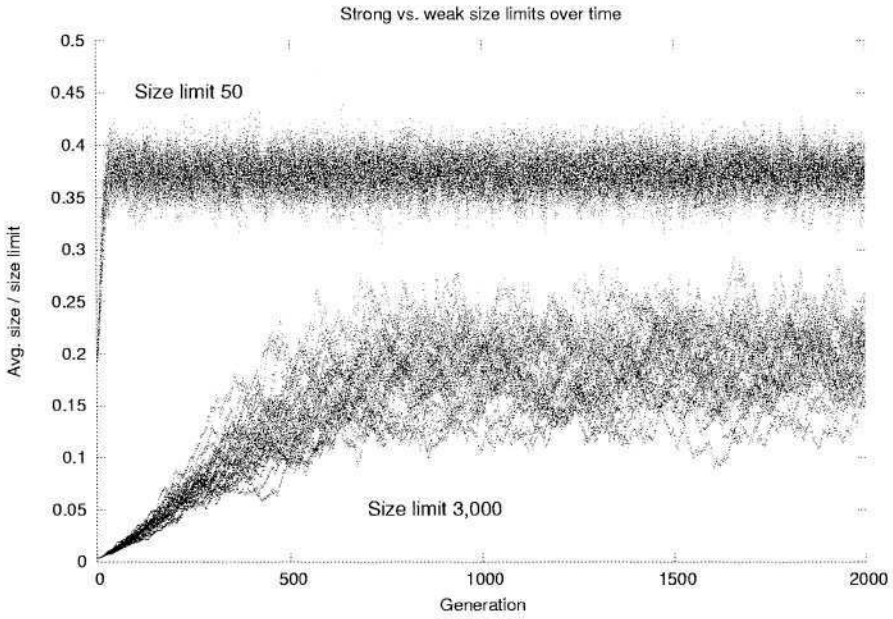


Fig. 2. Average size divided by size limit over time of multiple runs with size limits 50 and 3,000. The average sizes were divided by their respective size limits to provide a uniform scale. In all cases the population size was 1,000. The limit of 50 is clearly strong, as the average sizes of all of the runs converge onto a stable value with limited variance. The limit of 3,000 is much weaker, and appears to perform a random walk within a bounded area.

weaker limits. To better illustrate this we performed a regression on the average sizes for limits in the range 50 to 1,000 and graphed that as the dashed curve in Fig. 1. This clearly shows divergence that starts around size limit 1,500 and increases over time.

It is also interesting to note that the average sizes were consistently less than one third of the size limit in cases where the size limit was strong. The average sizes were proportionally even smaller for weak limits.

3.3 Impact of Size Limit Strength

Another method of examining the differences in size limit strength is presented in Fig. 2. This figure displays the average sizes of multiple runs where each dot is the average size across the population for a single run at a particular generation. The upper set of data comes from runs using a strong size limit of 50, whereas the lower set used the weak size limit of 3,000. In the graph the average sizes of each run were divided by the size limit to provide a uniform scale for the two data sets. The first and third quartiles for the unsealed limit 50 data are 18.11 and 19.17 respectively, so the average size typically stays within 0.5 of the mean

of 18.64. The first and third quartiles for the unsealed limit 3,000 data are 496.1 and 639.4, so the average size varies quite substantially from the mean of 571.4.

The average sizes for runs in which the size limit is 50 clearly increase rapidly to a convergent size and stay within a narrow range from then on. The upper bound is presumably generated by the size limit culling overly large individuals, and the lower bound is presumably a result of bloat pressure.

The average size of the runs using size limit 3,000, however, does not converge. The average sizes increase fairly steadily in the early generations, but seem to perform a random walk in later generations. This early behavior is an example of traditional bloat, but it's less clear what's happening later on. It appears, however, that there is a both an upper and lower bound to the random walk. The apparent lower bound is likely tied to the fact that if the average size is too small, it becomes increasingly probable that crossover will generate individuals that don't match the one-then-zeros pattern. The upper bound is presumably a function of the size limit where if the average size is too large, crossover is likely to generate individuals whose size exceeds the limit. This is discussed further in the theoretical analysis in the next section.

4 Theoretical Analysis and Discussion

4.1 Explaining the Data: The Theory of Holes

The data presented in Section 3 clearly indicates that there is a continuum of strengths for size limits in the one-then-zeros problem. The question, then, is why strong size limits tightly constrain the average size to such a narrow range, and weak limits allow the average size to drift over a substantial range of values.

In this section we'll present a theoretical analysis of these empirical results using findings from exact schema theory for genetic programming ([12,16]). This analysis suggests that the strength of a size limit is a function of how well the crossover operator samples the set of unfit individuals. In particular, we find that there is an overall balance between the sampling of the set of individuals that fail to match the one-then-zeros target pattern and the sampling of the set of individuals that exceed the size limit.

An important result from [15] uses schema theory to derive a predictive relationship between the average size of a population, the average size of its unfit individuals, and the change in the average size over time. (This was subsequently used as the basis of the Tarpeian bloat control mechanism in [13].) In this section we will present a simplified summary of those results, which can be used to better understand the empirical data presented earlier.

First assume, as is the case in the one-then-zeros problem, that the problem's fitness function has only two values: 0 and 1. We will refer to the region of the search space where the fitness is 0 as the "hole". In the standard one-then-zeros problem the hole is the set of individuals that don't match the one-then-zeros pattern. If we add size limits to the one-then-zeros problem, then the hole also contains all strings that exceed the size limit.

The key result from the theory of holes in [15] is that the average size of the population will move away from the average size of the hole. To formalize this let us first present some definitions:

- $\Lambda(i)$ to be the size of an individual i
- $|S|$ to be the size of a set (of individuals)
- $\mu(t)$ to be the average size of the population at time t
- $E[\mu(t+1)]$ to be the expected average size of the population at time $t+1$ given what is known about the population at time t
- $U(t)$ to be the set of unfit individuals at time t
- $\mu(U(t))$ to be the average size of the set $U(t)$, i.e., $\mu(U(t)) = \frac{\sum_{u \in U(t)} \Lambda(u)}{|U(t)|}$

Given these definitions we can express the result from [15] as:

$$\mu(U(t)) > \mu(t) \Longleftrightarrow E[\mu(t+1)] < \mu(t)$$
$$\mu(U(t)) < \mu(t) \Longleftrightarrow E[\mu(t+1)] > \mu(t)$$

This tells us that if the average size of the unfit individuals is greater than the average size of the population, we would expect the population to shrink (move away from the hole). Similarly, if the average size of the unfit individuals is less than the population's average size, we would expect the population to grow (again moving away from the hole).

Implicit in this result, but not discussed in [15], is the fact that if the average population size is relatively stable, i.e., $E[\mu(t+1)] \approx \mu(t)$, then the average size of the hole must be approximately the same as the average size of the population, i.e., $\mu(U(t)) \approx \mu(t)$. Thus if we observe experimentally that the average size is roughly constant over a number of generations, we can infer that the average size of the unfit individuals must also be roughly the same as the average size of the population over that period of time. This result, in fact, still holds (at least in the aggregate) even if the average size varies considerably over time as long as the mean over time remains fairly constant. Thus while $\mu(U(t))$ might differ significantly from $\mu(t)$ for a particular generation t, if we average over a number of consecutive generations we would expect the mean of $\mu(U(t))$ to be close to the mean of $\mu(t)$.

The previous result holds for any GP system (linear or not) where the fitness function has just two values. We can further extend this result in the case of the one-then-zeros problem with size limits by noting that the set U of unfit individuals can be split into two disjoint sets $U = Z \cup L$, where

- $Z(t)$ is the set of individuals (at time t) that have fitness 0 because they don't match the one-then-zeros pattern, but that are legal in the sense that they are not too large.
- $L(t)$ is the set of individuals (at time t) that are larger than the size limit, regardless of whether they match the one-then-zeros pattern.

In this case we can rewrite as follows:

$$\mu(U(t)) = \mu(t)$$

$$\equiv \quad \langle \text{Avg. length is the sum of lengths over number of strings} \rangle$$
$$\frac{\sum_{u \in U(t)} \Lambda(u)}{|U(t)|} = \frac{|U(t)|\mu(t)}{|U(t)|}$$

$$\equiv \quad \langle \text{Splitting } U \text{ into } Z \text{ and } L \rangle$$
$$\frac{(\sum_{z \in Z(t)} \Lambda(z)) + (\sum_{l \in L(t)} \Lambda(l))}{|Z(t)| + |L(t)|} = \frac{|Z(t)|\mu(t) + |L(t)|\mu(t)}{|Z(t)| + |L(t)|}$$

$$\equiv \quad \langle \text{Moving } \mu(t) \text{ inside the summations}$$
$$\frac{(\sum_{z \in Z(t)} (\Lambda(z) - \mu(t))) + (\sum_{l \in L(t)} (\Lambda(l) - \mu(t)))}{|Z(t)| + |L(t)|} = 0$$

Assuming that $|Z(t)| + |L(t)| \neq 0$ (i.e., there is at least one unfit individual) this is equivalent to

$$\sum_{z \in Z(t)} (\mu(t) - \Lambda(z)) = \sum_{l \in L(t)} (\Lambda(l) - \mu(t)) \tag{1}$$

This means that if the average size is (roughly) constant we would expect the sum of the distances between $\mu(t)$ and the length of the elements of $Z(t)$ to be (roughly) the same as the sum of the distances between $\mu(t)$ and the length of the elements of $L(t)$. Since we know that all the elements of $L(t)$ are larger than the size limit, they are also presumably larger than $\mu(t)$ in almost all typical circumstances. Thus the right hand side of Eq. (1) will be positive except for pathological circumstances (e.g., having all the individuals in the initial population being larger than the size limit). Consequently the left hand side must also be positive, indicating that the bulk of the legal individuals that don't match the one-then-zeros pattern are smaller than $\mu(t)$.

Given this result, several important observations about Eq. (1) can be made. First, Eq. (1) says only that the sums are (roughly) equal, and nothing about the relative number of individuals in $|Z(t)|$ or $|L(t)|$, or the relative magnitudes of $\mu(t) - \Lambda(z)$ or $\Lambda(l) - \mu(t)$. While these can be close, it is in fact more likely that they will be quite different. The natural distribution of lengths for linear structures when using crossover alone is a discrete gamma distribution (see [15, 9]) where the majority of the individuals have below average size and are then balanced out by a smaller numbers of fairly long individuals. The same sort of distribution appears to hold even with size limits. A strong limit leads to an average size that is roughly one third of the size limit, and a weak limit leads to an average size that is even smaller. Thus what one sees in practice is a small number of individuals in $L(t)$ whose sizes are significantly greater than $\mu(t)$. They are then balanced by a larger number of individuals in $Z(t)$ whose average distance from $\mu(t)$ is considerably smaller.

The second observation is that this result is actually quite general, only depending on the ability to split U up into two disjoint groups, which will be

possible whenever size limits are employed. There are no assumptions here about linear structures so this would apply just as well to N-ary trees. There are also no reliance on the details of the one-then-zeros problem (except for the fact that it only has two fitness values), and would apply to any problem with just two fitness levels. While this may seem like a serious restriction, in practice it is not uncommon for a population to have a very limited range of fitness values, especially in the later stages of a run. This can (depending on the details of the selection mechanism) lead to an effective division of the population into two groups. One group has such low fitness that the chances of a parent coming from this group is nearly 0, and fitnesses in the other group are sufficiently similar that their probability of being selected are effectively the same. In such a situation we would expect Eq. (1) to hold – at least approximately.

The final observation is that these results and those in [15] make it clear that changes in the average size of a population are driven largely by how well (or poorly) the crossover operator samples $Z(t)$ and $L(t)$. Thus we see marked code growth in the early stages of a run when crossover doesn't sample $L(t)$ at all (i.e., no individuals are constructed whose size exceeds the limit) and the average size of individuals not matching the pattern is less than $\mu(t)$. As $\mu(t)$ grows, however, the probability of sampling $L(t)$ increases and the probability of sampling $Z(t)$ decreases (because there are fewer destructive crossovers).

Let, then, $P(S_{Z(t)})$ be the probability of (the crossover operator) sampling $Z(t)$, and $P(S_{L(t)})$ be the probability of sampling $L(t)$. Then the strength of the size limit is then determined by whether $P(S_{Z(t)})$ drops to nearly 0 before $P(S_{L(t)})$ rises appreciably. If $P(S_{L(t)})$ rises quickly enough that both $Z(t)$ and $L(t)$ are both being consistently sampled, then the balance point defined by Eq. (1) is reached and $\mu(t)$ is constrained fairly tightly over time (see the size limit 50 data in Fig. 2). If, on the other hand, $P(S_{L(t)})$ rises slowly, the population reaches a state where there is little chance of consistently sampling either $Z(t)$ or $L(t)$. In this case $\mu(t)$ essentially performs a bounded random walk over time. $\mu(t)$ is then free to drift up and down, with the constraints that if it drifts too high there will start to be consistent sampling of $L(t)$, which will push $\mu(t)$ down, and if drifts too low it will start to sample $Z(t)$, pushing $\mu(t)$ back up. This is illustrated by the size limit 3,000 data in Fig. 2.

4.2 Generalization to N-ary Trees

An important question is how these results will change when applied to more traditional (non-linear) GP trees. While depth limits and size limits are equivalent for variable length linear structures (unary trees), they can behave quite differently than bushier N-ary trees. A key difference is that size is essentially conserved in the sense that if nodes are added to one part of a tree an equivalent number of nodes need to be removed elsewhere to maintain the same size. On the other hand, nodes can be added to a tree in a way that changes the depth of parts of the tree without changing the depth of the overall tree. Thus when using depth limits there can still be considerable growth (measured as increasing

tree size) even among trees that are at or near the depth limit. Alternatively, trees near a size limit cannot grow but can only redistribute their nodes.

It then seems likely that these results will more readily generalize to size limits for N-ary trees. For the unary trees both size and depth are conserved, and there is in fact considerably less flexibility than in the N-ary case. Since there is only one possible shape for a given size, there is no longer the option to redistribute nodes while preserving size.

In the case of N-ary trees near a size limit, the tendency of the crossover operator to generate offspring that are too large (i.e., sample the space of trees that are larger than the size limit) is likely to be similar to the sampling behavior presented here. We conjecture, however, that the sampling is weaker in the case of depth limits, especially for fairly balanced trees, as there are likely to be many crossovers that don't increase the depth.

While these sampling differences between unary and N-ary structures would seem to be a major concern, it's unclear how important the difference is in practice. First, there is considerable evidence that N-ary GP often generates very deep narrow trees (see [1] for excellent visualizations of this phenomena). In such cases the distribution of sizes and lengths of random subtrees will be much more uniform, and therefore closer to the results presented in this paper.

5 Future Work

Because we were primarily focused on code growth, we used only the subtree crossover operator in this work as it was known to induce bloat. Prior results on different mutation operators [10,14] and combinations of operators [9], however, suggests that this work could be extended to include mutation operators. Since mutation operators tend to act as size limiting factors (see [9]) it would be interesting to see what effects they would have on convergent average sizes and the transition from strong to weak size limits.

We conjectured in Section 4.2 that many of these results will generalize in broad terms from linear (unary) structures to more traditional N-ary trees. While some of the theoretical arguments from Section 4.1 suggest that we should see some of the same dynamics, the details are likely to be quite different, and this obviously deserves further exploration.

We have collected preliminary data that suggests that population size plays a significant role in determining the value of the convergent average size for a given size limit. The specifics of this relationship are not yet known, however, and warrant further investigation.

6 Conclusion

The empirical results from Section 3 clearly show a qualitative and quantitative difference between strong and weak size limits. Strong size limits cause the average size of a population to converge to a relatively stable value. When a size

limit is weak, no such convergence occurs. The average size instead appears to perform a random walk within a bounded region.

In Section 4.1 we use the schema theory of holes to better understand the impact of size limit strength. The key result of that analysis is that if the average size is approximately stable then there is a balance between the set of individuals not matching the one-then-zeros pattern and the set of individuals exceeding the size limit.

This work again highlights the challenges of inferring broad structural trends based on empirical data, especially in the face of unavoidably limited resources. The inferences one would make from Fig. 1, for example, would be quite different if one only had data for size limits below 1,000. Similarly, the behavior shown in Fig. 2 changes quite dramatically over time. It seems plausible, for example, that the distribution of mean sizes for limit 3,000 has (in the aggregate) leveled off, but the data given is not sufficient to warrant any strong conclusion. Theory, then, becomes especially important as at least some theoretical results (e.g., Eq. (1) from Section 4.1) are sufficiently general that they can be used to predict behavior even in distant, unseen regions of the problem domain.

Acknowledgement. We would like to thank Engin Sungur for his generous assistance and comments.

References

1. J. M. Daida, A. M. Hilss, D. J. Ward, and S. L. Long. Visualizing tree structures in genetic programming. In E. Cantú-Paz and *et al,* editors, *Genetic and Evolutionary Computation – GECCO-2003,* volume 2724 of *LNCS,* pages 1652–1664, Chicago, 12-16 July 2003. Springer-Verlag.
2. C. Gathercole and P. Ross. An adverse interaction between crossover and restricted tree depth in genetic programming. In J. R. Koza and *et al,* editors, *Genetic Programming 1996: Proceedings of the First Annual Conference,* pages 291–296, Stanford University, CA, USA, 28–31 July 1996. MIT Press.
3. J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, 1992.
4. W. B. Langdon and R. Poli. *Foundations of Genetic Programming.* Springer, 2001.
5. W. B. Langdon, T. Soule, R. Poli, and J. A. Foster. The evolution of size and shape. In L. Spector, W. B. Langdon, U.-M. O'Reilly, and P. J. Angeline, editors, *Advances in Genetic Programming 3,* chapter 8, pages 163–190. MIT Press, Cambridge, MA, USA, June 1999.
6. S. Luke. Modification point depth and genome growth in genetic programming. *Evolutionary Computation,* 11(1):67–106, Spring 2003.
7. S. Luke and L. Panait. Lexicographic parsimony pressure. In W. B. Langdon and *et al,* editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference,* pages 829–836, New York, 9-13 July 2002. Morgan Kaufmann Publishers.
8. N. F. McPhee and R. Poli. A schema theory analysis of the evolution of size in genetic programming with linear representations. In J. F. Miller and *et al,* editors, *Genetic Programming, Proceedings of EuroGP'2001,* volume 2038 of *LNCS,* pages 108–125, Lake Como, Italy, 18-20 April 2001. Springer-Verlag.

9. N. F. McPhee and R. Poli. Using schema theory to explore interactions of multiple operators. In W. B. Langdon and *et al,* editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference,* pages 853–860, New York, 9-13 July 2002. Morgan Kaufmann Publishers.
10. N. F. McPhee, R. Poli, and J. E. Rowe. A schema theory analysis of mutation size biases in genetic programming with linear representations. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001,* pages 1078–1085, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, 27-30 May 2001. IEEE Press.
11. M. O'Neill and C. Ryan. Grammatical evolution. *IEEE Transaction on Evolutionary Computation,* 5(4), 2001.
12. R. Poli. Exact schema theory for genetic programming and variable-length genetic algorithms with one-point crossover. *Genetic Programming and Evolvable Machines,* 2(2):123–163, June 2001.
13. R. Poli. A simple but theoretically-motivated method to control bloatin genetic programming. In C. Ryan and *et al,* editors, *Proceedings of the Sixth European Conference on Genetic Programming (EuroGP-2003),* volume 2610 of *LNCS,* pages 204–217, Essex, UK, 2003. Springer Verlag.
14. R. Poli and N. F. McPhee. Exact gp schema theory for headless chicken crossover and subtree mutation. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001,* pages 1062–1069, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, 27-30 May 2001. IEEE Press.
15. R. Poli and N. F. McPhee. Exact schema theorems for GP with one-point and standard crossover operating on linear structures and their application to the study of the evolution of size. In *Genetic Programming, Proceedings of EuroGP 2001,* LNCS, Milan, 18-20 Apr. 2001. Springer-Verlag.
16. R. Poli and N. F. McPhee. General schema theory for genetic programming with subtree-swapping crossover: Part 1. *Evolutionary Computation,* 11(1):53–66, Spring 2003.
17. J. E. Rowe and N. F. McPhee. The effects of crossover and mutation operators on variable length linear structures. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001),* San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann.
18. S. Silva and J. Almeida. Dynamic maximum tree depth. In E. Cantú-Paz and *et al,* editors, *Genetic and Evolutionary Computation – GECCO-2003,* volume 2724 of *LNCS,* pages 1776–1787, Chicago, 12-16 July 2003. Springer-Verlag.
19. T. Soule and J. A. Foster. Effects of code growth and parsimony pressure on populations in genetic programming. *Evolutionary Computation,* 6(4):293–309, Winter 1998.

Softening the Structural Difficulty in Genetic Programming with TAG-Based Representation and Insertion/Deletion Operators

Nguyen Xuan Hoai and R.I. McKay

School of IT & EE, Australian Defence Force academy, University of New South Wales, ACT 2600, Australia.
x.nguyen@adfa.edu.au; rim@cs.adfa.edu.au

Abstract. In a series of papers [3-8], Daida et. al. highlighted the difficulties posed to Genetic Programming (GP) by the complexity of the structural search space, and attributed the problem to the expression tree representation in GP. In this paper, we show how to transform a fixed-arity expression tree in GP to a non fixed-arity tree (Catalan tree) using representation based on Tree Adjoining Grammars (TAGs). This non fixed-arity property, which is called feasibility, allows us to design many types of genetic operators (as in [16]). In particular, insertion/deletion operators arising naturally from the representation play a role as structural mutation operators. By using these dual operators on TAG-based representation, we demonstrate how these operators can help to soften the structural search difficulties in GP.

1 Introduction

Since its original proposal [1, 13], standard genetic programming (GP) has been using expression tree as the representation for programs. In an expression tree, each node has a fixed arity (fixed number of children). We argue that this fixed-arity property makes it hard to design operators that can act on structure of the expression tree. Moreover, in a series of papers [3-8], Daida et. al. have shown that structure search alone can pose a great difficulty to standard GP (using expression tree representation and sub-tree swapping as crossover operator). In particular, they pointed out that vast majority of expression tree structures are essentially not searchable with GP, attributing the problem to the expression tree representation itself.

In this paper, we show how to transform a fixed-arity expression tree in GP to a non fixed-arity tree (Catalan tree) using a representation based on Tree Adjoining Grammars (TAGs). This non fixed-arity property, which we call feasibility, allows us to design many types of genetic operators (as in [16]). In particular, insertion/deletion operators arising naturally from the representation play a role as structural mutation operators. By using these dual operators on the TAG-based representation, we demonstrate how these operators can help to soften the structural difficulty in GP.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 605–616, 2004.

The paper, therefore, proceeds as follows. In section 2, we introduce the idea of TAG-based representation, whereby tree adjoining grammar derivation trees are used as genotypes and the expression trees are phenotypes. We also show how this genotype-to-phenotype map transforms fixed-arity trees to non-fixed-arity trees and obtain deletion and insertion as structural mutation operators in a natural way. Section 3 contains a brief summary of structural difficulty in standard GP, based on work [3-8] by Daida et. al. In section 4, experimental results using insertion/deletion in a hill-climbing search on Daida's LID problem [8] are given; they are discussed, and compared with results of standard GP in [8]. We give a brief survey of related work in section 5 and conclude the paper with section 6, containing some ideas for extending the work.

2 TAG-Based Representation for GP

In this section, we first give the definitions of tree adjoining grammars (TAGs) and their derivation trees. Next, we describe how TAG-derivation trees can be used for genetic programming as in [14-16]. Finally, we describe insertion and deletion operators used with TAG-based representation.

2.1 Tree Adjoining Grammars

Joshi and his colleagues in [11] proposed tree-adjunct grammars, the original form of tree adjoining grammars (TAG). Adjunction was the only tree-rewriting operation. Later, the substitution operation was added and the new formalism became known as TAG. Although the addition of substitution did not change the strong and weak generative power of tree adjunct grammars (their tree and string sets), it compacted the formalism with fewer elementary trees [12].

TAGs are tree-rewriting systems, defined in [12] as a 5-tuple (T, V, I, A, S), where T is a finite set of terminal symbols; V is a finite set of non-terminal symbols ($T \cap V = \varnothing$); $S \in V$ is a distinguished symbol called the start symbol; and $E = I \cup A$ is a set of elementary trees (initial and auxiliary respectively). In an elementary tree, interior nodes are labeled by non-terminal symbols, while nodes on the frontier are labeled either by terminal or non-terminal symbols. The frontier of an auxiliary tree must contain a distinguished node, the foot node, labeled by the same non-terminal as the root. The convention in [12] of marking the foot node with an asterisk (*) is followed here. With the exception of the foot node, all non-terminal symbols on the frontier of an elementary tree are terminal or marked as $\downarrow$ for substitution. Initial and auxiliary trees are denoted α and β respectively. A tree whose root is labeled by X is called an X-type tree. Figure 1 shows some examples of initial and auxiliary trees.

The key operations used with tree-adjoining grammars are the adjunction and substitution of trees. Adjunction builds a new (derived) tree γ from an auxiliary tree β and a tree α (initial, auxiliary or derived). If tree α has an interior node labeled A, and β is an A-type tree, the adjunction of β into α to produce γ is as follows: Firstly, the sub-tree α_1 rooted at A is temporarily disconnected from α. Next, β is attached to α to

replace the sub-tree. Finally, α_1 is attached back to the foot node of β. γ is the final derived tree achieved from this process. Adjunction is illustrated in Figure 2.

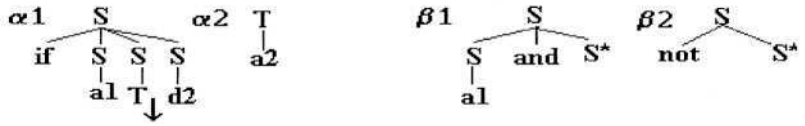


Fig. 1. Some examples of initial and auxiliary trees.

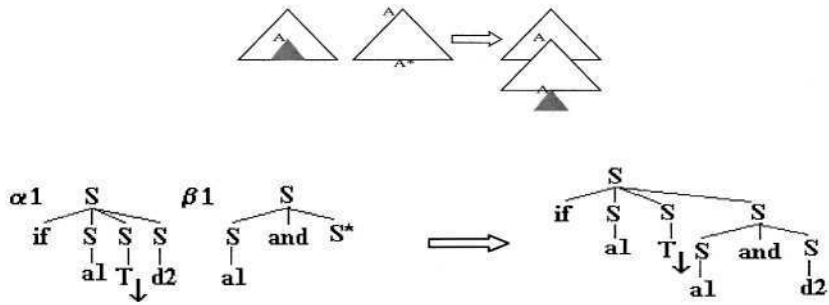


Fig. 2. Adjunction.

In substitution, a non-terminal node on the frontier of an elementary tree is substituted with another initial tree with a root labelled with the same non-terminal symbol. Substitution is illustrated in Figure 3.

The tree set of a TAG can be defined as follows [12]:

$\mathbf{T}_G$ = {all tree t: t is completed and t is derived from some initial S-trees through adjunctions and substitutions}.

Where a tree t is completed, if t is an initial tree and all of the leaf nodes of t are labelled by terminal symbols. The language generated by the TAG G is defined as

$L_G = \{w \in T^*$: w is the yield of some tree $t \in T_G\}$.

In TAG, there is a distinction between derivation and derived trees. A derivation tree in TAG [12, 19, 23, 26] is a tree-structure, which encodes the history of derivation (substitutions and adjunctions) to produce the derived tree. Each node is labelled by an elementary tree name: the root must be labelled by an α tree name, and the other nodes with either an α or β tree. The links between a node and its offspring are marked by addresses for adjunctions and substitutions. Figure 4 illustrates the derivation and derived trees in TAGs (the discontinuous lines mean substitutions).

The set of languages generated by TAGs (called TAL) is a superset of the context-free languages generated by CFGs; and is properly included in indexed languages [12]. More properties of TAL can be found in [12].

One special class of TAGs is lexicalized TAGs (LTAGs) [12, 20, 21], in which each elementary tree of an LTAG must have at least one terminal node. It has been proven that there is an algorithm, which for any context-free grammar G, generates a corresponding LTAG $\mathbf{G}_{lex}$ that generates the same language and tree set as G ($\mathbf{G}_{lex}$ is then said to strongly lexicalize G) [12, 20, 21]. The derivation trees in G are the derived trees of $\mathbf{G}_{lex}$.

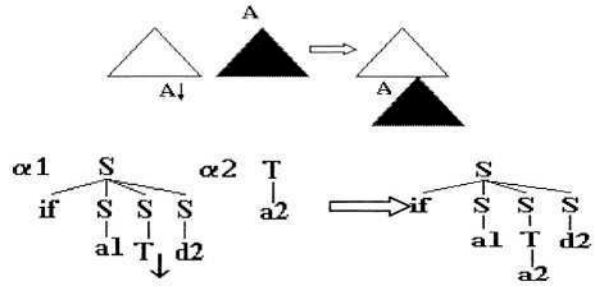


Fig. 3. Substitution.

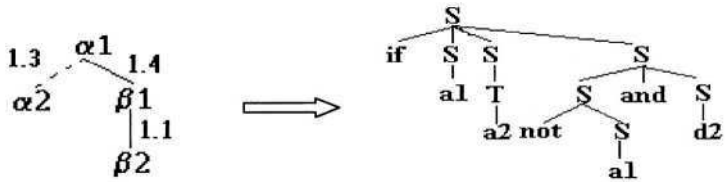


Fig. 4. Examples of a derivation tree and derived tree in TAGs.

2.2 TAG-Based Representation for Genetic Programming

The algorithm in [12, 20, 21] to find an LTAG to strongly lexicalize a CFG is based on the ideas of separation between the recursive part (structure) and non-recursive part (lexicon) of the CFG. In [21], the only operation necessary in the resultant LTAG is adjunction. However, substitution can be added to make the elementary set more compact [12]. Moreover, it is possible to encode the non-recursive parts of the grammar purely as substitution trees. In so doing, the initial tree used for substitution cannot be adjoined by other auxiliary trees: a process that simplifies the structure of derivation trees in LTAGs while maintaining their generative powers. Consequently, on the structure of LTAG derivation trees, substitution becomes an in-node operation and can be ignored to simplify the discussion of this paper (in fact, one can choose to entirely ignore substitution in implementing a TAG-based representation, at the cost of increasing the number of elementary trees). Figure 5 depicts this type of LTAG derivation tree (supposing each elementary tree has two adjoining addresses – i.e. the maximum arity is 2).

In [16], the derivation tree in LTAG was used as genotype structure for Tree-Adjoining Grammar-Guided Genetic Programming (TAG3P). TAG3P uses a genotype-to -phenotype map and can handle problems with context-sensitive syntactical constraints, context-free syntactical constraints, or (as in standard GP) no syntactical constraints. In the first case, an LTAG grammar G_{lex} is used on its own as the formalism for language bias declaration. The phenotype is the derived tree of G_{lex}. In the second case, the context-free grammar (CFG) G is used to generate the strongly lexicalised LTAG G_{lex}. The derivation trees of G_{lex} is used as the genotype, and the phenotype in that case is the derivation tree of G (derived tree of G_{lex}). In the final case a set of GP functions and terminals is used to create a context-free

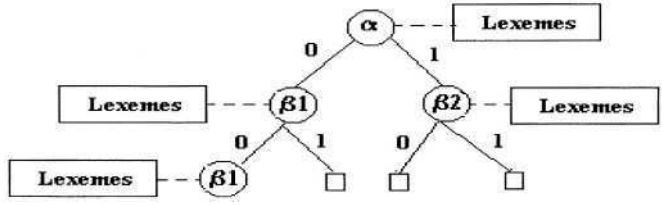


Fig. 5. Derivation tree structure for TAG-based representation. The squares means there is no tree adjoining to that address (a NULL node). Lexemes are values for the lexicons in each node.

grammar in the manner described in [29] (page 130). It was proven in [29] that there is a one to one map between the derivation trees of G and the expression trees in GP. The mapping schema can be summarized in figure 6 as follows where the second phase of the map is optional.

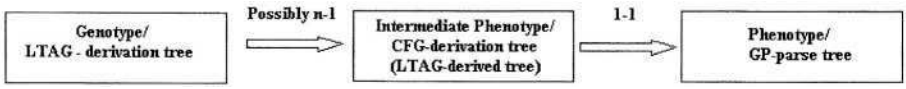


Fig. 6. Scheme for Genotype-to-Phenotype map in TAG-based Representation.

The derivation tree structure in LTAG has an important property: when growing it, one can stop at any time, and the derivation tree and the corresponding derived tree are still valid. In other words, the derivation tree in LTAG is a non-fix-arity tree structure (Catalan tree [22]). The maximal arity (number of children) of a node is the number of adjoining addresses that are present in the elementary tree of that node. If this arity is n, the node can have 0, 1,..., or n children.

In [16], this property was called feasibility. Feasibility allows us to design and implement many new search operators which would not be possible in standard GP systems, including bio-inspired ones. In particular, insertion and deletion operators arise naturally from this TAG-based representation. In insertion, a random NULL node in the LTAG-derivation tree is replaced by a new node that can adjoin to the adjoining address of the corresponding parent node. Conversely, deletion randomly deletes a node that has all NULL children in the LTAG-derivation tree (i.e. a leaf node). Insertion and deletion simulate the growth and shrinkage of a natural tree. The change in genotype structure (and consequently in phenotype structure) is small. Figure 7 illustrates how insertion and deletion work.

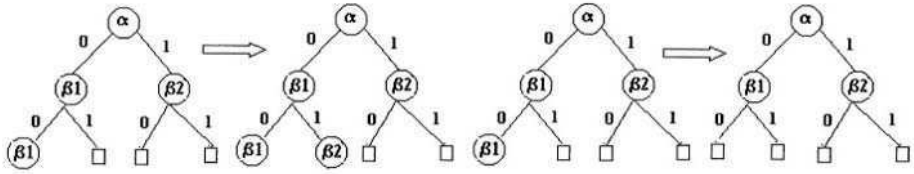


Fig. 7. Examples of insertion (on the left) and deletion (on the right).

3 Structural Difficulty in GP

In a series of works [3-8], Daida et. al. showed that structure alone can pose great difficulty to standard GP search (using expression tree representation and sub-tree swapping crossover). In particular, they delineated 4 regions of the space tree structure [6, figure 3]. Region 1 is where most of standard GP solutions lie, i.e. it is easy for GP to find a solution in that area; region 2 is increasingly difficult for GP to search; region 3, including fuller trees and thinner trees, is almost impossible for GP to search; and, region 4 is out of bounds (i.e. infeasible tree structures). Moreover regions 2 and 3 account for the majority of tree structures, even when, as is usual in practical GP, a relatively small search space bound (in terms of size or depth) is used (see [6, figure 3]).

To further validate this discovery, in their latest paper [8], Daida et al. specified a test problem known as LID. In the LID problem for GP, there is only one function of arity 2 named join, and one terminal named leaf. The raw fitness of one individual tr depends purely on its structural difference from the target solution. It is defined as follows [8].

$$\text{Fitness}_{raw}(tr) = \text{Metric}_{depth} + \text{Metric}_{terminal} \quad (1)$$

Where Metric_{depth} and $\text{Metric}_{terminal}$ are defined as:

$$\text{Metric}_{depth} = W_{depth} \times \left(1 - \left(\frac{|d_{target} - d_{actual}|}{d_{target}}\right)\right) \quad (2)$$

$$\text{Metric}_{terminal} = \begin{cases} W_{terminal} \times \left(1 - \left(\frac{|t_{target} - t_{actual}|}{t_{target}}\right)\right) & \text{if } \text{Metric}_{depth} = W_{depth} \\ 0, \text{otherwise} \end{cases} \quad (3)$$

d_{target}, and t_{target} are the depth and number of leaves of the target solution, and d_{actual} and t_{actual} are the depth and number of leaves of individual tr. In [8], W_{depth} and $W_{terminal}$ are two weighted numbers satisfying $W_{depth} + W_{terminal} = 100$. It is noted that the size s of a tree in LID problem is related to its t_{target} by the equation: $s = 2 \times t_{target} - 1$.

In [8], two families of LID problem instances were used to probe the search space of tree structure: 'horizontal cut' and 'vertical cut'. In the first family, the t_{target} was fixed as 256 and the d_{target} was varied from 8 to 255. In the second, d_{target} was fixed as 15 while t_{target} was varied from 16 to 32768. For a GP system using either size or depth as the chromosome complexity measure, these bounds on size and depth (256 and 15) are quite typical.

Surprisingly, the results in [8, figure 3 and 4] show that standard GP, using expression tree representation and sub-tree swapping crossover, performed extremely poorly on the two families of problem instances, especially for those vertical and horizontal cut regions that lie in regions 3 and 4. The results thus support the hypothesis that there is great structural difficulty in GP. Daida et. Al. [8, figures 6 and 7] went further, in showing that the results cannot be fully explained by the sparsity of tree structures in regions 3 and 4 (i.e. they are not an equilibrium problem). Their explanation pointed to the expression tree representation itself as the main cause of the structural difficulty.

Our further work on this problem has identified the lack of appropriate structural edit operators, resulting from the fixed-arity expression tree representation in standard GP, as the culprit. Using TAG-based representation we have been able to solve the problem of fixed-arity and design structural mutation operators, namely, insertion and deletion. In the next section, we investigate how the TAG-based representation, coupled with insertion/deletion operators, softens the structural difficulty in GP.

4 Experiments and Results

To investigate how well TAG-based representation and insertion/deletion operators handle the structural difficulty, we tried a hill-climbing (greedy) search using insertion/deletion as operators (TAG-HILL) on the same families of LID problem instances as in [8], namely the horizontal cut and the vertical cuts. The grammar for the LID problem is as follow:

G={N={S}, T={ Join, Leaf },P,{S}} where the rule set P is defined as follows:

$S \rightarrow S$ Join S

$S \rightarrow$ Leaf

The corresponding LTAG (as found in [21]) is G_{lex}= {V={S}, T={Join, Leaf}, I, A) where $I \cup A$ is as in Figure 8.

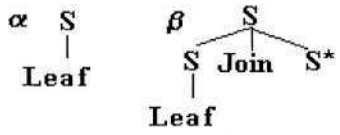


Fig. 8. Elementary trees of G_{lex} for LID problem.

It is noted in [12] that the map between derivation tree of G_{lex} and derivation tree of G (and therefore to expression tree of GP) is one-to-one. However, while the GP expression tree for LID is a fixed-arity (binary) tree, the derivation tree in G_{lex} is a non-fixed-arity (Catalan binary) tree.

In our experiments, the maximal number of steps in TAG-HILL is set to 100000 for each instance. This gives the same number of evaluations as in [8], where the size of population and the number of generations are set to 500 and 200 respectively. Consequently, the maximal allowed number of fitness evaluation in TAG-HILL is the same as in the GP experiments in [8]. For the horizontal cut, t_{target} was fixed as 256 while d_{target} was varied from 8 to 255. For each varied d_{target}, 100 runs were allocated. Similarly, in the vertical cut, the d_{target} was fixed as 15 while t_{target} was varied from 16 to 32768. For each t_{target} in [16..1024], we carried out 100 runs, while in [1025..32768], we run 100 trials for each point of the 118 equi-sampled points in that interval. Although the setting of W_{depth} and $W_{terminal}$ do not affect TAG-HILL search, we still set them the same as in [8], i.e. as 30 and 70 respectively. The size of the initial individuals is randomly chosen between 2 and 10. Both operators have equal chance of being chosen. Figures 9 and 10 depict the results on the frequency of success for TAG-HILL compared with the GP results reported in [8]. The results for TAG-HILL

are based on total 137600 runs. The graphs of GP results are adapted from figures 3 and 4 in [8] (based on 90000 runs).

The results show that TAG-HILL outperforms GP on the two families of LID problem instances by an extremely wide margin. For the horizontal cut family, TAG-HILL solved all except the three rightmost points (where the frequencies of success were 98%, 80%, 72% respectively) with 100% reliability, while GP could not reliably find solutions for the whole range of problems with $d_{target} < 12$ and $d_{target} > 70$; with $d_{target} > 100$ and $d_{target} = 8$ or 9, GP failed to find any solutions. For the vertical cut family, GP runs were unreliable in finding solutions for $t_{target} > 500$, and failed to find any solutions with $t_{target} > 1024$. By contrast, TAG-HILL can solve the problem with 100% success for t_{target} up to 13400, and only failed to find solutions when $t_{target} > 16000$. Figures 11 and 12 show the average number of search steps for TAG-HILL to find solutions. The almost linear scale suggests that, except for some extreme points, the landscape of the two families of LID problem instances is quite smooth for TAG-HILL. This is no trivial matter, since when t_{target} (d_{target}) approach their extreme values, the tree structure become exponentially sparse [23]. To see just how sparse, take the example of the leftmost point on the horizontal cut where $t_{target} = 256$ and $d_{target} = 8$. There is only one tree with that combination of (t_{target}, d_{target}) out of $\frac{4^{255}}{\sqrt{\pi 255^3}} \sim 2^{497}$ trees with t_{target} of 256 [23].

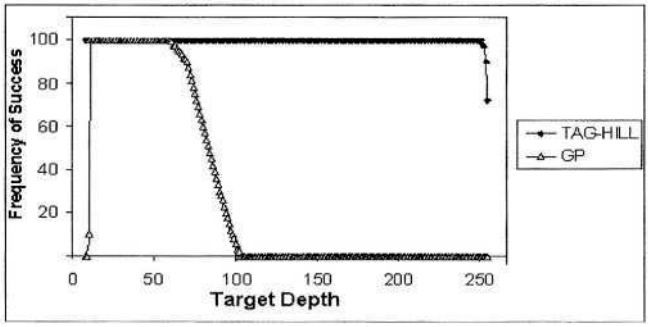


Fig. 9. Frequency of success for the 'horizontal cut'.

The results show that TAG-based representation, equipped with insertion/deletion, can soften the structural difficulties in GP. Of course, except for specially tailored problems (like LID) it may be difficult to determine whether a problem has inherent structural difficulty and hence whether to apply insertion and deletion. Further investigation of this issue will be deferred to our future work. In other recent work [17], we have investigated the role of insertion and deletion as structural mutation and/or local search operators in the context of Tree Adjoining Grammar Guided Genetic Programming (TAG3P). The results show that, on some standard GP problems, insertion and deletion help TAG3P to improve the performance significantly, even with very small population sizes.

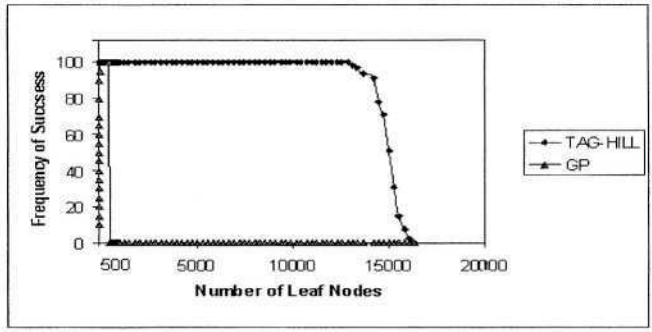


Fig. 10. Frequency of success for the 'Vertical cut'.

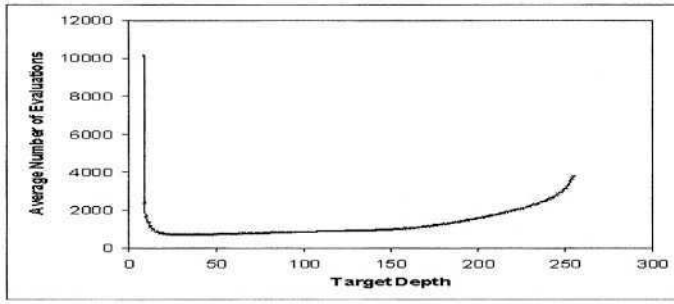


Fig. 11. Average number of Evaluations (TAG-HILL) for the 'horizontal cut'.

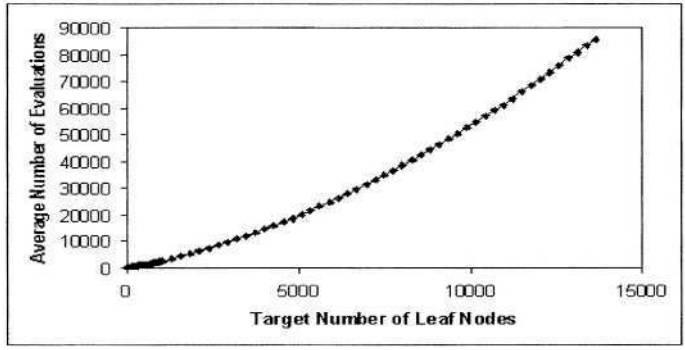


Fig. 12. Average number of Evaluations (TAG-HILL) for the 'vertical cut'.

5 Related Works

To the best of our knowledge, in the field of grammar guided genetic programming (GGGP) [9,10,18, 27-30], structural mutation and the problem of structural difficulty have not yet been studied. We believe that the structural difficulty problem also

affects the derivation tree structure of (context-free, logic) grammars (at least when solving non-syntactical constraints like in GP). However, it would be extremely difficult in GGGP to solve the problem using structural mutations like our TAG-based insertion and deletion, because of the complexity of defining them through syntactical (string-rewriting) rules.

Recently, Vanneschi et al. [24, 25] designed new GP structural mutation operators, called inflate mutation and deflate mutation, for studying fitness-distance correlation. They formally proved that their structural mutation operators are consistent with the alignment-tree metric. This means, if the alignment metric between to tree t_1 and t_2 is D, there is a sequence of inflate and deflate mutations with length D/2 to transform t1 into t2. However, it is not yet known whether this optimal sequence can be obtained in real time. Being based on the arity incremental ranking of nodes, their operators become meaningless when all the functions in GP have the same arity (as in LID), and it is hard to imagine how these operators can be extended to handling syntactically constrained domains. Moreover, their structural mutation operators are complicated in implementation and unnatural in definition.

6 Conclusion

In this paper, we have reintroduced a representation for genetic programming, based on the use of tree adjoining grammars. We described in detail how this representation transforms the fixed-arity expression tree representation in GP into a non-fixed-arity tree structure in LTAG. From this property of TAG-based representation (it was named feasibility in [16]), insertion and deletion arise naturally as structural mutation operators. The results on two families of LID problem instances, using stochastic hill-climbing search, show that the TAG-based representation significantly softens the structural difficulty previously found in standard GP using expression tree representation and sub-tree swapping crossover [8].

In future, we are planning to analyze the run time behaviour of TAG-HILL and GP (using sub-stree swapping crossover) on the LID problem, aiming at prediction of the convergence time. We are developing a way to measure the structural difficulty of problems, aiming to predict the usefulness of insertion and deletion for those problems.

References

1. Banzhaf W., Nordin P., Keller R.E., and Francone F.D.: *Genetic Programming: An Introduction.* Morgan Kaufmann Pub (1998).
2. Daida J.M., Ross S.J., McClain J.J., Ampy D.S., and Holczer M.: Challenges with Verification, Repeatability, and Meaningful Comparisons in Genetic Programming. In *Genetic Programming 1997: Proceedings of the Second Annual Conference,* Koza J. etal.(Eds), Morgan Kaufmann, (1997) 64-69.
3. Daida J.M., Bertram J.A., Polito 2 J.A., and Stanhope S.A.: Analysis of Single-Node (Building) Blocks in Genetic Programming. In *Advances in Genetic Programming 3,* Spector L., Langdon W.B., O'Reilly, and Angeline P.J. (Eds), the MIT Press (1999) 217-241.

4. Chaudri O.A., et al.: Characterizing a Tunably Difficult Problem in Genetic Programming. In *Proceedings of GECCO 2000,* Witley L.D., et al. (Eds), Morgan Kaufmann Publisher (2000) 395-402.
5. Daida J.M., Polito 2 J.A., Stanhope S.A., Bertram R.R., Khoo, J.C., Chaudhary S.A., and Chaudhri O.: What Makes a Problem GP-Hard? Analysis of a Tunably Difficult Problem in Genetic Programming. *Journal of Genetic Programming and Evolvable Machines,* vol. 2 (2001) 165-191.
6. Daida J.M.: Limit to Expression in Genetic Programming: Lattice-Aggregate Modeling. In *Proceedings of the 2002 Congress on Evolutionary Computation,* IEEE Press (2002) 273-278.
7. Daida J.M. and Hilss. Identifying Structural Mechanism in Standard GP. In *Proceedings of GECCO 2003,* LNCS, Springer-Verlag (2003) 1639-1651.
8. Daida J.M., Li H., Tang R., and Hilss A.M.: What Makes a Problem GP-Hard? Validating a Hypothesis of Structural Causes. In *Proceedings of GECCO 2003,* LNCS, Springer-Verlag (2003) 1665-1677.
9. Gruau F.: On Using Syntactic Constraints with Genetic Programming. In: *Advances in Genetic Programming* II, The MIT Press, (1996) 377-394.
10. Geyer-Schulz A.: *Fuzzy Rule-Based Expert Systems and Genetic Machine Learning.* Physica-Verlag, Germany, (1995).
11. Joshi A. K., Levy L. S., and Takahashi M.: Tree Adjunct Grammars. *Journal of Computer and System Sciences,* 10 (1), (1975) 136-163.
12. Joshi, A. K. and Schabes, Y.: Tree Adjoining Grammars. In: *Handbook of Formal Languages,* Rozenberg G. and Saloma A. (Eds.) Springer-Verlag, (1997) 69-123.
13. Koza, J. : *Genetic Programming.* The MIT Press (1992).
14. Nguyen Xuan Hoai and McKay R.I.: A Framework for Tree Adjunct Grammar Guided Genetic Programming. In: *Proceedings of Post Graduate ADFA Conference on Computer Science (PACCS'01),* H.A. Abbass and M. Barlow (Eds), (2001) 93-99.
15. Nguyen Xuan Hoai, McKay R.I., Essam D., and Chau R.: Solving the Symbolic Regression Problem with Tree Adjunct Grammar Guided Genetic Programming: The Comparative Result. In *Proceedings of Congress on Evolutionary Computation (CEC'2002)* (2002) 1326-1331.
16. Nguyen Xuan Hoai, McKay R.I., and Abbass H.A.: Tree Adjoining Grammars, Language Bias, and Genetic Programming. In *Proceedings of EuroGP 2003,* Ryan C. et al (Eds), LNCS 2610, Springer Verlag (2003) 335-344.
17. Nguyen Xuan Hoai and McKay R.I: An Investigation on the Roles of Insertion and Deletion Operators in Tree Adjoining Grammar Guided Genetic Programming. To appear in *The Proceedings of Congress on Evolutionary Computation* (CEC'2004) (2004).
18. O'Neil M. and Ryan C.: Grammatical Evolution. *IEEE Trans on Evolutionary Computation,* 4 (4), (2000) 349-357.
19. Schabes Y. and Shieber S.: An Alternative Conception of Tree-Adjoining Derivation. *Computational Linguistics,* 20 (1), (1994) 91-124.
20. Schabes Y. and Waters R. C.: Tree Insertion Grammar: A Cubic-Time Parsable Formalism that Lexicalizes Context-Free Grammar without Changing the Trees Produced. *Computational Linguistics,* 21 (4), (1995) 479-514.
21. Schabes Y.: *Mathemantical and Computational Aspects of Lexicalized Grammars,* Ph.D. Thesis, University of Pennsylvania, USA, (1990).
22. Sedgewick R. and Flajolet P: *An Introduction to the Analysis of Algorithms.* Addison-Wesley (1996).
23. Shanker V.: *A Study of Tree Adjoining Grammars.* PhD. Thesis, University of Pennsylvania, USA, 1987.

24. Vanneschi L., Tomassini M, Collard P., and Clergue M: Fitness Distance Correlation in Structural Mutation Genetic Programming. In *Proceedings of EuroGP 2003,* Ryan C. et al (Eds), LNCS 2610, Springer Verlag, (2003) 455-464.
25. Vanneschi L., Tomassini M, Collard P., and Clergue M: Fitness Distance Correlation in Genetic Programming: a Constructive Counterexample In *Proceedings of Congress on Evolutionary Computation (CEC'2003),* IEEE Press (2003) 289-296.
26. Weir D. J.: *Characterizing Mildly Context-Sensitive Grammar Formalisms.* PhD. Thesis, University of Pennsylvania, USA, (1988).
27. Whigham P. A.: *Grammatical Bias for Evolutionary Learning.* Ph.D Thesis, University of New South Wales, Australia, (1996).
28. Whigham P. A.: Grammatically-based Genetic Programming. In: *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications,* Morgan Kaufmann Pub (1995) 33-41.
29. Whigham P. A.: Search Bias, Language Bias and Genetic Programming. In: *Genetic Programming* 1996, The MIT Press, USA, (1996) 230-237.
30. Wong M. L. and Leung K. S.: Evolutionary Program Induction Directed by Logic Grammars. *Evolutionary Computation,* 5 (1997) 143-180.

πGrammatical Evolution

Michael O'Neill[1], Anthony Brabazon[2], Miguel Nicolau[1], Sean Mc Garraghy[2], and Peter Keenan[2]

[1] Biocomputing and Developmental Systems Group
University of Limerick, Ireland
{Michael.ONeill, Miguel.Nicolau}@ul.ie
[2] University College Dublin, Ireland
{Anthony.Brabazon, john.mcgarraghy, Peter.Keenan}@ucd.ie

Abstract. πGrammatical Evolution is presented and its performance on four benchmark problems is reported. πGrammatical Evolution is a position-independent variation on Grammatical Evolution's genotype-phenotype mapping process where the order of derivation sequence steps are no longer applied to nonterminals in a predefined fashion from left to right on the developing program. Instead the genome is used to specify which nonterminal will be developed next, in addition to specifying the rule that will be applied to that nonterminal. Results suggest that the adoption of a more flexible mapping process where the order of non-terminal expansion is not determined a-priori, but instead itself evolved, is beneficial for Grammatical Evolution.

1 Introduction

In standard Grammatical Evolution (GE) [1], the practical effect of the application of each rule to the non-terminals of the developing program is dependent upon all previous rule applications. As we successfully choose rules to apply to the developing program, given the tight coupling of the dependency from left to right, the greater the probability of choosing an inappropriate rule at some point during the creation of the program. Therefore, as programs become bigger the harder it will be to find the target solution due to the increased likelihood of picking sub-optimal rules. In other words, in GE, it is hard to identify clear building blocks apart from the extremely fine-grained individual production rules, although analysis of GE's ripple crossover has provided some evidence to support the useful exchange of derivation subsequences during crossover events [2]. In terms of schema theory this has serious implications for the faithful propagation of these coarse-grained building blocks, and the propagation of the more fine-grained production rule building blocks does not facilitate the creation of hierarchical structures to any great degree. It must be stressed, however, that the GE representation has been shown to be extremely robust and productive at producing solutions on a broad range of problems, seemingly exploiting the tight dependencies that exist within the representation. A representation that exploited these dependencies on a smaller scale, allowing derivation subsequences to act as building blocks, may provide more productive evolutionary search by

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 617–629, 2004.

providing better building blocks to achieve a hierarchical solution construction. It is through GE's feature of intrinsic polymorphism that derivation subsequences can be exchanged to different locations with sometimes the same or indeed different contexts.

The main idea behind position independent GE, πGE[1], is to break the dependency chain into smaller fragments that can be exchanged between appropriate contexts through a specific position independent mechanism open to evolutionary search. This should facilitate the creation of smaller, functional, building blocks, similar to sub-trees in GP, which may be easier to preserve and thus should enhance the scalability of GE to harder problem domains. Given the position independent nature of the representation it means that as long as a rule, whether structural or terminal, is present, its context can then be adapted to solve the problem at hand. In addition, it is unclear what effect the depth first non-terminal expansion adopted in the standard GE mapping process has on performance. To this end, this study compares the performance of the standard mapping ordering to πGE's mapping where the order of non-terminal expansion is explicitly evolved.

The remainder of the paper is structured as follows. The next section provides a brief background on Grammatical Evolution, with Section 3 introducing πGE. The problem domains tackled, experimental setup, and results are provided in Section 4, followed by conclusions and future work (Section 5).

2 Grammatical Evolution

Grammatical Evolution (GE) is an evolutionary algorithm that can evolve computer programs in any language [1,2,3,4,5], and can be considered a form of grammar-based genetic programming. Rather than representing the programs as parse trees, as in GP [6,7,8,9,10], a linear genome representation is used. A genotype-phenotype mapping is employed such that each individual's variable length binary string, contains in its codons (groups of 8 bits) the information to select production rules from a Backus Naur Form (BNF) grammar. The grammar allows the generation of programs in an arbitrary language that are guaranteed to be syntactically correct, and as such it is used as a generative grammar, as opposed to the classical use of grammars in compilers to check syntactic correctness of sentences. The user can tailor the grammar to produce solutions that are purely syntactically constrained, or they may incorporate domain knowledge by biasing the grammar to produce very specific forms of sentences.

BNF is a notation that represents a language in the form of production rules. It is comprised of a set of non-terminals that can be mapped to elements of the set of terminals (the primitive symbols that can be used to construct the output program or sentence(s)), according to the production rules. A simple example BNF grammar is given below, where <expr> is the start symbol from which all programs are generated. The grammar states that <expr> can be replaced with

[1] The name πGE is inspired by *Life of Pi* by Yann Martel in which a young boy Piscine Patel embarrassed by his name decides to rename himself π.

either <expr><op><expr> or <var>. An <op> can become either +, –, or *, and a <var> can become either x, or y.

```
<expr> ::= <expr><op><expr> (0)
         | <var>            (1)
  <op> ::= +                (0)
         | -                (1)
         | *                (2)
 <var> ::= x                (0)
         | y                (1)
```

The grammar is used in a developmental process to construct a program by applying production rules, selected by the genome, beginning from the start symbol of the grammar. In order to select a production rule in GE, the next codon value on the genome is read, interpreted, and placed in the following formula:

$$Rule = Codon\ Value\ \%\ Num.\ Rules$$

where % represents the modulus operator.

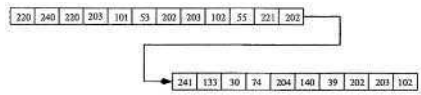


Fig. 1. An example GE individual's genome represented as integers for ease of reading.

Given the example individual's genome (where each 8-bit codon has been represented as an integer for ease of reading) in Fig.1, the first codon integer value is 220, and given that we have 2 rules to select from for <expr> as in the above example, we get 220 % 2 = 0. <expr> will therefore be replaced with <expr><op><expr>.

Beginning from the left hand side of the genome codon integer values are generated and used to select appropriate rules for the left-most non-terminal in the developing program from the BNF grammar, until one of the following situations arise: (a) A complete program is generated. This occurs when all the non-terminals in the expression being mapped are transformed into elements from the terminal set of the BNF grammar. (b) The end of the genome is reached, in which case the *wrapping* operator is invoked. This results in the return of the genome reading frame to the left hand side of the genome once again. The reading of codons will then continue unless an upper threshold representing the maximum number of wrapping events has occurred during this individual's mapping process, (c) In the event that a threshold on the number of wrapping events has occurred and the individual is still incompletely mapped, the mapping process is halted, and the individual assigned the lowest possible fitness value. Returning to the example individual, the left-most <expr> in <expr><op><expr> is mapped by reading the next codon integer value 240 and used in 240 % 2 = 0 to become another <expr><op><expr>. The developing program now looks like

<expr><op><expr><op><expr>. Continuing to read subsequent codons and always mapping the left-most non-terminal the individual finally generates the expression y*x-x-x+x, leaving a number of unused codons at the end of the individual, which are deemed to be introns and simply ignored. A full description of GE can be found in [1].

3 πGrammatical Evolution

In the first derivation step of the example mapping presented earlier, <expr> is replaced with <expr><op><expr>. Then in the standard GE genotype-phenotype mapping process, the left-most non-terminal (the first <expr>) in the developing program is always expanded first. The πGE mapping process differs in an individual's ability to determine and adapt the order in which non-terminals will be expanded. To this end, a πGE codon corresponds to the pair *(nont,rule),* where *nont* and *rule* are represented by N bits each (N=8 in this study), and a chromosome, then, consists of a vector of these pairs.

In πGE, we analyse the state of the developing program before each derivation step, counting the number of non-terminals present. If there is more than one non-terminal present in the developing program the next codons *nont* value is read to pick which non-terminal will be mapped next according to the following mapping function, where NT means non-terminal:

$$NT = Codon\ nont\ Value\ \%\ Num.\ NT's.$$

In the above example, there are 3 non-terminals (<expr>$_0$<op>$_1$<expr>$_2$) after application of the first production rule. To decide which non-terminal will be expanded next we use NT = 9 % 3 = 0, i.e., <expr>$_0$ is expanded (see second codon (9,102) left-hand side of Fig. 2). The mapping rule for selecting the appropriate rule to apply to the current non-terminal is given in the normal GE fashion:

$$Rule = Codon\ rule\ Value\ \%\ Num.\ Rules.$$

In this approach, evolution can result in a derivation subsequence being moved to a different context as when counting the number of non-terminals present we do not pay attention to the type of non-terminals (e.g. <expr> versus <op>). An alternative approach to πGE is to respect non-terminal types and only allow choices to be made between non-terminals of the same type, thus preserving the semantics of the following derivation subsequence, and simply changing the position in which it appears in the developing program. An example of this occurring can be seen in Fig. 2.

One could consider πGE to be similar in approach to the position independent GAuGE system [11] and related through its positional independence nature to Chorus [12]. However, aside from the motivational differences (i.e, to facilitate the evolution of derivation sub-sequence building blocks), there are a number of distinguishing characteristics arising from the type of position independence

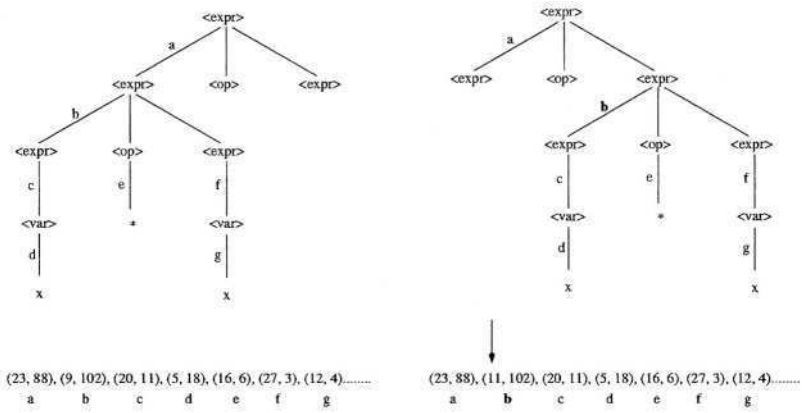


Fig. 2. On the bottom left an example πGE individuals' genome, with the 2 part 8-bit codons represented as integers for ease of reading, and its corresponding derivation tree (top left) can be seen. The right side of this figure illustrates the effect of an example mutation to the *nont* value of the second codon on the resulting derivation tree, where the subtree from the left-most `<expr>` has moved to the right-most non-terminal, also an `<expr>` in this case.

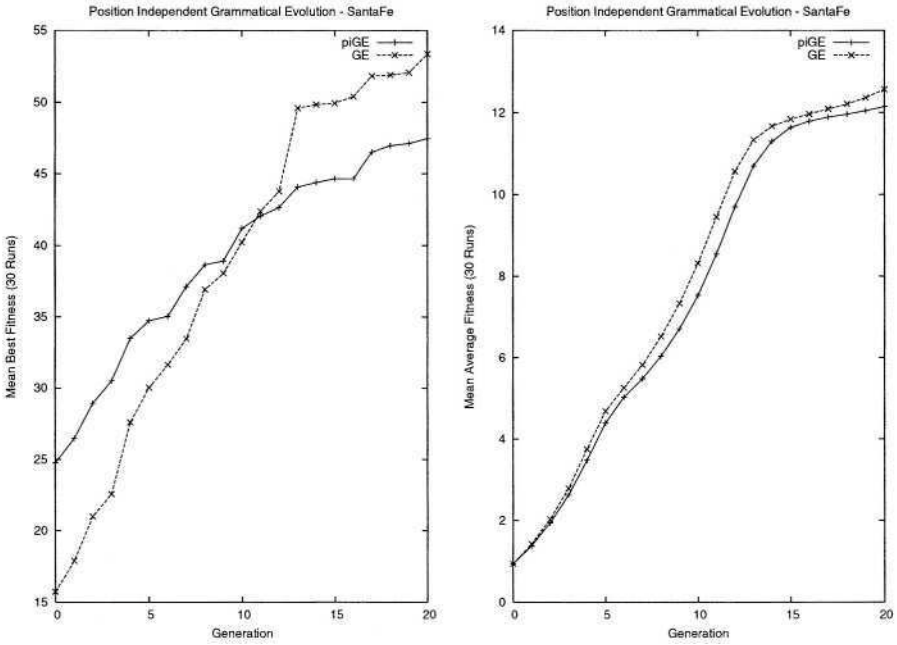


Fig. 3. Plot of the mean best and mean average fitness on the Santa Fe Ant Trail problem instance.

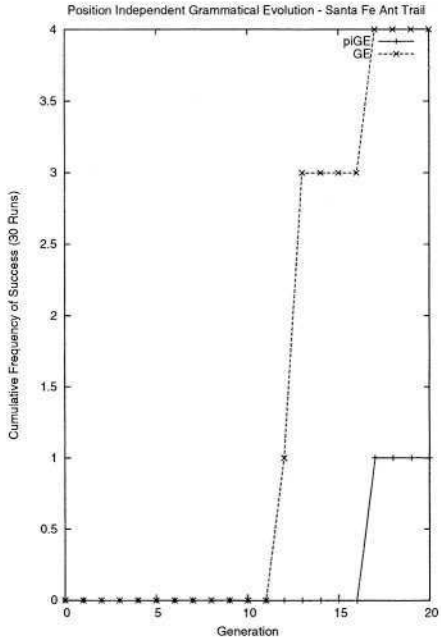


Fig. 4. Plot of cumulative frequency of success on the Santa Fe Ant Trail problem instance.

introduced. For example, the application of position independence to the order in which GE's mapping process occurs, as opposed to the order of bits in a phenotype; in the variable-length chromosomes due to the nature of the mapping process; and, in the constantly fluctuating ordering choices over the lifetime of the mapping process. Instead of making choices between a fixed number of homologous positions on a linear solution vector as in GAuGE, πGE makes choices between the variable-number of heterogeneous non-terminals that may exist in a developing program at any one point in the derivation sequence. The number of position choices available is no longer constant, instead depending on the state of the derivation sequence, with the number of available positions increasing as well as decreasing over time. Unlike in Chorus, where each gene's (referred to as codons in GE) meaning is fixed (i.e. the same gene corresponds to a specific grammar production rule irrespective of its location on the chromosomes), πGE maintains the intrinsic polymorphism property characteristic of the GE genotype-phenotype mapping. That is, a codon's meaning adapts to its context.

4 Experiments and Results

A variety of benchmark problems are tackled to demonstrate the feasibility of πGE, namely, the Santa Fe ant trail, a symbolic regression instance, mastermind, and a multiplexer instance. By feasibility of πGE, we mean that we wish to test

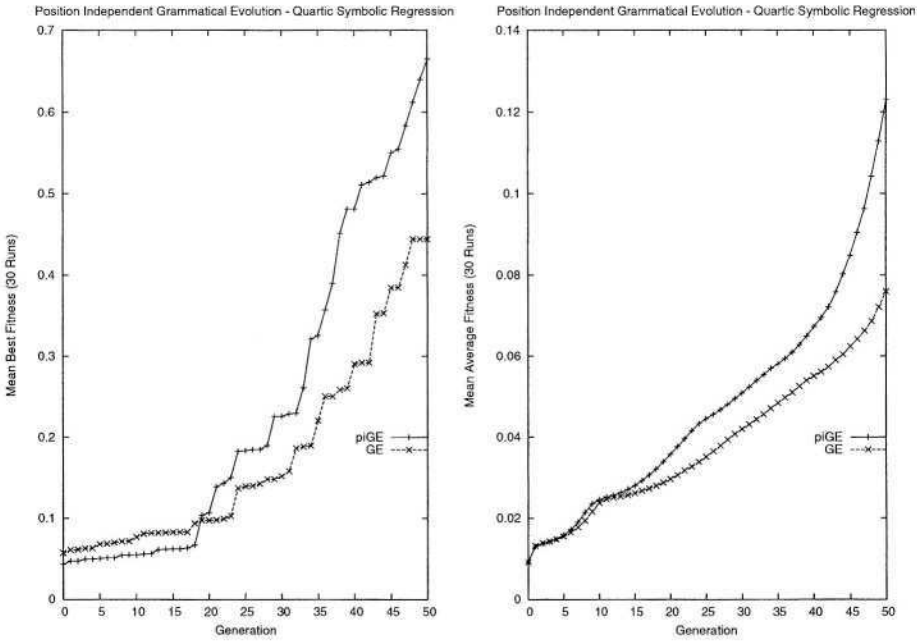


Fig. 5. Plot of the mean best and mean average fitness on the quartic symbolic regression problem instance.

if the standard depth first expansion of non-terminal symbols in a developing program can be improved upon by allowing the order of non-terminal expansion to be evolved. Performance is compared to GE on the same problem set. A random initialisation procedure is adopted that selects the number of codons in each individual to be in the range 1 to 20 codons. Wrapping is allowed with an upper limit of 10 events, and only mutation (probability of 0.01 per bit) and one-point crossover (probability of 0.9) are adopted, along with roulette selection, and a steady state replacement.

4.1 Santa Fe Ant Trail

The Santa Fe ant trail is a standard problem in the area of GP and can be considered a deceptive planning problem with many local and global optima [13]. The objective is to find a computer program to control an artificial ant so that it can find all 89 pieces of food located on a non-continuous trail within a specified number of time steps, the trail being located on a 32 by 32 toroidal grid. The ant can only turn left, right, move one square forward, and may also look ahead one square in the direction it is facing to determine if that square contains a piece of food. All actions, with the exception of looking ahead for food, take one time step to execute. The ant starts in the top left-hand corner of the grid facing the first piece of food on the trail. The grammar used in this problem

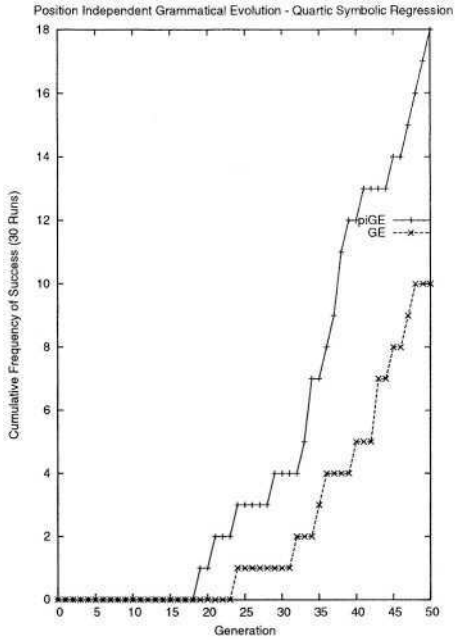


Fig. 6. Plot of cumulative frequency of success on the quartic symbolic regression problem instance.

is different to the ones used later for symbolic regression, mastermind, and the multiplexer problems in that we wish to produce a multi-line function in this case, as opposed to a single line expression. The grammar for the Santa Fe ant trail problem is given below.

```
<code> ::= <line> | <code> <line>
<line> ::= <condition> | <op>
<condition> ::= if(food_ahead()) { <line> } else { <line> }
<op> ::=  left(); | right(); | move();
```

Figures 3 and 4 shows the performance of πGE and GE on the Santa Fe ant trail problem using population sizes of 500 for 20 generations, with GE finding the greater number of correct solutions. Interestingly, πGE appears to achieve higher fitness levels earlier on in the run compared to GE.

4.2 Quartic Symbolic Regression

An instance of a benchmark symbolic regression problem is tackled in order to further verify that it is possible to generate programs using πGE. The target function is $f(a) = a + a^2 + a^3 + a^4$, and 100 randomly generated input vectors are created for each call to the target function, with values for the input variable drawn from the range [0,1]. The fitness for this problem is given by the reciprocal of the sum, taken over the 100 fitness cases, of the absolute value of the

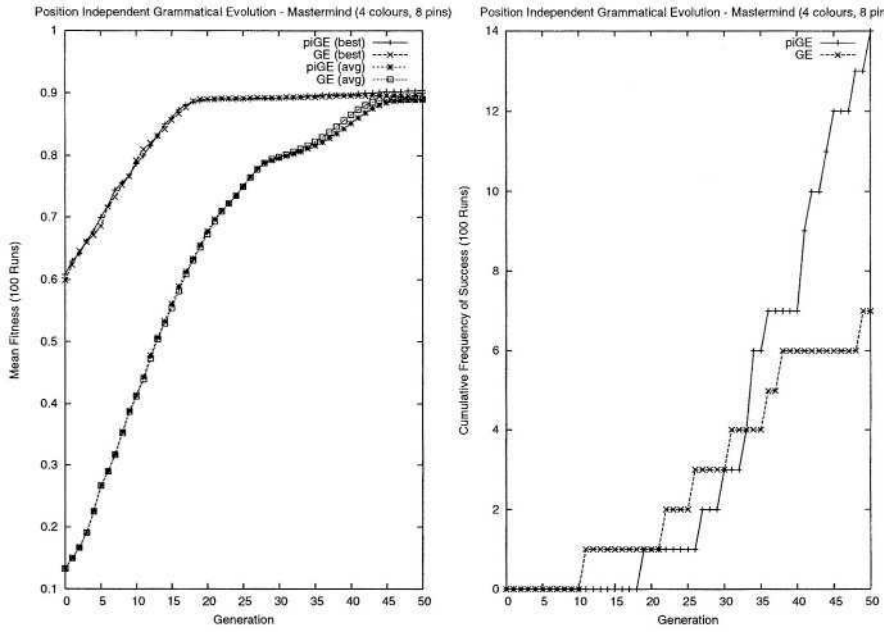


Fig. 7. Plot of the mean best and mean average fitness (left) and the cumulative frequency of success (right) on the Mastermind problem instance using 8 pins and 4 colours.

error between the evolved and target functions. The grammar adopted for this problems is as follows:

```
<expr> ::= <expr> <op> <expr> | <var>
<op> ::=  + | - | * | /
<var> ::= a
```

Results are presented for population sizes of 500 running for 50 generations in Fig's. 5 and 6, where it can be seen clearly that πGE outperforms GE.

4.3 Mastermind

In this problem the code breaker attempts to guess the correct combination of coloured pins in a solution. When an evolved solution to this problem (i.e. a combination of pins) is to be evaluated, it receives one point for each pin that has the correct colour, regardless of its position. If all pins are in the correct order than an additional point is awarded to that solution. This means that ordering information is only presented when the correct order has been found for the whole string of pins. A solution, therefore, is in a local optimum if it has all the correct colours, but in the wrong positions. The difficulty of this problem is controlled by the number of pins and the number of colours in the target combination. The instance tackled here uses 4 colours and 8 pins with the following values 3 2 1 3 1 3 2 0. The grammar adopted is as follows.

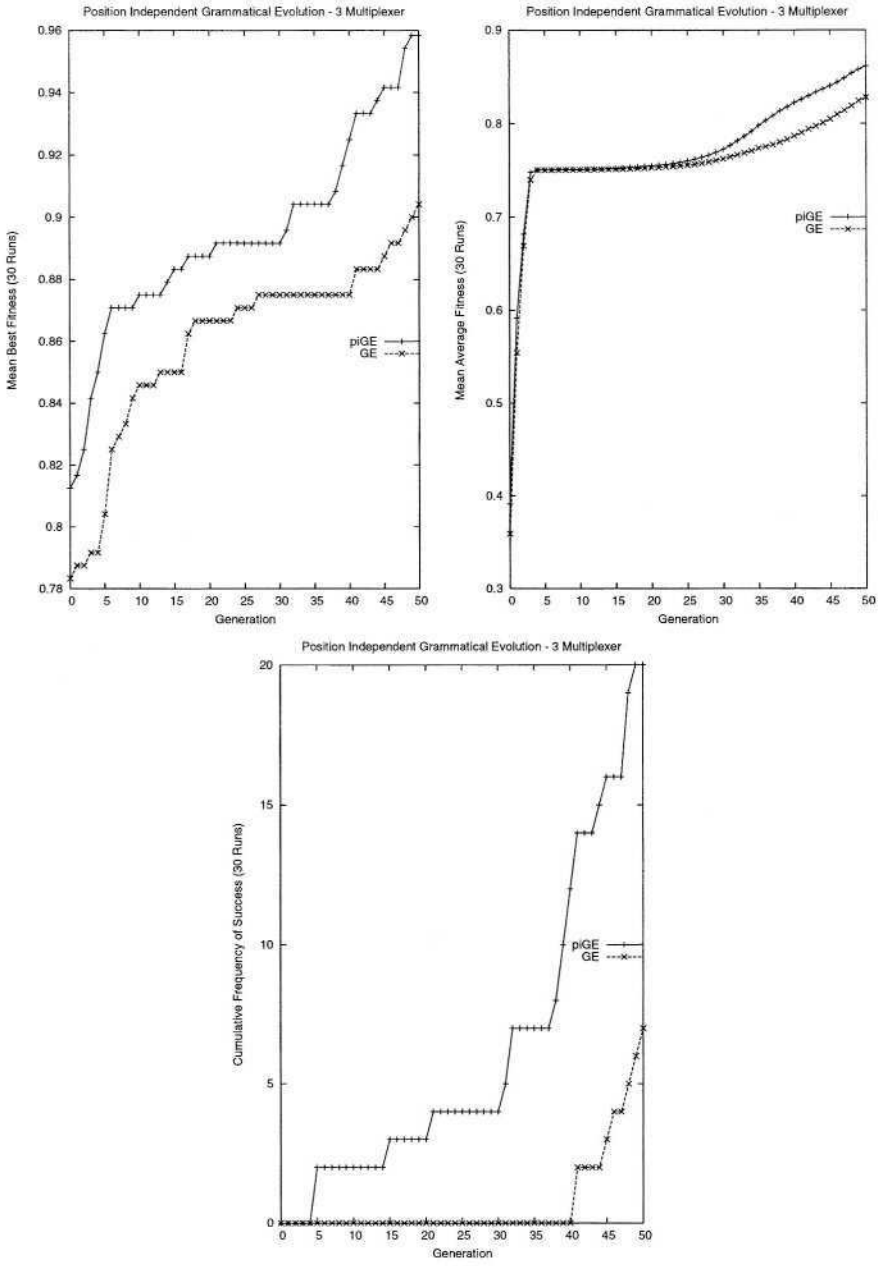


Fig. 8. Plot of the mean best and mean average fitness (top) and cumulative frequency of success (bottom) on the 3 Multiplexer problem instance.

```
<pin> ::= <pin> <pin> | 0 | 1 | 2 | 3
```

Runs are conducted for 50 generations with population sizes of 500, and the results are provided in Fig. 7.

4.4 3 Multiplexer

The aim with this problem is to discover a boolean expression that behaves as a 3 Multiplexer. There are 8 fitness cases for this instance, representing all possible input-output pairs. Fitness is the number of input cases for which the evolved expression returns the correct output. The grammar adopted is given below, and results are presented in Fig. 8 using a population size of 1000 running for 50 generations, where it can be seen clearly that πGE outperforms GE.

```
<mult> ::= guess = <bexpr> ;
<bexpr> ::= (<bexpr><bilop><bexpr>) | <ulop>(<bexpr>) | <input>
<bilop> ::= and | or
<ulop> ::= not
<input> ::= input0 | input1 | input2
```

Table 1. A comparison of the results obtained for GE and πGE across all the problems analysed.

	Mean Best Fitness (Std.Dev.)	Mean Average Fitness (Std.Dev.)	Successful Runs
Santa Fe ant			
πGE	47.47 (10.98)	12.14 (0.44)	1
GE	**53.37** (20.68)	**12.57** (0.68)	**4**
Symbolic Regression			
πGE	**.665** (0.42)	**.123** (0.07)	**18**
GE	.444 (0.4)	.076 (0.05)	10
Mastermind			
πGE	**.905** (0.04)	.89 (0.001)	**14**
GE	.897 (0.02)	.89 (0.003)	7
Multiplexer			
πGE	**.958** (0.06)	**.861** (0.04)	**20**
GE	.904 (0.05)	.828 (0.05)	7

5 Conclusions and Future Work

This study demonstrates the feasibility of the generation of computer programs using πGE, and provides evidence to support positive effects on performance using a position independent derivation sequence over the standard left to right GE mapping. A comparison of the performance of πGE with GE across all the

problems presented here can be seen in Table 1. With the exception of the Santa Fe ant trail problem, πGE outperforms GE.

The additional degree of freedom provided in evolving the choice of non-terminal to which a production rule is applied has been demonstrated to have a positive influence across the majority of problems analysed here.

Looking towards future research on πGE, we wish to analyse derivation subsequence propagation, looking at mutation and crossover events to codon *nont* values, and to undertake an analysis of the parallels between πGE, GAuGE, and Chorus. We wish to test the hypothesis that the position-independent nature of the representation might allow the formation and propagation of more useful derivation subtree building blocks as derivation subsequences can easily move between non-terminal positions in the derivation sequence.

It would also be interesting to investigate variations on the position independent mapping theme introduced by πGE, for example, to determine the effects of restricting position choices to non-terminals of the same type. It is also our intention to test the generality of the results across a number of additional problems domains.

Acknowledgements. The authors would like to thank Conor Ryan and Atif Azad for a discussion on this work.

References

1. O'Neill, M., Ryan, C. (2003). *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language.* Kluwer Academic Publishers.
2. O'Neill, M., Ryan, C., Keijzer M., Cattolico M. (2003). Crossover in Grammatical Evolution. *Genetic Programming and Evolvable Machines,* Vol. 4 No. 1. Kluwer Academic Publishers, 2003.
3. O'Neill, M. (2001). *Automatic Programming in an Arbitrary Language: Evolving Programs in Grammatical Evolution.* PhD thesis, University of Limerick, 2001.
4. O'Neill, M., Ryan, C. (2001). Grammatical Evolution, *IEEE Trans. Evolutionary Computation.* Vol. 5, No. 4, 2001.
5. Ryan, C., Collins, J.J., O'Neill, M. (1998). Grammatical Evolution: Evolving Programs for an Arbitrary Language. *Proc. of the First European Workshop on GP,* LNCS 1391, Paris, France, pp. 83-95, Springer-Verlag.
6. Koza, J.R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press.
7. Koza, J.R. (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs.* MIT Press.
8. Banzhaf, W., Nordin, P., Keller, R.E., Francone, F.D. (1998). *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications.* Morgan Kaufmann.
9. Koza, J.R., Andre, D., Bennett III, F.H., Keane, M. (1999). *Genetic Programming 3: Darwinian Invention and Problem Solving.* Morgan Kaufmann.
10. Koza, J.R., Keane, M., Streeter, M.J., Mydlowec, W., Yu, J., Lanza, G. (2003). *Genetic Programming IV: Routine Human-Competitive Machine Intelligence.* Kluwer Academic Publishers.

11. Ryan, C., Nicolau, M., O'Neill, M. (2002). Genetic Algorithms Using Grammatical Evolution. *Proc. of the 4th European Conference on Genetic Programming, EuroGP 2002,* LNCS 2278, pp. 279-288. Springer-Verlag.
12. Ryan, C., Azad, A., Sheahan, A., O'Neill, M. (2002). No Coercion and No Prohibition, A Position Independent Encoding Scheme for Evolutionary Algorithms—The Chorus System. *Proc. of the 4th European Conference on Genetic Programming, EuroGP 2002,* LNCS 2278, pp. 132-142. Springer-Verlag.
13. Langdon, W.B., and Poli, R. (1998). Why Ants are Hard. In *Genetic Programming 1998: Proc. of the Third Annual Conference,* University of Wisconsin, Madison, Wisconsin, USA, pp. 193-201, Morgan Kaufmann.

Alternative Bloat Control Methods

Liviu Panait and Sean Luke

George Mason University, Fairfax, VA 22030
lpanait@cs.gmu.edu, sean@cs.gmu.edu

Abstract. Bloat control is an important aspect of evolutionary computation methods, such as genetic programming, which must deal with genomes of arbitrary size. We introduce three new methods for bloat control: Biased Multi-Objective Parsimony Pressure (BMOPP), the Waiting Room, and Death by Size. These methods are unusual approaches to bloat control, and are not only useful in various circumstances, but two of them suggest novel approaches to attack the problem. BMOPP is a more traditional parsimony-pressure style bloat control method, while the other two methods do not consider parsimony as part of the selection process at all, but instead penalize for parsimony at other stages in the evolutionary process. We find parameter settings for BMOPP and the Waiting Room which are effective across all tested problem domains. Death by Size does not appear to have this consistency, but we find it a useful tool as it has particular applicability to steady-state evolution.

1 Introduction

When evolutionary computation uses arbitrary-sized representations, often the evolutionary process drives not only towards fitter individuals, but often dramatically larger individuals. This rapid increase in size, known as *bloat* (or as Bill Langdon calls it, "survival of the fattest") can hinder the evolutionary mechanism itself and can slow successive generations to the point that further progress is not feasible. Bloat occurs across the evolutionary computation landscape (for example, [1,2]), but most attention paid to it has been in the context of genetic programming (GP).

In this paper we will introduce three new methods for controlling bloat, *Biased Multi-Objective Parsimony Pressure, the Waiting Room,* and *Death by Size* respectively. These methods are effective, and some (such as Death by Size) are applicable to special evolutionary methods such as steady-state evolution. Because it is nearly always helpful, we combine the methods with the most common method of GP bloat-control, namely establishing fixed limits on tree depth, and compare them against using tree depth limits alone.

The dynamics of bloat are complex [3] and in the absence of a widely-accepted generalized theory of bloat, most effective methods for dealing with the problem are justified empirically. However empirical comparisons must take into consideration two simultaneous, and often contradictory, objectives: increasing fitness and reducing size. We will attempt to ascertain settings for these methods which

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 630–641, 2004.

seem to find a good middle-ground between these objectives: specifically, we view one method as better than another if its fitness is not statistically significantly worse, but its tree size is significantly smaller.

We begin with a discussion of previous approaches to bloat control. We then introduce the new methods and argue for their effectiveness in a series of experiments using various tree-style genetic programming test problems. We then conclude with a brief discussion.

1.1 Previous Approaches to Bloat Control

The most common approach to bloat control is establishing hard limits on size or depth, primarily because this approach was popularized by early work in genetic programming [4]. In such tree-based genetic programming, if a newly-created child is deeper than 17 nodes, it is rejected and a parent is introduced into the population in its stead[1]. Depth limiting has been criticized in the past for its effect on breeding [7], but it has proven a surprisingly successful method [8,9]. In the experiments in this paper we will augment our new methods with depth limiting (because in all cases it is beneficial to them), and compare this combination against plain depth limiting alone.

Outside of genetic programming, the common alternative is *parsimony pressure,* which includes the size of an individual as a factor in the selection procedure. Parsimony pressure has often been *parametric,* meaning that it considers the actual values of size and fitness together in a parametric statistical model for selection. For example, one might compute fitness as a linear function of raw fitness and size (for example, [10]) or use them together in a nonlinear fashion [2]. The problem with parametric parsimony pressure is that it requires the experimenter to state that N units of size are worth M units of raw fitness, but fitness assessment procedures are usually ad-hoc. One of several undesirable effects is that late in the evolutionary run, when all individuals have approximately the same fitness, size begins to overwhelm the selection procedure.

Recent work has instead focused on so-called "nonparametric" parsimony pressure methods, where the size and raw fitness values are not combined during the selection process. Instead, selection methods compare sizes with sizes and fitnesses with fitnesses. One example of nonparametric parsimony pressure is *lexicographic parsimony pressure* [8]. Lexicographic parsimony pressure methods compare individuals first by fitness, and then by size only if there are ties in fitness. Lexicographic parsimony pressure fails in some problem domains (notably Symbolic Regression) where large swaths of individuals have completely unique fitness values, and so size is rarely considered. The method can be improved by *bucketing* the individuals by fitness. Here individuals are sorted by fitness into N buckets, and if two individuals being compared are from the same bucket, then size is used as the comparison. This has the effect of discretizing

[1] Some depth limiting variants: [5] retries crossover some N times until it generates a valid-depth child; and [6] dynamically expands the tree depth as fitter individuals are discovered.

the fitness into classes. Related nonparametric parsimony pressure methods were discussed in [9]. Specifically, in *double tournament,* selection is done by holding several tournaments by size, and the winners of those tournaments then go on to enter a final tournament by fitness. The alternative approach is *proportional tournament,* where an ordinary tournament selection occurs, but first a coin toss of probability p is held to determine whether the tournament will select based on size or on fitness. A related technique is the *Tarpeian* method, whereby a random subset of above-average-sized individuals are simply eliminated from the population [11].

There is also some literature on treating size as an alternative objective in a pareto-optimization scheme [12,13,14]. In a pareto scheme, individual A is said to *dominate* individual B if A is as good as B in all objectives (fitness, size) and is better than B in at least one objective. This family of methods use one of several multi-objective optimization algorithms to discover the "front" of solutions which are dominated by no one else. The literature has had mixed results, primarily because nondominated individuals tend to cluster near the front extremes of all-fitness or all-size, and a highly fit but poorly sized individual is undesirable, as is a tiny but highly unfit individual. [13] has tackled this problem successfully by introducing diversity preferences into the pareto system, thereby discouraging these clusters.

2 Three New Bloat Control Methods

In this paper we introduce three new bloat control methods and examine their performance. The first method, *Biased Multi-Objective Parsimony Pressure* (BMOPP) is another variation on the pareto-optimization theme which combines lexicographic ordering, pareto dominance, and a proportional tournament. This admittedly convoluted-sounding method is in fact quite easy to implement, is effective, and (unlike many multi-objective approaches) may easily be combined with common evolutionary computation algorithms.

The remaining two methods are unusual in that size is not considered in the selection process at all, but rather it plays a factor in punishing the individual in other parts of the evolutionary cycle. In *The Waiting Room,* newly-created individuals are not permitted to enter the population until they have sat in a queue (the "waiting room") for a period of time proportional to their size. This gives small individuals an advantage in that they may spread more rapidly through the population[2]. Finally, *Death by Size* works with those evolutionary methods

[2] We mention as an aside one bloat-control approach which is related to the waiting room but which we have *not* found to be effective: asynchronous island models. Island models connect parallel evolutionary processes via a network; every so often an evolutionary processes will "migrate" individuals to other processes over the network. In an asynchronous island model the processes are not synchronized by generation, but may run at their own pace. In theory processes with smaller individuals should run faster, and hence migrate more individuals to other processes, and those migrants will tend to be small. This gives small individuals a chance to spread throughout the

which must choose individuals to die and be replaced — in our experiments, we used steady state evolution. While individuals are selected for breeding using fitness, they are selected for death and replacement using size.

2.1 Biased Multi-objective Parsimony Pressure

One of the problems with pareto-based multiobjective methods is that they consider any point along the front to be a valid candidate. Thus it is easy to generate individuals at one extreme of the Pareto front, namely individuals with small tree sizes but terrible fitness values. Usually, however, the experimenter is more interested in the other extreme: highly fit individuals. The idea behind the *Biased Multi-Objective Parsimony Pressure* (BMOPP) is to bias the search along the front towards the fitness end of the gamut.

BMOPP, like other pareto methods, uses fitness and size as its two objectives. At each generation, BMOPP places each individual into a *Pareto layer* as follows. First, individuals along the nondominated front of the two objectives are assigned to layer 1. Those individuals are then removed from consideration, and a new front is computed from among the remaining individuals. Individuals in that new front are assigned to layer 2 and removed from consideration, another new front is then computed, and so on.

After Pareto layers have been established, individuals are selected using a form of tournament selection which, with probability P, compares individuals based solely on their fitnesses, and with probability $1 - P$ compares them based on their respective Pareto layers (lower layers being preferred). Ties are broken using the alternative comparison (fitness or Pareto layer). If both individuals are identical in fitness and in layer, one individual is chosen at random. The particular value of P is of interest to us: as P increases, selection is more likely to be based on tree size and less likely to be based on fitness. If $P = 1$, BMOPP is exactly lexicographic tournament. If $P = 0$, BMOPP is entirely pareto-based.

2.2 The Waiting Room

In *The Waiting Room* newly-created individuals are punished for being large by being placed in a queue (the "waiting room") prior to entry into the population. Let the population size be N. At each generation, some RN newly-created individuals are evaluated, then added to the waiting room. $R > 1$, and so the waiting room will be larger than the final population size. Each individual in the waiting room is assigned a *queue value* equal to the individual's size. Next, the N children with the smallest queue values are removed from the waiting room and form the next generation. The remaining individuals have their queue values multiplied by a cut-down value A between 0 and 1. A prevents queue stagnation: eventually even giant individuals may have a chance to be introduced to the population.

network more rapidly. While we have not been able to get an asynchronous island model on its own to produce sufficiently parsimonious results, nonetheless we find the notion intriguing enough to be worth mentioning.

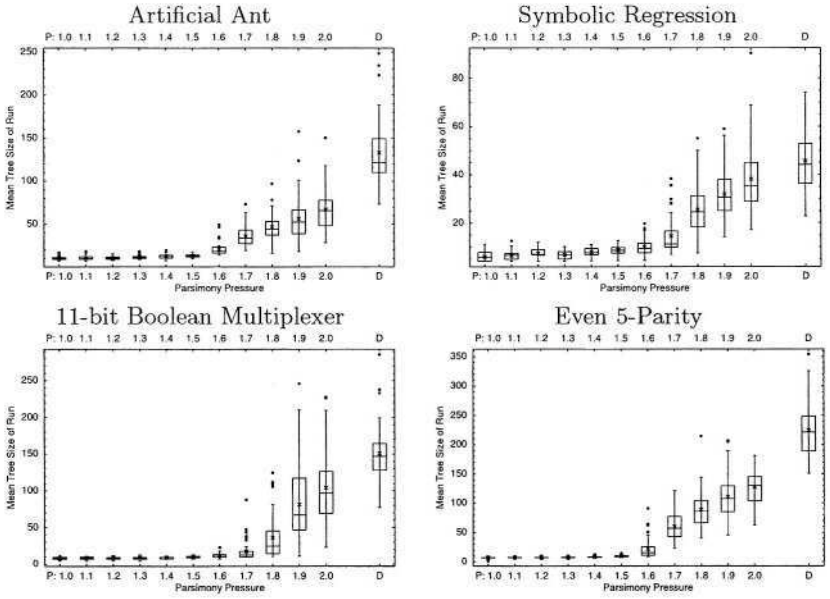


Fig. 1. Mean tree sizes for BMOPP in combination with depth limiting, as compared to depth limiting alone (labeled D). The size of the tournament selection which selects between lexicographic tournaments (layer, fitness) and (fitness, layer) is labeled *P*. The mean of each distribution is indicated with an ×.

The idea of the waiting room is that smaller individuals get to compete in the evolutionary process sooner, and thus spread (smaller) genetic material more rapidly. Our implementation of the waiting room allows it to grow without bound; but we imagine that real-world implementations would require some maximum size. Besides memory concerns, naive implementations of the waiting room impose significant computational complexity by cutting down individuals by *A* each time. We suggest using a binomial heap, and introducing new individuals to the queue by multiplying them by some increasingly large number (the inverse of *A*) rather than decreasing *A* for the existing individuals in the heap. This imposes at most an $O(\lg |\text{Waiting Room}|)$ cost per selection.

2.3 Death by Size

The two methods presented so far, and indeed all of our previously discussed methods, are designed for generational evolutionary computation. In contrast, *Death by Size* is intended for methods such as steady-state evolution which require a procedure for marking individuals for death. Death by Size is very simple: use fitness to select individuals for breeding, as usual, but when selecting an individual for death, select by size (preferring to kill larger individuals). In our

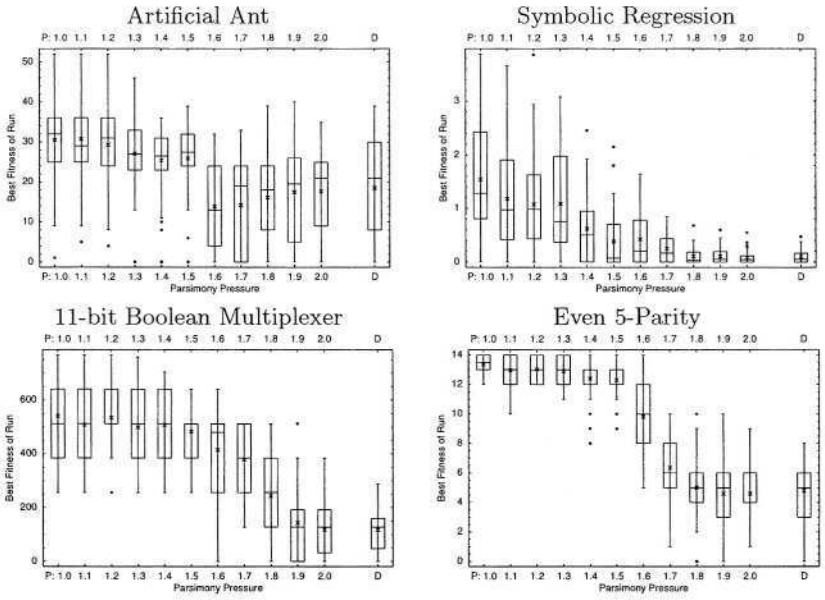


Fig. 2. Best fitnesses of run for BMOPP in combination with depth limiting, as compared compared to depth limiting alone (labeled D). The size of the tournament selection which selects between lexicographic tournaments (layer, fitness) and (fitness, layer) is labeled *P*. The mean of each distribution is indicated with an ×.

experiments we used steady-state evolution, with tournament selection for both selection procedures.

3 Experiments

The bloat-control technique most used in the literature is Koza-style depth limiting. Like most of the literature, we compare our technique against it, and maintain the traditional depth of 17 to stay relevant for comparison. As it turns out, depth limiting can be easily added to the parsimony techniques presented in this paper. As our focus is in creating efficient methods to control bloat, we decided to augment our parsimony techniques with depth limiting.

The experiments used population sizes of 1000, run for 50 generations. Similarly, the waiting room method was stopped after evaluating 50000 individuals. The runs did not stop when an ideal individual was found. Unless stated otherwise, generational GP runs used plain tournament selection of size 7. We chose four problem domains: Artificial Ant, 11-bit Boolean Multiplexer, Symbolic Regression, and Even-5 Parity. We followed the parameters specified in these four domains as set forth in [4], as well as its breeding, selection, and tree generation parameters. Artificial Ant used the Santa Fe food trail. Symbolic Regression used

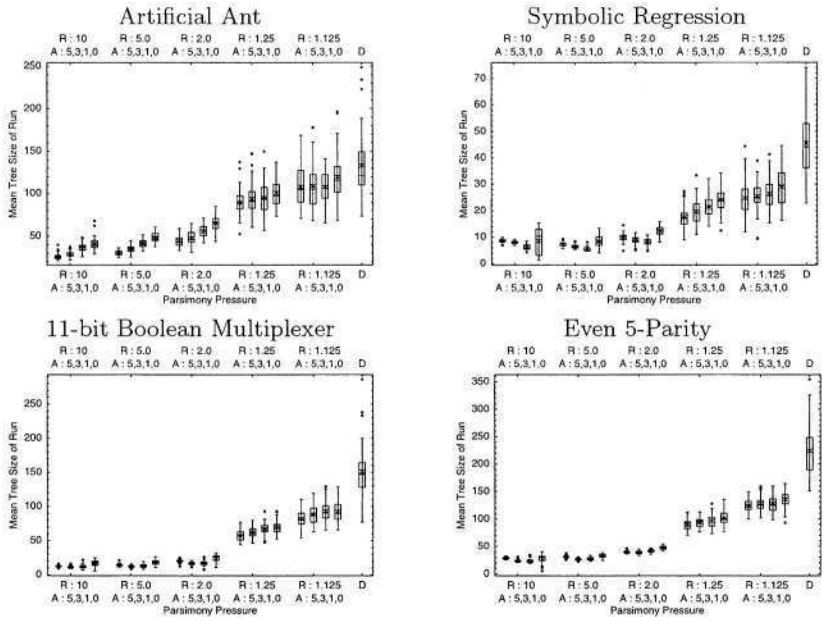


Fig. 3. Mean tree sizes for the waiting room parsimony pressure method in combination with depth limiting, as compared compared to depth limiting alone (labeled D). The ratio of waiting room to the population size is labeled *R,* and the aging rate is labeled *A*. The mean of each distribution is indicated with an ×.

the quartic polynomial fitness function and no ephemeral random constants. The evolutionary computation system used was ECJ [15]. ECJ augments standard tournament selection with a the addition of legal real-valued "tournament sizes" ranging from 1.0 to 2.0. For a tournament size P in this range, two individuals are selected, and the better is returned with probability $P/2$; with probability $1 - P/2$, a random individual is returned.

Our statistical analysis of the results used the Welch's two-sample test. In the Artificial Ant, 5-Bit Parity and 11-Bit Multiplexer domains, we compared results for each setting against the results of depth limiting alone. In the Symbolic Regression domain, we first computed ranks for the results of the two methods, and then used Welch's test directly on the ranks, in order to compensate for Symbolic Regression's non-normality in fitness. We used 95% confidence level for tests on best fitnesses of run, but used 99.98864% confidence level (derived from Bonferroni's inequality) for all tests on mean tree size. The reason for this was to be conservative when reporting better tree size at the same fitness level.

In all graphs, lower fitness is better, and the rightmost bar (labeled D) represents plain depth-limiting alone.

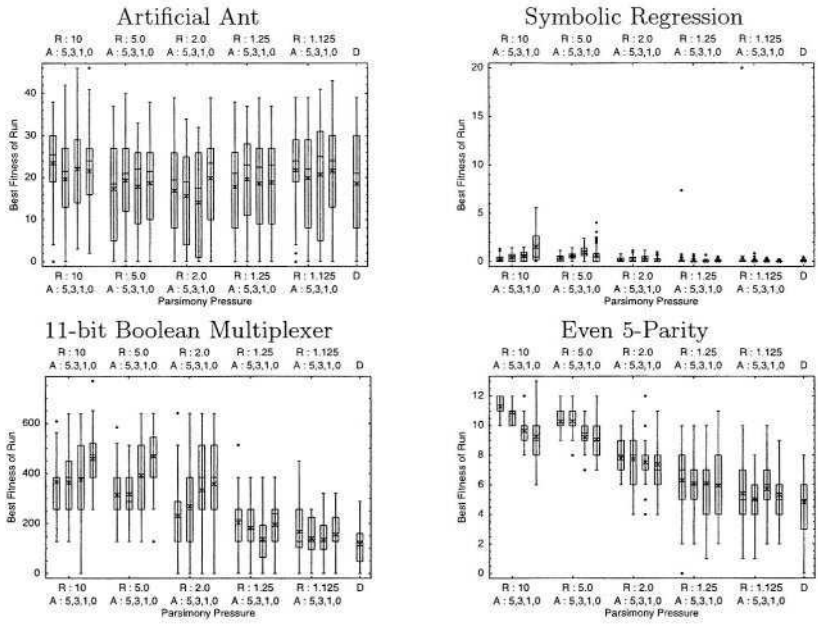


Fig. 4. Best fitnesses of run for the waiting room parsimony pressure method in combination with depth limiting, as compared compared to depth limiting alone (labeled D). The ratio of waiting room to the population size is labeled *R,* and the aging rate is labeled *A*. The mean of each distribution is indicated with an ×.

3.1 Biased Multi-objective Parsimony Pressure

Figures 1 and 2 show the mean tree size and best fitness of run for the experiments on the BMOPP method in combination with depth limiting. The extremes of tournament size (P = 1 and *P* = 2.0) represent pure pareto multiobjective optimization and pure lexicographic ordering, respectively.

The aim is to achieve lower tree sizes while at least maintaining equivalent fitness to depth limiting alone. In the Artificial Ant domain, this occurred when *P* ranged from 1.7 to 2.0. For the Multiplexer domain: 1.9 and 2.0. For Parity: 1.8 to 2.0. For Symbolic Regression: 1.5, 1.8, and 1.9.

As expected, smaller values of *P* add more parsimony pressure to the search process: this significantly reduces the tree size, but usually this comes at the expense of fitness. Overall, the mild value of 1.9 for *P* significantly reduced bloat without affecting fitness in all problem domains in our experiments.

3.2 The Waiting Room

Figures 3 and 4 show the mean tree size and best fitness of run for the experiments on the waiting room method in combination with depth limiting. There

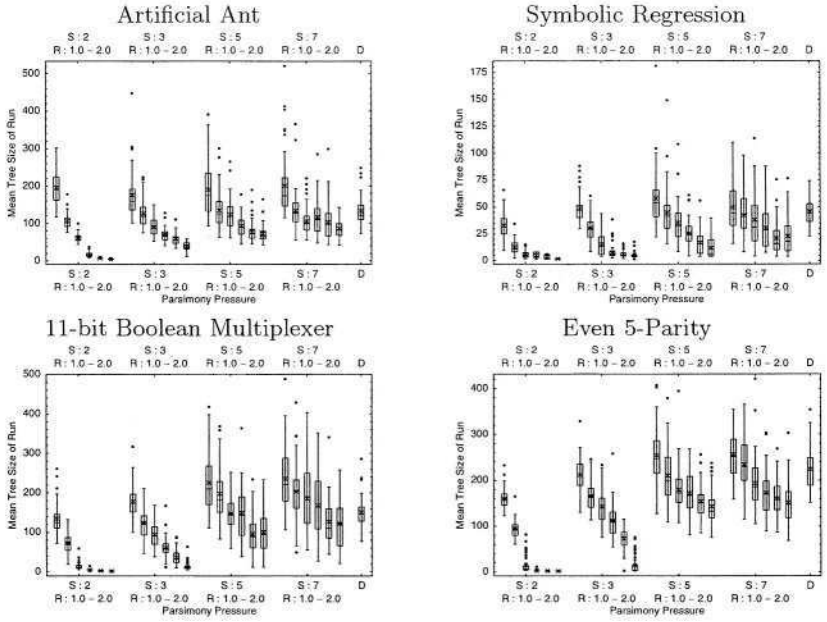


Fig. 5. Mean tree sizes for the death by size parsimony pressure method in combination with depth limiting, as compared compared to depth limiting alone (labeled *D*). The size of the selection tournament is labeled *S,* and the size of the de-selection tournament is labeled *R.* The mean of each distribution is indicated with an ×.

are two parameters: the ratio R of the size of the waiting room relative to the population size; and the cut-down parameter A. Generally smaller values of R and larger values of A resulted in smaller tree sizes.

The results in the Artificial Ant domain show no degradation in the fitness of the individuals, but the the tree size is significantly reduced in all cases with the exception of (R=10, A=5) and (R=1.125, A=5, 3, 0). Interestingly, the mean tree size of run is reduced by as much as 5 times for (R=10, A=3). The Multiplexer domain is difficult for the waiting room algorithm. Only a few settings have similar fitness and better tree sizes than depth limiting: (R=1.25, A=1) and (R=1.125, A=3, 1). The results in the 5 Bit Parity domain are similar to the ones obtained in the Multiplexer domain: only (R=1.125, A=5, 3, 0) had similar fitness and smaller trees than depth limiting alone. Symbolic Regression yielded better results: (R=1.25, 1.125, A=5, 3, 1, 0) and (R=1, A=5, 3, 0).

Overall, it appears that a small pressure for parsimony significantly reduces the mean tree size of individuals, but does not significantly hinder the performance of the search process. This implies that the size of the waiting room will not increase substantially, which is also good news. The setting (R=1.125, A=3) appears to perform well across all four domains.

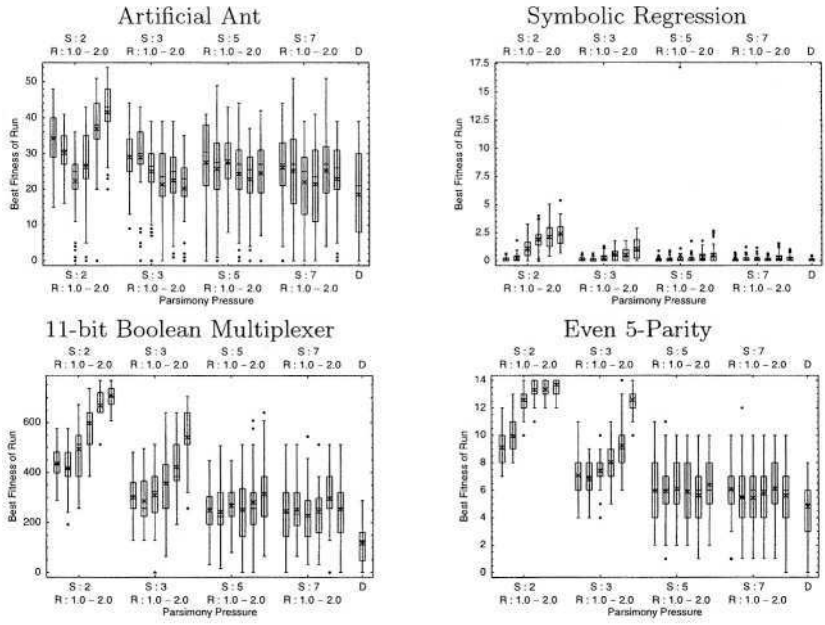


Fig. 6. Best fitnesses of run for the death by size parsimony pressure method in combination with depth limiting, as compared compared to depth limiting alone (labeled D). The size of the selection tournament is labeled *S,* and the size of the de-selection tournament is labeled *R.* The mean of each distribution is indicated with an ×.

3.3 Death by Size

Figures 5 and 6 show the mean tree size and best fitness of run for the experiments on the death by size method in combination with depth limiting. We varied two parameters: *S* is the size of the tournament selection (based on fitness) to pick individuals to breed. *R* is the size of the tournament selection (based on size) for individuals to die and be replaced.

In the Artificial Ant domain, the following *(S, R)* tournament size settings resulted in similar fitnesses to depth limiting: (2, 1.4), (3, 1.6), (3, 1.8), (3, 2.0), (5, 1.8), (7, 1.4), (7, 1.6), and (7, 2.0). Of them, (7, 1.4) and (7, 1.6) resulted in similar tree sizes, while all others settings had a smaller mean tree size. In particular, tree size was halved by the (2, 1.4), (3, 1.6), (3, 1.8) and (5, 1.8) settings, and it was reduced by a factor of three when using (3, 2.0). In the 5-Bit Parity domain, the following settings had similar best fitness of run: (7, 1.2), (7, 1.4) and (7, 2.0). Of them, only (7, 2.0) yielded significantly smaller mean tree sizes. In the Symbolic Regression domain, the following settings had both similar best fitness of run and lower mean tree size: (2, 1.0), (2, 1.2), (3, 1.2), (3, 1.4), (5, 1.6), (5, 1.8), (7, 1.6), (7, 1.8) and (7, 2.0).

Unfortunately in the Multiplexer domain, *no* setting yielded both smaller tree sizes *and* equivalent fitness values as plain depth limiting. We believe that

the Multiplexer problem domain provides a difficult testbed for steady-state approaches, and we plan to investigate this issue further.

As expected, higher values of R bias the search process more towards smaller individuals, while higher values of S bias towards higher fitness at the expense of parsimony. (7, 2.0) appears to be a good combination across all domains except for Multiplexer.

4 Conclusion

This paper introduced three new techniques for controlling bloat, each with different approaches to penalizing bloated individuals: Biased Multi-Objective Parsimony Pressure (BMOPP), the Waiting Room, and Death by Size. BMOPP uses fitness and tree size together to compute Pareto nondominated layers, which are later used in combination with fitness to compare individuals. The Waiting Room penalizes new, large children by delaying their entry into the evolutionary process. Death by Size uses a steady-state setting where bigger individuals are more likely to be replaced by newly created children.

We tested these three bloat control methods on four traditional GP benchmarks, and found that they were able to reduce the mean tree size of individuals evaluated without degradation in best fitness of run. Biased Multi-Objective Parsimony Pressure and the Waiting Room allow for settings that perform well across all four domains. The results of Death by Size suggest that the 11-Bit Multiplexer domain may be a particularly difficult testbed for steady-state approaches. However, we found settings that performed well in all other three domains, and we believe this method may be a useful tool due to its unusual application to steady-state evolution.

Our future work will provide comparative examinations of the many bloat control methods proposed in the literature. We also plan to investigate the applicability of these techniques to non-GP environments as well.

References

1. Smith, S.F.: A Learning System Based on Genetic Adaptive Algorithms. PhD thesis, Computer Science Department, University of Pittsburgh (1980)
2. Bassett, J.K., De Jong, K.A.: Evolving behaviors for cooperating agents. In: International Syposium on Methodologies for Intelligent Systems. (2000) 157–165
3. Luke, S.: Issues in Scaling Genetic Programming: Breeding Strategies, Tree Generation, and Code Bloat. PhD thesis, Department of Computer Science, University of Maryland, A. V. Williams Building, University of Maryland, College Park, MD 20742 USA (2000)
4. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA, USA (1992)
5. Martin, P., Poli, R.: Crossover operators for A hardware implementation of GP using FPGAs and Handel-C. In: GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, New York, Morgan Kaufmann Publishers (2002) 845–852

6. Silva, S., Almeida, J.: Dynamic maximum tree depth. In: Genetic and Evolutionary Computation – GECCO-2003, Chicago, Springer-Verlag (2003) 1776–1787
7. Haynes, T.: Collective adaptation: The exchange of coding segments. Evolutionary Computation **6** (1998) 311–338
8. Luke, S., Panait, L.: Lexicographic parsimony pressure. In: GECCO-2002: Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kauffman (2002) 829–836
9. Luke, S., Panait, L.: Fighting bloat with nonparametric parsimony pressure. In Merelo Guervós, J.J., Adamidis, P., Beyer, H.G., Fernández-Villacañas, J.L., Schwefel, H.P., eds.: Proceedings of the International Conference on Parallel Problem Solving from Nature (PPSN VII), Springer-Verlag (2002) 411–421
10. Burke, D.S., De Jong, K.A., Grefenstette, J.J., Ramsey, C.L., Wu, A.S.: Putting more genetics into genetic algorithms. Evolutionary Computation **6** (1998) 387–410
11. Poli, R.: A simple but theoretically-motivated method to control bloat in genetic programming. In: Genetic Programming, Proceedings of EuroGP'2003, Springer-Verlag (2003) 204–217
12. Bleuler, S., Brack, M., Thiele, L., Zitzler, E.: Multiobjective genetic programming: Reducing bloat using SPEA2. In: Proceedings of the 2001 Congress on Evolutionary Computation, IEEE Press (2001) 536–543
13. de Jong, E.D., Pollack, J.B.: Multi-objective methods for tree size control. Genetic Programming and Evolvable Machines **4** (2003) 211–233
14. Ekart, A., Nemeth, S.Z.: Selection based on the pareto nondomination criterion for controlling code growth in genetic programming. Genetic Programming and Evolvable Machines **2** (2001) 61–73
15. Luke, S. ECJ 10 : An Evolutionary Computation research system in Java. Available at http://www.cs.umd.edu/projects/plus/ec/ecj/ (2003)

Robotic Control Using Hierarchical Genetic Programming

Marcin L. Pilat[1] and Franz Oppacher[2]

[1] Department of Computer Science, University of Calgary,
2500 University Drive N.W., Calgary, AB, T2N 1N4, Canada
pilat@cpsc.ucalgary.ca
[2] School of Computer Science, Carleton University,
1125 Colonel By Drive, Ottawa, ON, K1S 5B6, Canada
oppacher@scs.carleton.ca

Abstract. In this paper, we compare the performance of hierarchical GP methods (Automatically Defined Functions, Module Acquisition, Adaptive Representation through Learning) with the canonical GP Implementation and with a linear genome GP system in the domain of evolving robotic controllers for a simulated Khepera miniature robot. We successfully evolve robotic controllers to accomplish obstacle avoidance, wall following, and light avoidance tasks.

1 Introduction

Genetic programming (GP) is a powerful evolutionary search algorithm to evolve computer programs. However, its main drawback is that it tries to solve the entire problem at once. While canonical GP is suitable for smaller problems, it is often not powerful enough to solve difficult real world problems. Hierarchical genetic programming (HGP) tries to overcome the weakness of GP by introducing a divide-and-conquer approach to problems. Instead of solving a complex problem entirely, the method breaks the problem into smaller parts, generates solutions to the parts, and integrates the solutions into the solution to the overall problem. This modularization approach is based on what humans do to solve complex tasks.

Several hierarchical genetic programming methods have been proposed in the literature, each with its own advantages and disadvantages. In our research, we examine the methods of Automatically Defined Functions (ADFs) [4], Module Acquisition (MA) [2], and Adaptive Representation through Learning (ARL) [8].

We study the application of genetic programming techniques to the evolution of control programs for an autonomous miniature robot. We are interested in the evolution of robotic controllers for the obstacle avoidance, wall following, and light avoidance tasks. Nordin and Banzhaf [6] have studied the use of linear genome GP [5] for real time control of a miniature robot. Our research extends the analysis of [6] by carrying out experiments with the ADF, MA, and ARL hierarchical GP methods, the canonical tree-based GP implementation [3], and a variation of the linear genome GP system. Our results enable us to compare the performance of the studied method in the domain of robotic control.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 642–653, 2004.

2 Robotic Controllers

Nordin and Banzhaf [6] have experimented with a simulated and real Khepera miniature robot to evolve control programs with genetic programming. They used the Compiling Genetic Programming System (CGPS) [5] which worked with a variable length linear genome composed of machine code instructions. The system evolved machine code that was directly run on the robot without the need of an interpreter.

The initial experiments of Nordin and Banzhaf were based on a memory-less genetic programming system with reactive control of the robot. The system performed a type of function regression to evolve a control program that would provide the robot with 2 motor (actuator) values from an input of 8 (or more) sensor values. GP successfully evolved control programs for simple control tasks of obstacle avoidance, wall following, and light-seeking. The work was extended [6,7] to include memory of previous actions and a two-fold system architecture composed of a planning process and a learning process. Speed improvements over the memory-less system were observed in the memory-based system and the robots exhibited more complex behaviours [6].

The GP system in the robotic controller evolves control programs that best approximate a desired solution to a pre-defined problem. This procedure of inducing a symbolic function to fit a specified set of data is called Symbolic Regression [6]. The goal of the system is to approximate the function: f(s0, s1, s2, s3, s4, s5, s6, s7) = {m1, m2} where the input is comprised of 8 robotic sensors (s0-s7) and the output is the speed of two motors controlling the motion of the robot (m1-m2). The control program code of each individual constitutes the body of the function. The results are compared with a behaviour-based fitness function that measures the accuracy of the approximation by the deviation from desired behavior of the robot.

In our research, we evolve a population of control programs for the Khepera robot. The GP variants studied here are steady-state tournament selection systems with tournament size of 7. Experiments are performed with populations of size 50 to 200. The algorithm executes until desired behaviour is learned or until 300 generations elapse. We use our simulator - Khepera GP Simulator for Microsoft Windows® - for all our experiments. The simulator contains a user friendly interface and allows run-time modification of all simulation parameters, fitness functions, and robotic controllers. The simulator and its C++ source code are available free-of-charge for academic research purposes [1].

2.1 Linear Genome GP

The linear genome GP system with binary machine code was introduced in [5] as Compiling Genetic Programming System (CGPS). The method was used to evolve a robotic controller for Khepera robots [6]. The structure of our linear genome GP controller closely resembles the controller used by Nordin and Banzhaf.

[1] Khepera GP Simulator home page is available at http://www.pilat.org/khepgpsim.

The controller by Nordin and Banzhaf [6] used variable length strings of 32 bit instructions for a register machine performing arithmetic operations on a small set of registers. We represent each instruction as a text string and process it by a Genome Interpreter prior to evaluation. A loss in performance is noticed since processing of the string based instructions is more time intensive than for numerical representations. However, the performance of the text-based representation is sufficient for the purpose of the research and provides greater readability.

Each individual is composed of a series of instructions (genes). The instructions are of the form: $resvar = var1\ op\ \{var2|const\}$ where *resvar* is the result variable and *op* is a binary operator working on either two variables (*var*1 and *var*2) or a variable and a constant (*var*1 and *const*). Values of instruction parts are selected randomly from a set of primitive values. Valid variable values are 8 robotic infrared proximity sensors $(s0 - s7)$, 8 ambient light sensors $(l0 - l7)$ and intermediate variables $(a - f)$. The operator set consists of: *add, subtract, multiply, leftshift, rightshift, XOR, OR, AND.* Constants are randomly chosen from the range 0 – 8191.

The linear genome GP method employs three genetic operators: reproduction, crossover and mutation. Variable length 2-point crossover is applied to the two fittest individuals of a tournament, according to a given probability. Genes are treated as atomic units by the crossover operator and are not internally modified. If crossover is not selected, reproduction occurs and copies of fittest individuals are inserted into the new population. Simulated bit-wise mutation modifies the contents of a gene. We use crossover probability of 0.9 and mutation probability of 0.05.

2.2 Tree-Based GP

The tree-based GP system works with the canonical tree representation of chromosomes. Program trees for the tree-based GP method and all HGP methods are created randomly with the "full" method [3] to generate the initial random population. Maximum tree height at creation is 6 and maximum overall tree height is 10. The standard function set is composed of functions: *Add, Sub, Mul, Div, AND, OR, XOR, <<, >>, IFLTE.* The terminal set consists of variables denoting the robotic sensors $(s0 - s7$ and $l0 - l7)$ and constants in range 0 – 8192.

Reproduction and crossover are the genetic operators for this method. Single subtree switching crossover is applied to the two fittest individuals in a tournament, with a given probability. For the tree-based GP method and all HGP methods, we use crossover probability of 0.9 in our experiments and no mutation.

2.3 Automatically Defined Functions HGP

The Automatically Defined Function (ADF) method in our research is based on the method proposed by Koza [4]. The method automatically evolves function definitions while evolving the main GP program.

The result producing branch is built with the standard terminal set and the standard function set augmented with the ADFs contained in the same chromosome. Separate terminal and function sets are used by the function defining branches to define the ADFs. These secondary sets contain dummy variables to denote the arguments to function calls. All ADFs defined in an individual are only available locally to the program tree of the same individual. We study ADF program trees with one, two, and three ADF definitions and two ADF arguments.

As in the tree-based method, reproduction and crossover operators are used in the ADF chromosomes. To preserve the structure of valid ADF chromosomes, the crossover operator is only allowed to swap non-invariant nodes and nodes of similar type using branch typing.

2.4 Module Acquisition HGP

The Module Acquisition (MA) method of Angeline and Pollack [2] uses two extra operators of compression and expansion to modularize the program code into subroutines. The program tree structure is identical to the structure of the tree-based representation. The function set of each chromosome is extended by parameterized subroutines called modules. Modules are local to a chromosome and are only propagated through the population by reproduction and crossover.

The Module Acquisition method employs four genetic operators: reproduction, crossover, and two mutation operators of compression and expansion. The reproduction and crossover operators are similar to those used by the canonical tree-based representation.

The compression operator creates a new subroutine from a randomly selected subtree of an individual in the population using depth compression [1] of maximum depth range 2 – 5. A subroutine call replaces the chosen subtree in the program tree and the new module extends the function set of the chromosome. If the chosen subtree is beyond the maximum depth value, a parameterized subroutine is created. The expansion operator is complementary to the compression operator and restores a single module call to its original subtree. We use compression probability of 0.1 and expansion probability of 0.01.

2.5 Adaptive Representation HGP

The Adaptive Representation through Learning (ARL) method by Rosca and Ballard [8] extends the canonical tree-based GP system by introducing parameterized blocks (i.e. functions). Unlike in the ADF approach, the functions are discovered automatically and without human-imposed structure. The method differs from the MA approach by the algorithms for function discovery and the management of function libraries.

The structure of the ARL chromosome program trees is identical to the tree structure of the tree-based GP method. The ARL functions are stored in a global library and are accessible to all the chromosomes in the population.

The function library is dynamically grown by the execution of the evolutionary algorithm. Standard tree reproduction and crossover operators are used.

The implementation relies on differential fitness [8] for function discovery. We do not use the concept of block activation [8] because of the large performance overhead on the system. We employ a simple measure of subroutine utility [8] by assigning to each subroutine an integer utility value. This value is decremented each generation the subroutine is not used until the subroutine is removed from the library.

Population entropy provides a measure of the state of a dynamic system represented by the population and is calculated by using Shannon's information entropy formula [9] on groupings of individuals based on their phenotype. Our entropy calculation is not based on raw fitness values because of the dynamic nature of our fitness calculation. We implement a standardized classification measure to calculate a value for individual classification. The value is calculated using an average of motor output values obtained from three test cases with fixed input sensor values. Individuals are partitioned into 20 categories based on the value of their standardized classification measures. Population entropy value is then calculated by Shannon's formula applied to the categorized chromosomes.

The measure of population entropy is important since it correlates to the state of diversity in the population during a GP run [9]. We use a static entropy threshold value of 1.5 to decide on the initiation of a new epoch and discovery of new subroutines. Candidate blocks of height 3 are selected from most promising individuals. Discovered functions are automatically inserted into the function set. To counteract the drops in population entropy, the ARL method generates random individuals using the new function set. The new individuals are placed into the existing population. The number of new individuals to create is specified by the replacement fraction parameter which is set to 0.2 in our experiments.

3 Results

3.1 Obstacle Avoidance

The task of obstacle avoidance is very important for many real-world robotic applications. Any robotic exploratory behavior requires some degree of obstacle avoidance to detect and manoeuver around obstacles in the environment. We define obstacle avoidance as the behavior to steer the robot away from obstacles. For the Khepera robot, this task is equivalent to minimizing the values of the proximity sensors while moving around the environment.

Our fitness function for obstacle avoidance is based on the work of Nordin and Banzhaf [6]. The pleasure part of the fitness function is computed from motor values and encourages the robot to move around the environment using straight motion. The pain part is composed of sensor values and punishes the robot for object proximity. The fitness function can be expressed as the equation: $fitness = \alpha\ (|m_1|+|m_2|-|m_1-m_2|)\ -\ \beta \sum_{i=0}^{7} s_i$ where m_x are motor values (in range -10 to 10) and s_x are proximity sensor values (in range 0-1023). In our experiments, scale parameters are set to $\alpha = 10$ and $\beta = 1$.

Various robotic behaviours are observed while learning the obstacle avoidance task and are summarized in Figure 1. We subdivide the learned behaviours into groups based on the complexity and success rate of each behavior. The simplest Type 1 behaviours (straight, curved) are solely based on the simple movement of the robot with no sensory information. The second level of behaviour, Type 2, is composed of circling, bouncing, and forward-backup behaviours. The highest level of behaviour, Type 3, is called sniffing and it involves processing sensor data to detect and avoid obstacles. Avoidance is learned either by circling away from an obstacle or by backup behaviour. The perfect sniffing behaviour produces smooth obstacle avoidance motion around the entire testing environment.

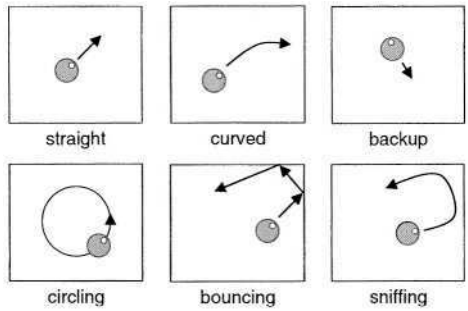


Fig. 1. Summary of learned behaviours.

We used the measures of entropy stability and average chromosome size stability to compare the studied methods. We define stability by a gradual change of measured values and lack of large abrupt changes. For the obstacle avoidance task, the ARL method provides the most stable entropy values. The MA and tree-based methods provide the worst stability with large drops of entropy values. Best chromosome size stability is seen with the linear genome method and worst with the MA method. Among the HGP methods, the most stable method is the ARL method.

The results with Type 2 and 3 behaviours are processed to calculate average generation values of first occurrence of the stated behaviour. Summary of the results of our behaviour calculation can be found in Figure 2. The method with best initial behaviour occurrence values is the ARL HGP method and with the worst is the linear genome GP method. Overall, the HGP methods perform comparable with the tree-based GP method. A sample experimental trace of evolved perfect obstacle avoidance behaviour can be seen in Figure 3.

3.2 Wall Following

The task of wall following allows the robot to perform more difficult and interesting behaviours such as maze navigation. The purpose of the task is to teach

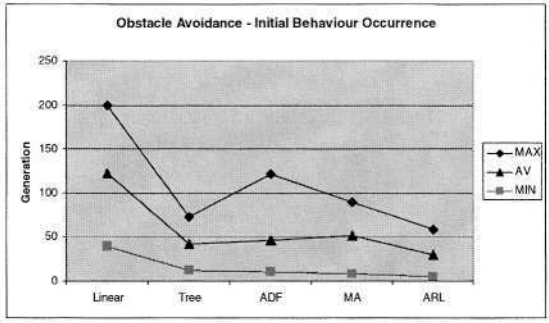


Fig. 2. Graphs of minimum, maximum, and average generations of first detection of Type 2 and 3 obstacle avoidance behaviour for each chromosome representation method.

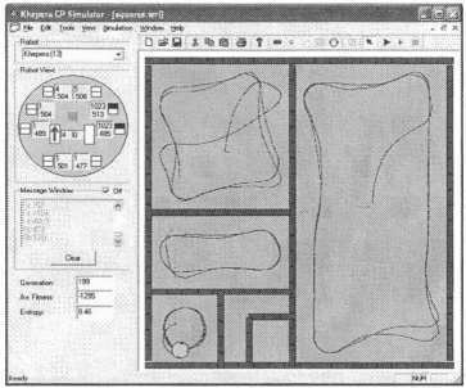


Fig. 3. Trace runs of perfect evolved obstacle avoidance behaviour in various testing environments.

the robot to move around the environment with a certain desirable distance away from obstacles. The learned task should include some obstacle avoidance behaviour; however, that is not the main requirement of the experiments.

The wall following fitness function is composed of sensor and motor parts. The sensor part is computed from a subset of the robotic sensor values and the motor part is defined as the absolute motor value sum minus the absolute value of the difference. The fitness function can be summarized as: $\text{Fitness} = \alpha \cdot MotorPart + \beta \cdot SensorPart$. The calculated sensor part value acts as either pleasure or pain depending on the values of the sensors. Thus, the robot is punished when it is either too far away from an obstacle or too close to it.

Summary of the behaviours obtained is provided in Figure 1. We categorize the behaviours based on their relative performance and success. The first Type 1 category is of poor wall following behaviour and consists of simple wall-bouncing and circling behaviours. The Type 2 category of good wall following behaviour

consists of wall-sniffing and some maze following. The best behaviour category, Type 3, consists of perfect maze following without wall touching and usually taking both sensor sides of the robot into consideration.

The most stable entropy behaviour is noticed in experiments using the ARL method whereas the least stable is observed using the ADF method. Most stability of average size values is seen in the linear genome and ARL methods. The largest drops in average chromosome size are noticed with the MA method.

Type 2 and 3 behaviour category results are processed to calculate average generation values of first occurrence of the stated behaviour. Summary of the results can be found in Figure 4. The ARL method produces the best overall results with the smallest deviation. The worst performance and largest deviation is seen with the MA method. A sample experimental trace of evolved perfect maze following behaviour can be seen in Figure 5.

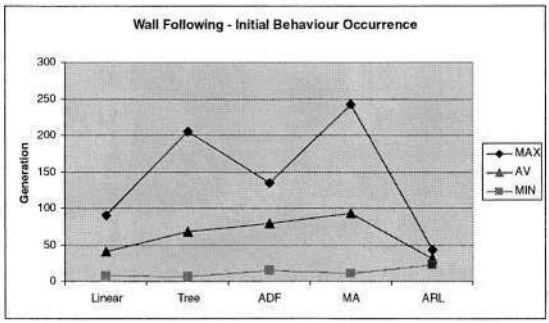


Fig. 4. Graphs of minimum, maximum, and average generations of first detection of Type 2 and 3 wall following behaviour for each chromosome representation method.

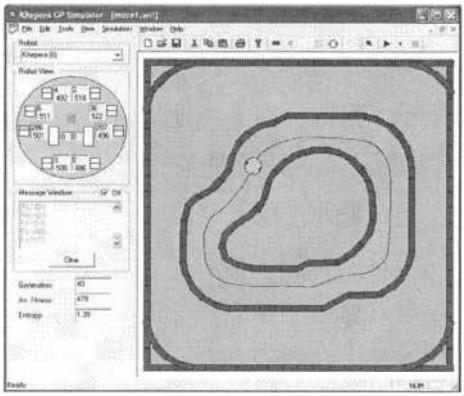


Fig. 5. Trace run of perfect evolved maze-following behaviour.

3.3 Light Avoidance

The light avoidance task is similar to the obstacle avoidance task but uses the ambient light sensors of the robot instead of the proximity sensors. Light sources in the training and testing environments are overhead lamps that cannot be touched by the robot. The robot must learn to stay inside an unlit section of the world environment while moving as much as possible.

The fitness function for light avoidance is similar to the fitness function used for obstacle avoidance. The function contains a pleasure part computed from the motor values of the robot and a pain part computed from the light sensors. The function can be expressed as: Fitness $= \alpha$ ($|m_1|+|m_2|-|m_1 - m_2|$) - β (4000 - $\sum_{i=0}^{7} l_i$) where m_x are motor values and l_x are ambient light sensor values (in range 0-500). In our experiments, scale parameters are set to $\alpha = 10$ and $\beta = 1$. The constant 4000 denotes the highest possible sensor sum and transforms the fitness function to behave similar to the fitness function of the obstacle avoidance task.

We subdivide the learned behaviours into two categories. The Type 2 category consists of behaviours with some degree of light detection and avoidance. The Type 3 behaviour category consists of behaviours with definite light detection and avoidance. Perfect behaviour usually consists of movement around the boundary of the dark area in the testing environment.

The most stable entropy behaviour is noticed with the linear genome method and worst stable with the ADF method. Most stable average size values are observed using the ARL and tree-based methods. The worst average size behaviour is seen with the MA method.

Type 2 and 3 categories are processed to calculate average generation values of first occurrence of the stated behaviour. Summary of the results can be found in Figure 6. The best results come from experiments with the ARL method and the worst from linear genome experiments. The HGP methods perform comparable to the tree-based method. A sample experimental trace of perfect evolved light avoidance behaviour can be seen in Figure 7.

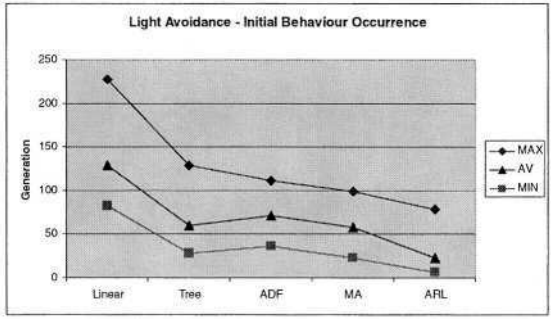


Fig. 6. Graphs of minimum, maximum, and average generations of first detection of Type 2 and 3 light avoidance behaviour for each chromosome representation method.

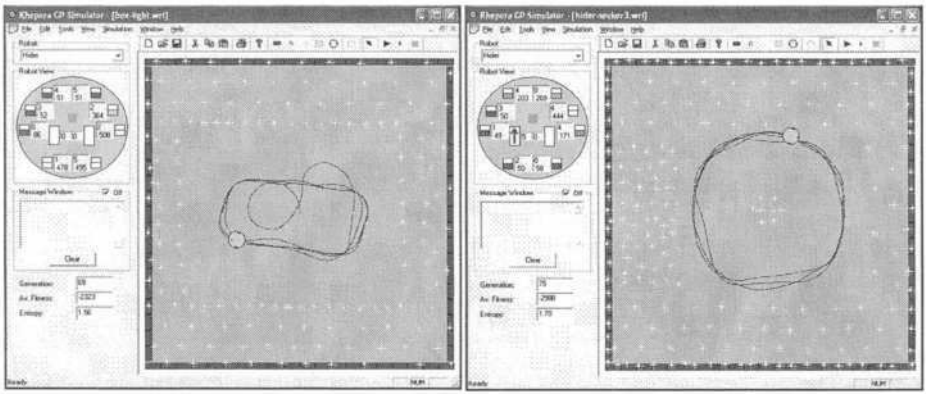


Fig. 7. Trace runs of perfect evolved light avoidance behaviour in different testing environments.

4 Conclusion

Our research is concerned with the evolution of robotic controllers for the Khepera robot to perform tasks. The reactive robotic control problem provides a challenge to the genetic programming paradigm. With the lack of test cases for fitness function evaluation, the fitness of an individual can differ greatly depending on the immediate neighbourhood of the robot. From tuning experiments, we notice that the definition of the fitness function can greatly influence the population contents and thus the resulting behaviours. Because of the constantly changing local environment, even good performing behaviour can eventually be replaced by worse behaviours. We feel that more testing of fitness functions and their parameters should be done to identify the optimal fitness function definition for each learning task.

We have noticed that the robotic controllers often over-adapt to the training environment. This problem of overfitting is a common problem in genetic programming. We notice that sharp corners of the environment form an area of difficulty for the robotic controller. This difficulty is probably due to the corner fitting between the fields of view of the proximity sensors.

The entropy value is an important indicator of population diversity in our experiments. We notice that best behaviours are observed in populations with relatively high entropy value (above 0.6). Low entropy value signifies convergence in the population which usually accompanies a convergence to a low average chromosome size. Populations of suboptimal individuals with low chromosome size do not contain enough information to successfully search for a good solution.

We study three HGP learning methods: Automatically Defined Functions, Module Acquisition, and Adaptive Representation through Learning. Robotic controllers using each method are able to evolve some degree of proper behaviour for each learning task. Summary of method performance is available in Table 1.

The ADF method uses a predefined, constant function set containing one or more ADFs. The only method of function call acquisition is through crossover with another individual of the population. The ADFs inside individuals showing nice behaviour are usually quite large and complex with no noticeable patterns. It is possible that in our experiments, the ADFs have not much purpose other than to provide few extra tree levels of instructions. In our experiments, the ADF method performance was usually below that of the tree-based method. We notice that the ADF trees do not shrink enough before the population prematurely converges. We think that it would be best to specify a smaller initial and maximum size of the ADF trees so that the functions require less time to find optimal configurations.

Table 1. Summary of results from our experiments for each of the studied methods. Behavioural performance was based on first occurrence of good behaviour. Performance of each method is scored relative to performance of other methods.

Method	Entropy Stability	Size Stability	Behavioural Performance
Linear GP	excellent	excellent	poor
Tree GP	average	average	average
ADF HGP	poor	average	average
MA HGP	average	poor	average
ARL HGP	excellent	excellent	excellent

The slowest function set growth is observed with the MA method. Most of the individuals in the populations with good behaviour do not use any of the functions in the module set. The creation of functions produces program size loss which in turn often lowers the entropy of the population. The behavioural performance of the MA method is usually worse than that of the tree-based method. Since no strong mechanism exists to counteract the loss of program size and accompanying loss of entropy, the population often converges prematurely to suboptimal solutions. Probability-based compression and expansion operator invocation might be replaced by need-based operator invocation similar to those found in the ARL method. This new operator invocation should lead to better behaviour through adjustments of operator frequencies based on population needs.

The ARL method displays the most stable entropy and average chromosome size behaviour in most experiments. This behaviour is observed only when function creation occurs, thus we think that the function creation and new individual creation processes are responsible for the stability. The method also achieves the best time and smallest deviation to reach good behaviour in most experiments. The size of the function set of the ARL method is very dynamic through the runs of the algorithm. Many of the functions from populations with good behaviour seem to calculate some form of ratio of the function parameters, thus, we think that some of the evolved functions were of benefit to the individuals. Influx of random individuals to the population during evolution can lead to problems

since too many random individuals would destabilize good solutions present in individuals of the previous population. We think that a low replacement fraction used with elitism of best individuals should produce the optimal evolutionary balance. Elitist individuals would always be copied into the population and would ensure the fittest individuals are not lost between generations.

Our results indicate that reactive memoryless controllers can be trained to exhibit some level of proper behaviour for the studied tasks. An extension to this research would be to study memory-based robotic controllers that can store previous actions and use them to decide future behaviour. Such controllers using the linear genome method have been shown in [6] to successfully and quickly evolve more complex behaviours than a memoryless controller.

We would also like to use a real Khepera robot to verify our results. Physical robots train in a noisy and sometimes unpredictable environment and would provide a real world test case for our research.

References

1. Angeline, P.J.: Genetic Programming and Emergent Intelligence. In K.E. Kinnear, Jr. (ed.): Advances in Genetic Programming, chapter 4. MIT Press (1994) 75–98
2. Angeline, P.J., Pollack, J.B.: The Evolutionary Induction of Subroutines. In: Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society. Lawrence Erlbaum, Bloomington, Indiana, USA (1992)
3. Koza, J.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA (1992)
4. Koza, J.: Genetic Programming II: Automatic Discovery of Reusable Programs. MIT Press, Cambridge, MA (1994)
5. Nordin, P.: A Compiling Genetic Programming System that Directly Manipulates the Machine-Code. In K.E. Kinnear, Jr. (ed.): Advances in Genetic Programming, chapter 14. MIT Press, Cambridge, MA (1994) 311–331
6. Nordin, P., Banzhaf, W.: Real Time Control of a Khepera Robot using Genetic Programming. Cybernetics and Control. **26**(3) (1997) 533–561
7. Nordin, P., Banzhaf, W., Brameier, M.: Evolution of a world model for a miniature robot using genetic programming. Robotics and Autonomous Systems. **25**(1-2) (1998) 105–116
8. Rosca, J.P., Ballard, D.H.: Discovery of Subroutines in Genetic Programming. In P.J. Angeline and K.E. Kinnear, Jr. (eds.): Advances in Genetic Programming 2, chapter 9. MIT Press", Cambridge, MA, USA (1996) 177–202
9. Rosca, J.P.: Entropy-Driven Adaptive Representation. In J.P. Rosca (ed.): Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications. Tahoe City, California, USA (1995) 23–32

A Competitive Building Block Hypothesis

Conor Ryan, Hammad Majeed, and Atif Azad

Biocomputing and Developmental Systems Group
Computer Science and Informations Systems Department,
University of Limerick, Ireland.
{Conor.Ryan, Hammad.Majeed, Atif.Azad}@ul.ie

Abstract. This paper is concerned with examining the way in which rooted building blocks grow in GP-like systems. We hypothesize that, in addition to the normal notion of co-operative building blocks, there are also *competitive* building blocks in the population. These competitive building blocks are all of the rooted variety, all share a similar root structure and compete with each other to spread their particular extensions to the common structure throughout the population. We demonstrate that not only do these competitive building blocks exist, but that they work in tandem with non-rooted co-operative building blocks.

1 Introduction

The traditional view of building blocks in Evolutionary Algorithms in general, and Genetic Algorithms in particular, is of their parallel discovery by a population. Different individuals in the same population may contain separate, non-overlapping building blocks which, when recombined with each other, produce a new, larger building block, usually with an associated increase in fitness. Building blocks are looked upon as being *co-operative,* as an individual can increase its fitness simply by accumulating more of them.

There is much work in the literature on building block exchange [5] [4] and the design of the *deceptive* family of problems [2] [3] was motivated in part to examine how GAs first discover, and then exploit and share building blocks.

There has also been some work in this area in GP [8], although the tree structures employed by GP complicate the matter [14], as the manner in which the meaning of nodes often depends on the way in which they are used, and the nodes above them. In particular, the root and other nodes around it dictate how the other nodes operate. This means that a subtree from one part of an individual will not necessarily operate the same way or give the same contribution to fitness if moved somewhere else.

This paper is concerned with testing the *competitive* building block hypothesis [18], which states that, for GP like structures, the most important building blocks are rooted ones. During evolution, as time moves on, an increasing number of individuals will have an increasingly large root structure in common, so the manner in which this common structure increases is vitally important to the

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 654–665, 2004.

success or otherwise of a run. Potential modifications to the common root structure can be looked upon as a *competition,* with different extensions being tried across the population. Once a useful extension has been discovered, it competes with other extensions for control of the population, and, in this manner, the new rooted building block extends through the population.

We use Grammatical Evolution(GE) [16] [12] to test this hypothesis, but also to test whether it is possible that *co-operative* building blocks exist in that system. GE employs a genotype-to-phenotype mapping (GPM) that, like GP, places more importance on genes (nodes) at the start of an individual. However, because of the way the mapping works, genes are interpreted in context, so their meaning can change, in an apparently more abrupt way than in GP, when the context changes. Earlier work [13] has shown that crossover still works in a sensible manner despite this, but did not attempt to capture the manner in which genes required in the final *best-of-run* individual could exist with different interpretations in the population.

2 Building Blocks

Building blocks are the cornerstone on which Genetic Algorithms are built. Most GAs assume the existence of short collections of genes (building blocks) which contribute to the fitness of individuals. Typically, when combining two non-overlapping building blocks in an offspring individual, one can expect to derive some fitness from both of building blocks that have been contributed.

However, constructing increasingly larger building blocks is not always this straightforward. In particular, the *epistatic* effect, where two or more genes contribute to a single trait, makes the discovery of a building block more difficult. For example, consider a genome G of length n. If loci G_0 and G_1 must be set to 1 to achieve an increase in fitness,then one cannot consider either on its own to be a building block. This was described by Goldberg [4] as *minimal superior building blocks.*

Many GA researchers have tried to capture characteristics of search spaces by examining or even designing building blocks. Examples include Royal Road functions from Mitchell et. al. [10], in which the perfect solution is made up of a hierarchical assembly of non-overlapping building blocks. It is argued that these functions are difficult for hill climbing algorithms as the fitness remains unchanged until the correct assembly of the complete building block. Thus, the search algorithm remains in the dark until the sudden discovery of a correct building block. Another, quite different set of examples are Deceptive Functions [2] [3], in which the building blocks are specifically designed to mislead search algorithms. In a typical case the algorithm finds a linear passage towards a local optimum which is a 1's complement of the global optimum. Goldberg [4] also provides a formula to verify if the problem is deceptive on average at schemata of all the orders below that of a fully specified building block. Higher order deceptive functions are generated by concatenating two or more deceptive functions together.

2.1 Competitive or Co-operative Building Blocks

Despite their apparent differences, one crucial trait that the two problems above have in common is their use of non-overlapping building blocks. In fact, virtually all "advanced" GAs (e.g. competent GAs [4]) rely on problems having this characteristic, in order to ensure that exchange and recombination of building blocks is possible.

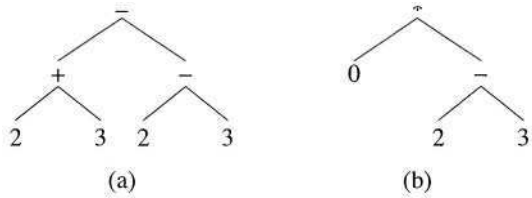


Fig. 1. Simple GP individuals in which the root node affects the outcome of the entire individual.

However, this usually is not the case in GP. Typically, when an individual is evaluated, the last node to be evaluated is the root, and all other nodes are dependent on it. Consider the two individuals in Fig. 1. Individual (a) evaluates to 6, but if the subtrees **(+ 2 3)** from (a) and **(- 2 3)** from (b) are swapped, the result is -6, even though the subtrees still evaluate to the same thing. Thus, it is not possible to assign credit to a particular subtree because, unless it is used in *exactly* the same way, one can not be guaranteed that it will have the same effect. Similarly, individual (b) is such that no subtree swapped into the right hand side of the tree will have an effect on the overall outcome due to the multiplication operator at the root.

Clearly then, the root node is the most important node in a tree, but what are the implications of this for evolution? Work on the "Eve Phenomenon" [9] demonstrated that in around 90% of GP runs there was a *single* ancestor from which *every* individual in the final population descended. Further, they demonstrated that, on average, 70% of the population shared the same *four* levels.

The *Competitive Building Block Hypothesis* follows on from this, and conjectures that a major part of the search effort in GP concentrates on correctly identifying how best to extend this common area. It views every member of the population as an attempt by GP to extend the area, and that much of the useful exploration being carried out by GP at any given time occurs in this area. Fig. 2 indicates the three areas; *(i)* the common area, in which little or no useful exploration happens, *(ii)* the area of discovery, where most of the useful discoveries are made, and *(iii)* the tails, where useful discoveries are less likely to be maintained, due to their distance from the fringe of the common area.

This paper is concerned with measuring the rate at which the common area grows, and the extent to which information in the tails and on the fringe of the area of discovery get incorporated into the common area. We provide evidence

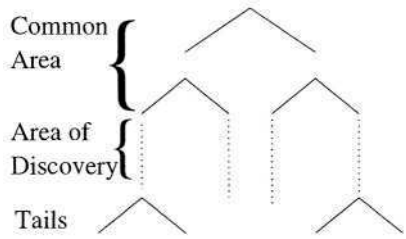


Fig. 2. The different areas of discovery for GP individuals. Root nodes are typically the most stable, while processing carried out at the leaves is likely to be modified by higher nodes.

to show that the competitive building block hypothesis exists, but also that the traditional co-operative building block hypothesis also holds, and that useful information can be held in subtrees lower down in individuals.

To aid our analysis, we employ the Grammatical Evolution system. GE is useful in this case because its linear structures are relatively simple to analyse, and its property of *intrinsic polymorphism* (see the following section) captures the dynamics of subtrees changing meaning with context.

3 Background

Grammatical Evolution [12] is a Genetic Programming system that combines the convenience of a GA style binary string representation with the expressive power of a high level language program. Following a biological metaphor the binary string is termed as the *genotype* that is translated into a program or the *phenotype* through a genotype-phenotype mapping process. The mapping process commonly uses a context free grammar(CFG) represented in Backus Naur Form (BNF). However, the modular design of this evolutionary algorithm permits the use of other types of grammars and notations, for example, Adaptive Logic Programming system (ALP) [6] combines the context sensitive programming features of Prolog logic programs with GE and demonstrates many interesting applications.

As mentioned earlier, CFG is the most widely used grammar in GE and it is represented by a tuple $\{T, N, P, S\}$. T is the set of terminals, the symbols that occur in the valid sentences of the language defined by the grammar. N denotes the set of non-terminals, the interim items that lead to the terminals. P is a set of production rules and $S \in N$ is the start symbol. A grammar typically used by GE for symbolic regression problems is given below.

```
S = <expr>
<expr>    ::= (<expr> <op> <expr>) | <pre-op> (<expr>) | <var>
<op>      ::= / | - | * | +
<pre-op>  ::= Sin | Cos | Log | Exp
<var>     ::= x | 1.0
```

The mapping process chooses different rules from the grammar to arrive at a particular sentence. The genotype is treated as a sequence of 8 bit genes. The mapping starts with the start symbol. At each step during the mapping process a gene is read from the genome to pick a rule for the non-terminal under consideration. Use is made of the **mod** operation to decode a gene into a rule in the following manner.

Rule index = (Gene) Mod (Number of rules for the particular non-terminal)

To elucidate the mapping process consider a sample individual **24 32 14 19 136 7 8 128 19 21.** Start symbol <expr> has the following options.

```
<expr> ::= ( <expr> <op> <expr> )   (0)
         | <pre-op> ( <expr> )      (1)
         | <var>                    (2)
```

24 mod 3 = 0. Thus, (<expr> <op> <expr) is chosen. The mapping process always resolves the left most non-terminal. <expr> being the same non-terminal as before the set of options remains unchanged. The mapping proceeds by reading the next gene **32** that decodes to <var>. The expression now becomes (<var> <op> <expr>). For <var> the set of choices is:

```
<var> ::= x       (0)
        | 1.0     (1)
```

14 mod 2 = 0. Thus, <var> is replaced by x in the expression. The mapping continues in this manner until the individual is completely mapped to (x + Exp (x)). The genetic material is reused if the mapping is incomplete at the end of a single pass through the individual. The phenomenon is termed as *wrapping* and discussed comprehensively in [17].

3.1 Ripple Effect and Ripple Crossover

The use of **mod** operation ensures that a gene is always interpreted *in context* of the mapping process. As a result a gene always decodes to a rule that is used immediately. This property is termed *intrinsic polymorphism* [7]. Consequently the meaning of a gene depends on all the genes that precede it in the chromosome. If a change occurs in the earlier part of the chromosome, the effect *ripples* through the rest of the genome. Thus, the one point crossover in GE is termed as the *ripple crossover.* Fig. 3 exemplifies the ripple crossover for the individual discussed previously. When the chromosome is cut at a certain location, it effectively dismantles multiple branches from the derivation tree leaving behind many *ripple sites.* The in-coming fragment may be of different length and may contain the genes used for different non-terminals. The polymorphic interpretation ensures that they are used in context to fill in the ripple sites.

3.2 Building Blocks in GE

The previous section may make it appear as though it is unlikely that building blocks can exist in GE, as they not only have the same property in GP that the

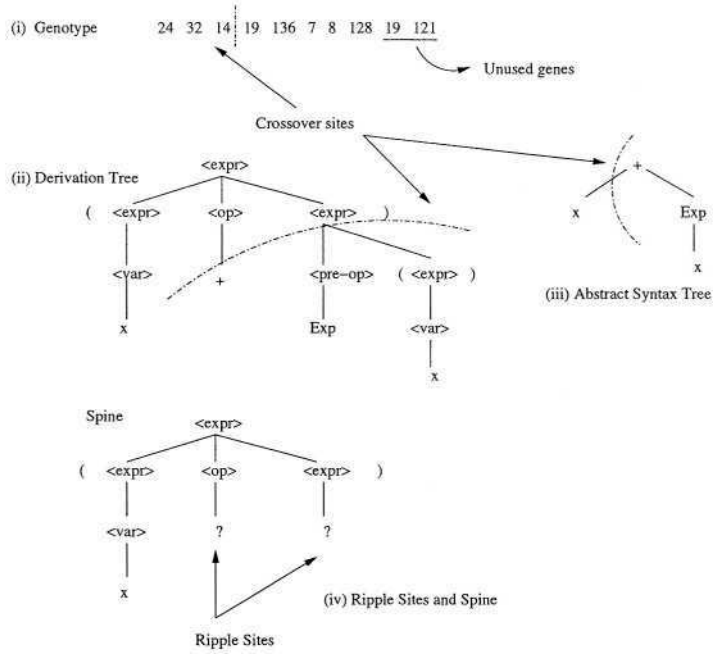


Fig. 3. Ripple crossover in GE. The change in the chromosome in (i) is correlated with the corresponding changes in the derivation tree and the abstract syntax tree in (ii) and (iii) respectively. (iv) depicts the spine and the vacant ripple sites.

operation of their phenotype depends to a large extent on the ones that preceded them (in the same manner that nodes in GP depend on their parent nodes), they can actually code for something entirely different.

An investigation in whether or not building blocks appear in GE was carried out by [11] in which they compared a variety of crossover operators, including the so-called *headless chicken* [1] crossover, in which randomly generated material is inserted into parents rather than having them exchange genes. They showed that a homologous crossover, that prevented individuals from choosing crossover points within their common areas performed approximately the same as the standard one point crossover, suggesting that a type of homologous crossover comes for free with GE. Our belief is that, while this is true, it is more the case that many of the *successful* crossovers are of this variety.

Rooted or unrooted building blocks? Rooted building blocks are clearly of importance to any type of GP system, but the fact that headless chicken crossover usually under-performs compared to standard crossover suggests that there have to be unrooted building blocks at play as well.

The following section presents a suite of experiments designed to test whether crucial building blocks from ideal individuals appear in the wrong position (and possibly with an entirely different meaning).

4 Experimental Design

The first experiment was to test if important building blocks appear in the population, but in the wrong location on individual's chromosomes. Important building blocks are defined as those that appear in the *best-of-run* individual for a particular run.

A second experiment was designed to investigate how newly extended rooted building blocks spread throughout the population. If there really is a competition to discover a rooted building block, we should be able to see evidence of the appearance of increasingly larger rooted building blocks, which then start to take over the population.

We used Koza's quartic polynomial symbolic regression problem, with a population of 500 running for 200 generations, averaged over 30 runs, using steady state replacement with roulette wheel selection. One point crossover was employed and probability was set to 0.9. Mutation was turned off; this was to permit us to focus on the effects of crossover alone. It has been shown [15] that the performance of GE drops off very sharply when mutation is removed, and we experienced a similar drop off in performance. In fact, for all the experiments here, we only experienced two successes, even though they ran for two hundred generations.

4.1 Occurrence of Incorrectly Positioned Building Blocks

These experiments were designed to check the frequency of existence of a building block at different positions throughout the population. The best individual was selected and compared to the entire population of a particular generation. Two counts were maintained, the first shows the frequency of existence of a building block at the *same* position in the whole population as it was in the best individual, while the second count keeps track of the frequency of the building block at *different* positions across the whole population. This test explains how the building blocks are spread across the genome. As the size of the building block was not known before hand, we checked the block sizes starting from one to the maximum size of the best individual. Due to space considerations we only show measures of building block size nine, as this is representative of the trend shown with other sizes. These are shown in Figs. 4 and 5. Every building block of length nine was counted, in an overlapping fashion, so the first building block occupies gene positions 0 to 8, while the second occupies 1 to 9.

As indicated by these figures, as early as after the first generation, the population (Fig. 4) has, on average, produced around seven individuals with the correct top root structure. The further along the genome we examined, the less likely a building block was to be found in its correct position - recall that we are interested in *individual* building blocks in the traditional sense rather than rooted building blocks. Curiously, the trend is that, as the number of building blocks in the correct position decreases, their number in the incorrect position increases, indicating a good mix of diversity in the population.

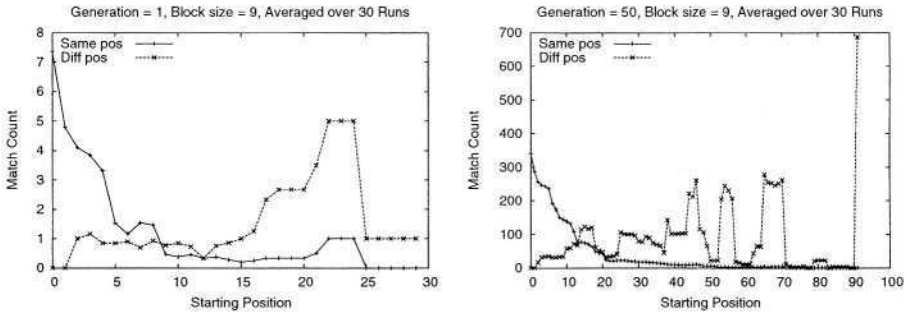


Fig. 4. Count of correctly and incorrectly positioned (overlapping) building blocks of size 9 after the first generation and after 50 generations.

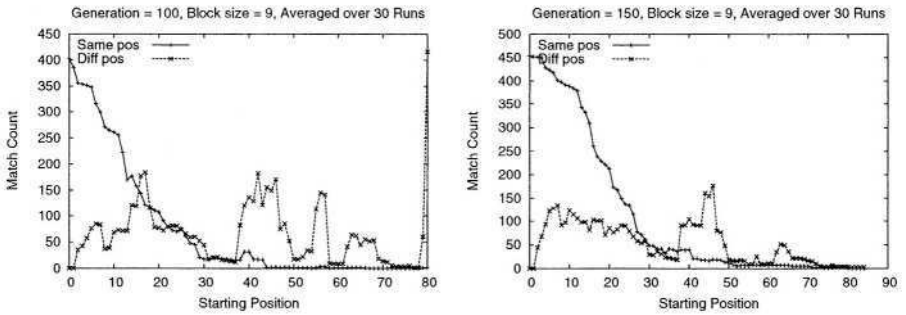


Fig. 5. Count of correctly and incorrectly positioned (overlapping) building blocks of size 9 after the 100 generation and after 150 generations.

By generation 50, also in Fig. 4, we see a very different picture. Notice the difference in the scales and that, on average, around 350 individuals have the same first nine genes. Further, for certain sequences towards the end of the chromosomes, we see relatively high peaks, indicating that many individuals have those sequences, albeit not in the correct position. This seems to support the findings of [17] which postulated the existence of a "stop sequence" in GE. That is, a sequence of genes that can successfully terminate most of the chromosomes in the population. An analogy in GP would be the existence of subtrees that, no matter where they appear in a tree at least don't give a pathological fitness.

This trend continues in Fig. 5, although the lines cross further along, indicating that the population is converging.

4.2 Growth of Rooted Building Blocks

The first batch of experiments here is designed to examine the manner in which the rooted building blocks grow over time. Fig. 6 indicates the average size of rooted building blocks in the population over time, expressed as a proportion of

the *best-of-run* individual for the particular population. It also shows the average size of the common root structure for all individuals in a population.

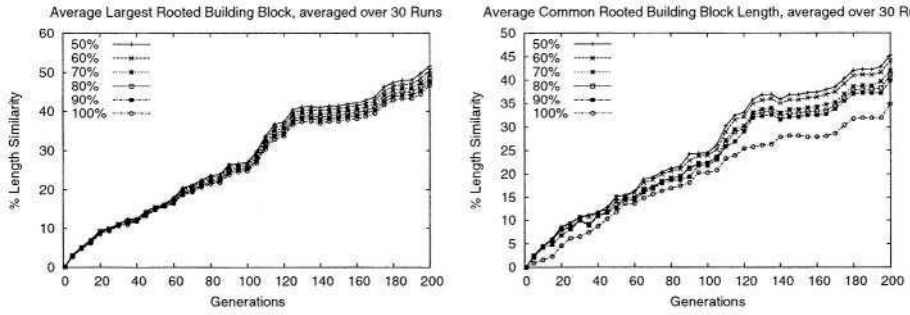


Fig. 6. On the left, the average size of the rooted building blocks, and, on the right, the average size of the common rooted building block.

As it was not clear to what extent a single rooted building block would take over the population, we made several different measures when examining what the average common root node structure was. [9] indicated that around 70% of individuals in GP tended to have a much larger area in common than when the entire population was examined. To reflect this, for this analysis we scored individuals according to how much in common their root structure was with the *best-of-run* individual to find a current *best-match* individual (although these scores were not taken into consideration during the run, this was only done for the retrospective analysis). The measurements that appear in Fig. 6 consider from the top 50% of the population down to the top 100%, the entire population.

Although examining the top 50% only gives a higher measure, one can see from the graph that there is not an enormous difference between it and using other, larger proportions of the population. The biggest drop is when the entire population is considered, but still shows that as early as generation four, *every* individual in the population has the same root node.

Fig. 7 indicates the manner in which newly discovered extended rooted building blocks take over the population. On the right one can see how the size of the largest building block increases over time, while on the left the count of individuals having that building block are shown.

The spiky nature of the counts indicates the constant discovery of new larger blocks, which then propagate throughout the rest of the population. In the earlier stages of the evolution rate of propagation is higher than the latter stages, possibly because the difference in fitness is more marked earlier on in a run. Furthermore, as the size of the root structure becomes larger, there is more space of small errors, and our method of measuring similarity only counts blocks that are *exactly* identical.

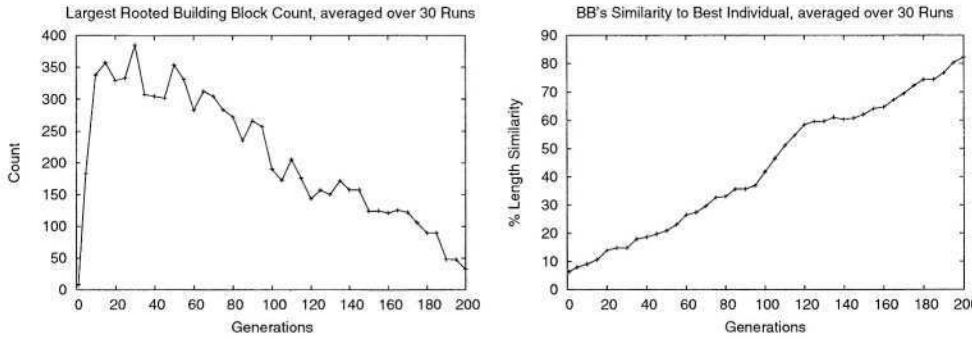


Fig. 7. Left: The count of individuals having the longest rooted building block over time. Right: The average size of the largest rooted building block in the population over time.

5 Discussion and Conclusions

These experiments appear to validate the hypothesis that the evolution is driven by a combination of competitive rooted building blocks and co-operative non-rooted building blocks. Crucially, in Figs. 4 and 5 the count of building blocks in *different positions* never reaches zero, which indicates the presence of the genetic material required for the construction of the building block at that position in the form of a *co-operative* building block, which, at a later stage, may move to the correct position.

The second set of experiments, in Fig. 6, focused on the rooted building blocks and their behavior over time. The linear slope of the curve indicates that, on average, the root structure continually grows towards the *best-of-run* individual. Also in Fig. 6 one can see that the *entire* population shares an increasingly similar root structure as time goes on.

The final set of experiments were designed to measure the convergence rate of the system towards the *best-of-run* individual, as well as the counts of the instances of each *best-match* individual. The counts of the best matching individual provides perhaps the best proof of the competitive nature of the root building blocks. Initially, the counts are very high, but, as the newly discovered root structure gets increasingly longer, the competition intensifies, with fewer and fewer individuals making the discovery each time. However, the undulating nature of the graph indicates that, once a new and longer root structure has been discovered, it very quickly spreads through the population.

In conclusion then, this paper supports a competitive building block hypothesis, and that a single rooted building block grows over time. Whether or not this is a good thing for GP-like systems remains to be seen, as it suggests that if the initial root structure is flawed, possibly through poor diversity or too small a population, then the population becomes trapped. However, to be fore-warned

is to be fore-armed, and future work will look at the implications of this, and ways in which to avoid prematurely converging on suboptimal root structure.

References

1. Peter J. Angeline and Jordan B. Pollack. Competitive environments evolve better solutions for complex tasks. In Stephanie Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93,* pages 264–270, University of Illinois at Urbana-Champaign, 17-21 July 1993. Morgan Kaufmann.
2. David E. Goldberg. Simple genetic algorithms and the minimal, deceptive problem. In L. Davis, editor, *Genetic Algorithms and Simulated Annealing,* pages 74–88. Morgan Kaufmann, 1987.
3. David E. Goldberg. Genetic algorithms and walsh functions: part i, a gentle introduction. *Complex Systems,* 3(2):129–152, 1989.
4. David E. Goldberg. *The Design of Innovation: Lessons from and for Competent Genetic Algorithms.* Kluwer Academic Publishers, 2002.
5. John H. Holland. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, Ann Harbor, 1975.
6. Maarten Keijzer. *Scientific Discovery using Genetic Programming.* PhD thesis, Danish Technical University, Lyngby, Denmark, March 2002.
7. Maarten Keijzer, Conor Ryan, Michael O'Neill, Mike Cattolico, and Vladin Babovic. Ripple crossover in genetic programming. In Julian F. Miller, Marco Tomassini, Pier Luca Lanzi, Conor Ryan, Andrea G. B. Tettamanzi, and William B. Langdon, editors, *Genetic Programming, Proceedings of EuroGP'2001,* volume 2038 of *LNCS,* pages 74–86, Lake Como, Italy, 18-20 April 2001. Springer-Verlag.
8. W. B. Langdon and Riccardo Poli. *Foundations of Genetic Programming.* Springer-Verlag, 2002.
9. Nicholas Freitag McPhee and Nicholas J. Hopper. Analysis of genetic diversity through population history. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference,* volume 2, pages 1112–1120, Orlando, Florida, USA, 13-17 July 1999. Morgan Kaufmann.
10. Melanie Mitchell, Stephanie Forrest, and John H. Holland. The royal road for genetic algorithms: Fitness landscapes and GA performance. In Francisco J. Varela and Paul Bourgine, editors, *Towards a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life, 1991,* pages 245–254, Paris, 11–13 1992. A Bradford book, The MIT Press.
11. Michael O'Neill and Conor Ryan. Crossover in grammatical evolution: A smooth operator? In Riccardo Poli, Wolfgang Banzhaf, William B. Langdon, Julian F. Miller, Peter Nordin, and Terence C. Fogarty, editors, *Genetic Programming, Proceedings of EuroGP'2000,* volume 1802 of *LNCS,* pages 149–162, Edinburgh, 15-16 April 2000. Springer-Verlag.
12. Michael O'Neill and Conor Ryan. *Grammatical Evolution: Evolutionary Automatic Programming in a Arbitrary Language,* volume 4 of *Genetic programming.* Kluwer Academic Publishers, 2003.
13. Michael O'Neill, Conor Ryan, Maarten Keijzer, and Mike Cattolico. Crossover in grammatical evolution. *Genetic Programming and Evolvable Machines,* 4(1):67–93, March 2003.

14. Una-May O'Reilly and Franz Oppacher. The troubling aspects of a building block hypothesis for genetic programming. In L. Darrell Whitley and Michael D. Vose, editors, *Foundations of Genetic Algorithms 3,* pages 73–88, Estes Park, Colorado, USA, 31 July–2 August 1994 1995. Morgan Kaufmann.
15. John O'Sullivan and Conor Ryan. An investigation into the use of different search strategies with grammatical evolution. In James A. Foster, Evelyne Lutton, Julian Miller, Conor Ryan, and Andrea G. B. Tettamanzi, editors, *Genetic Programming, Proceedings of the 5th European Conference, EuroGP 2002,* volume 2278 of *LNCS,* pages 268–277, Kinsale, Ireland, 3-5 April 2002. Springer-Verlag.
16. Conor Ryan, J. J. Collins, and Michael O Neill. Grammatical evolution: Evolving programs for an arbitrary language. In Wolfgang Banzhaf, Riccardo Poli, Marc Schoenauer, and Terence C. Fogarty, editors, *Proceedings of the First European Workshop on Genetic Programming,* volume 1391 of *LNCS,* pages 83–95, Paris, 14-15 April 1998. Springer-Verlag.
17. Conor Ryan, Maarten Keijzer, and Miguel Nicolau. On the avoidance of fruitless wraps in grammatical evolution. In E. Cantú-Paz et al, editor, *Genetic and Evolutionary Computation – GECCO-2003,* volume 2724 of *LNCS,* pages 1752–1763, Chicago, 12-16 July 2003. Springer-Verlag.
18. Conor Ryan and Miguel Nicolau. Doing genetic algorithms the genetic programming way. In Rick L. Riolo and Bill Worzel, editors, *Genetic Programming Theory and Practise,* chapter 12, pages 189–204. Kluwer, 2003.

Dynamic Limits for Bloat Control
Variations on Size and Depth

Sara Silva and Ernesto Costa

Evolutionary and Complex Systems Group
Centre for Informatics and Systems of the University of Coimbra
Polo II – Pinhal de Marrocos, 3030 Coimbra, Portugal
{sara,ernesto}@dei.uc.pt
http://cisuc.del.uc.pt/ecos/

Abstract. We present two important variations on a recently successful bloat control technique, Dynamic Maximum Tree Depth, intended at further improving the results and extending the idea to non tree-based GP. Dynamic Maximum Tree Depth introduces a dynamic limit on the depth of the trees allowed into the population, initially set with a low value but increased whenever needed to accommodate a new best individual that would otherwise break the limit. The first variation to this idea is the Heavy Dynamic Limit that, unlike the original one, may fall again to a lower value after it has been raised, in case the new best individual allows it. The second variation is the Dynamic Size Limit, where size is the number of nodes, instead and regardless of depth. The variations were tested in two problems, Symbolic Regression and Parity, and the results show that the heavy limit performs generally better than the original technique, but the dynamic limit on size fails in the Parity problem. The possible reasons for success and failure are discussed.

1 Introduction

Genetic Programming (GP) solves complex problems by evolving populations of computer programs, using Darwinian evolution and Mendelian genetics as inspiration. Bloat is an excess of code growth caused by the genetic operators in search of better solutions, without the corresponding improvement in fitness, and it may be caused by both introns and exons [1,14]. Several explanations for bloat have been proposed (reviews in [2,13,16,17]), and many different bloat control techniques have been tried (reviews in [8,10,15]). Most of them are based on parsimony pressure, some of the latest using modified tournaments [8,9].

Recently, a new bloat control technique, Dynamic Maximum Tree Depth, was shown to be very successful in two simple problems [12], particularly when coupled with Lexicographic Parsimony Pressure [8]. Especially suited for tree-based GP, the dynamic depth technique is based on a dynamic limit on the depth of the trees allowed in the population, initially set with a low value and only raised when needed to accommodate a new best individual that would otherwise break the limit. Once this limit is raised, it will not be lowered again.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 666–677, 2004.

This paper elaborates on the dynamic depth idea, introducing two important variations aiming at further improving the results and extending the concept to non tree-based GP. The first variation is the use of a *heavy* dynamic limit that, unlike the original dynamic limit, may fall again to a lower value after it has been raised, in case the new best individual allows it. The second is the use of a dynamic limit on the tree size (number of *nodes*), instead and regardless of depth. The heavy factor is used with both depth and size limits, alone and in combination with Lexicographic Parsimony Pressure, and the results are compared with the published work [12].

2 Previous Work

Dynamic Maximum Tree Depth [12] introduces a dynamic limit to the maximum depth of the individuals allowed in the population. It is similar to the traditional static limit [4], but it does not replace it – both dynamic and strict limits are used in conjunction. The dynamic limit should be initially set with a low value at least as high as the maximum depth of the initial random trees. Any new individual who breaks this limit is rejected and replaced by one of its parents instead, as it does when it breaks the static limit, unless it is the best individual found so far. In this case, the dynamic limit increases to match the depth of the new best and allow it into the population. The dynamic limit is never decreased, and it never surpasses the static limit in any circumstance.

Lexicographic Parsimony Pressure is based on a modified tournament that always selects smaller trees when their fitness is the same. Individuals are first compared in terms of fitness, and the best one wins, regardless of its size. But if their fitness is the same, they are then compared in terms of size, defined as the number of tree nodes, and the smaller one wins.

Lexicographic Parsimony Pressure performed well in three problems where it is usual to find two different individuals with the same fitness (Parity, Multiplexer, Artificial Ant), but failed in a problem where two different individuals seldom have the same fitness (Symbolic Regression), and the bucketing variation was not able to produce much better results either [8]. Dynamic Maximum Tree Depth performed equally well in two problems of different nature (Parity, Symbolic Regression), and its combination with Lexicographic Parsimony Pressure was beneficial in most cases [12].

3 Variations on Size and Depth

We now introduce two important variations on the original work on Dynamic Maximum Tree Depth, explaining the reasons for the options taken.

3.1 Heavy Dynamic Limit

In the previous section we have learned that the results of Dynamic Maximum Tree Depth were not harmed by the use of the most restrictive initial dynamic

limit 6, instead of 9. This suggests that the technique is able to withstand a high amount of parsimony pressure without losing performance. We have also learned that the dynamic limit is never allowed to decrease during the run, even if the new best individual is less deep than the current dynamic limit (something that we have often observed), and there seems to be no good reason for this. Lowering the dynamic limit whenever the new best individual allows it is our first variation on the original technique - we have called it *heavy dynamic limit.*

As expected, whenever the limit falls back to a lower value, some individuals already in the population immediately break the new limit, becoming 'illegals'. There was a vast range of options to deal with them, the more drastic being their immediate removal from the population, possibly replacing them by new random individuals. However, when considering that these new 'illegals' could be the ones who managed to produce the new best individual, this did not seem like a very good idea. We have decided to adopt a much softer option: the 'illegals' are allowed to remain in the population as if they were not breaking the limit, but when engaging in crossover their children cannot be larger (in depth or size, depending on the case - see Sect. 3.2) than the largest parent. This naturally and slowly places the population within limits again.

3.2 Dynamic Size Limit

Dynamic Maximum Tree Depth is only suited for tree-based GP, but it is known that bloat is prone to affect any other variable-length representation [6]. The extension of the dynamic limit idea to other domains must begin with the removal of the concept of depth, replacing it with the concept of size. In tree-based GP, this means ignoring tree depth and looking only at the number of tree nodes. Our second variation on the original work is the replacement of the dynamic depth limit with what we have called the *dynamic size limit,* where size is defined as the number of nodes. Because a static inviolable limit is useful to avoid the possible exhaustion of computer resources, the dynamic size variation also uses a static limit on the number of nodes, equivalent to the static depth limit always used with the original technique (see Sect. 2).

When using the dynamic size limit, it makes no sense to keep using depth as a restriction on tree initialization. So we have created a modified version of the Ramped Half-and-Half initialization method [4], where an equal number of individuals are initialized with sizes ranging between 2 and the initial value of the dynamic size limit. For each size, an equal number of individuals are initialized with the Grow method and with the Full method [4], that we have also modified to fit the size constraints only. In the modified Grow method, the individual grows by addition of random nodes (internal or terminal) without exceeding the maximum size specified; the modified Full method chooses only internal nodes until the size is close to the specified, and only then chooses terminals. Unlike the original Full version, it may not be able to create individuals with the exact size specified, but only close (never exceeding).

4 Experiments

To test the efficacy of the variations introduced in the original idea, we have used the same simple problems studied in [12]: Symbolic Regression of the quartic polynomial ($x^4 + x^3 + x^2 + x$, with 21 equidistant points in the interval −1 to +1), and Even-3 Parity. We have performed the same battery of tests using 10 different approaches, the first four techniques being previous work (see Sect. 2):

(1) K → static depth limit [4];
(2) L → lexicographic parsimony pressure [8];
(3) D → dynamic depth limit [12];
(4) DL → D coupled with L [12];
(5) HD → heavy dynamic depth limit;
(6) HDL → HD coupled with L;
(7) N → dynamic size limit;
(8) NL → N coupled with L;
(9) HN → heavy dynamic size limit;
(10) HNL → HN coupled with L.

Table 1 summarizes the combinations of dynamic limit and tournament settings that lead to each technique. The first two techniques (K, L) only use a static limit on tree depth; the following four 'D' techniques (D, DL, HD, HDL) use dynamic limit on tree depth, initially set at 6, the value that yielded better results [12]; the last four 'N' techniques (N, NL, HN, HNL) use dynamic limit on tree size (number of nodes). Regardless of what kind of dynamic limit they use, all techniques use a static limit on depth (17) or number of nodes (512), whichever is the case. 512 seems much less than the allowed number of nodes on a 17-depth tree, but in fact it almost doubles the maximum number of nodes actually found in 17-depth trees of the experiments performed with dynamic depth. The initial values for the dynamic size limit are 21 for Symbolic Regression and 40 for Even-3 Parity. These were the values that produced distributions of tree sizes more similar to the ones obtained with dynamic depth limit 6, as determined by the Kolmogorov-Smirnov goodness-of-fit criterium. All the 'H' techniques use the heavy limit described in Sect. 3.1. We have decided to combine it with both depth and size limits. All the 'L' techniques use the modified tournament described in Sect. 2 (with no bucketing, for the sake of simplicity).

A total of 30 runs were performed with each technique for each problem. The parameters adopted for the experiments were essentially the same as in [4,8,12] to facilitate the comparison between the techniques. All the runs used populations of 500 individuals evolved during 50 generations, even when an optimal solution was found earlier. Tree initialization was performed with the Ramped Half-and-Half method [4], modified in the techniques using size limit (see Sect. 3.2). Reproduction was set at 10%, no mutation was used, and the crossover points were totally random, independently of being internal or terminal nodes. Tournament size was always 2. The (protected) function sets for the Regression and Parity problems were {+, –, ×, ÷, *sin, cos, log, exp*} and {*nand, nor, and, or*}, respectively, with no random constants used.

Table 1. Dynamic limit and tournament settings for each technique used

Technique	Dynamic limit	Heavy factor	Tournament
(1) K	-	-	standard
(2) L	-	-	modified
(3) D	depth	no	standard
(4) DL	depth	no	modified
(5) HD	depth	yes	stardard
(6) HDL	depth	yes	modified
(7) N	size	no	standard
(8) NL	size	no	modified
(9) HN	size	yes	stardard
(10) HNL	size	yes	modified

All the experiments were performed using the GPLAB toolbox [11]. Statistical significance of the null hypothesis of no difference was determined with Kruskal-Wallis non-parametric ANOVAs at $p = 0.01$.

5 Results

The results of the experiments are presented as boxplots and evolution curves concerning: the mean size of the trees (Fig. 1), where size is the number of nodes; the mean percentage of introns in the trees (Fig. 2), calculated like in [12]; and the population diversity (Fig. 3), defined as the percentage of individuals in the population that account for the total number of (genotypically) different individuals (based on variety in [5]). The p-values concerning the best (lowest) fitness of run are also presented, in Table 2. The boxplots are based on the mean values, and each technique is represented by a box and pair of whiskers. Each box has lines at the lower quartile, median, and upper quartile values, and the whiskers mark the furthest value within 1.5 of the quartile ranges. Outliers are represented by + and × marks the mean. A dot on the bottom of the lower whisker indicates there are no outliers. The evolution curves represent each generation separately (averaged over all runs), one line per technique. Throughout the text, we use the acronyms introduced in Table 1 to designate the different techniques.

5.1 Mean Size of Trees

Figure 1 shows the results concerning the mean size of trees. In both problems, all the techniques using dynamic limits achieved significantly lower tree sizes than K and L, confirming and strengthening the results reported in [12]. In the Regression problem, the heavy depth techniques (HD, HDL) caused a significant decrease in tree sizes when compared to the original depth techniques (D, DL) and to any of the dynamic size techniques (the last four). The differences

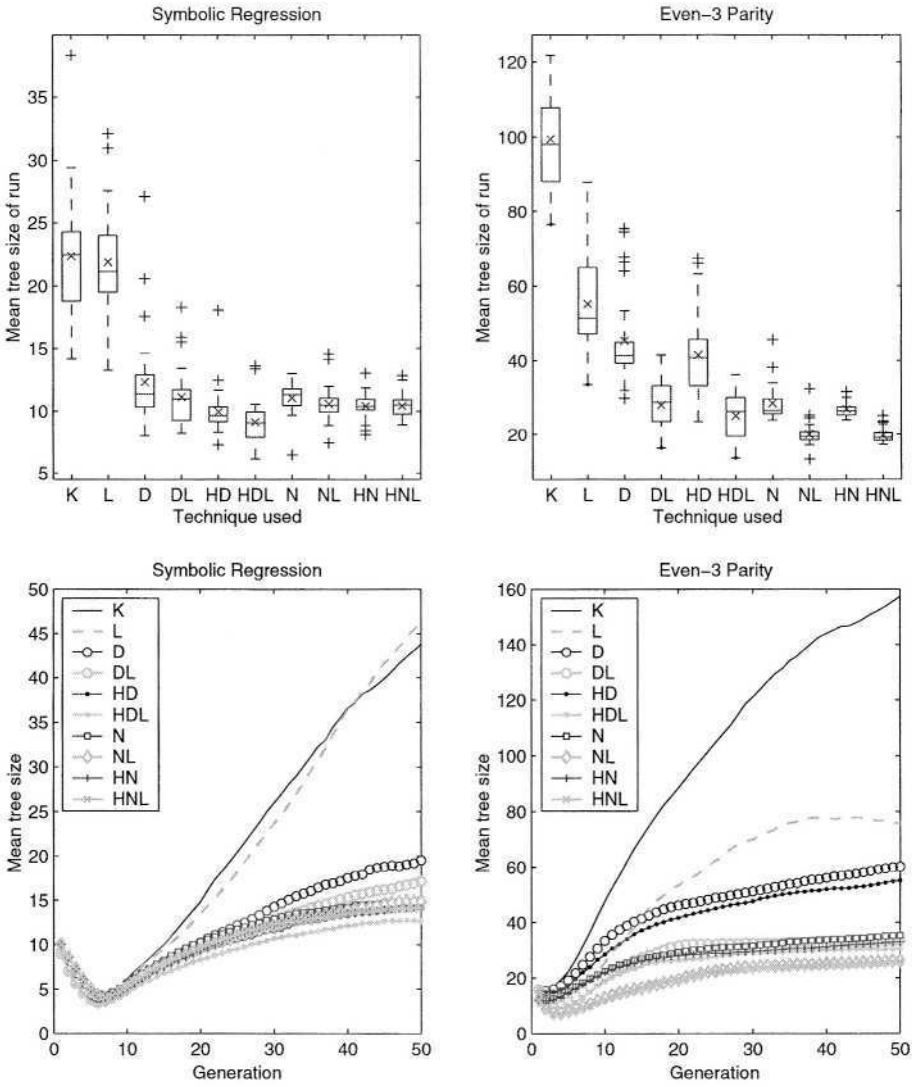


Fig. 1. Boxplots and evolution curves of mean tree size

introduced by the L tournament are not considered significant in any case. In general, the techniques based on size were not significantly better than the ones based on depth. Nevertheless, the evolution curves show that, by the end of the run, the lower tree sizes are achieved by both the heavy depth techniques and by all the size techniques.

In the Parity problem, the heavy factor never caused any significant difference. On the other hand, the introduction of the L tournament caused a signif-

icant decrease in tree sizes, in all cases. When comparing the techniques based on size with the ones based on depth, only the size techniques coupled with the L tournament (NL, HNL) were always significantly better than any of the depth techniques. The evolution curves show no surprises. The fact that the difference in mean tree size between the best and the worst techniques surpasses 130 nodes is noteworthy.

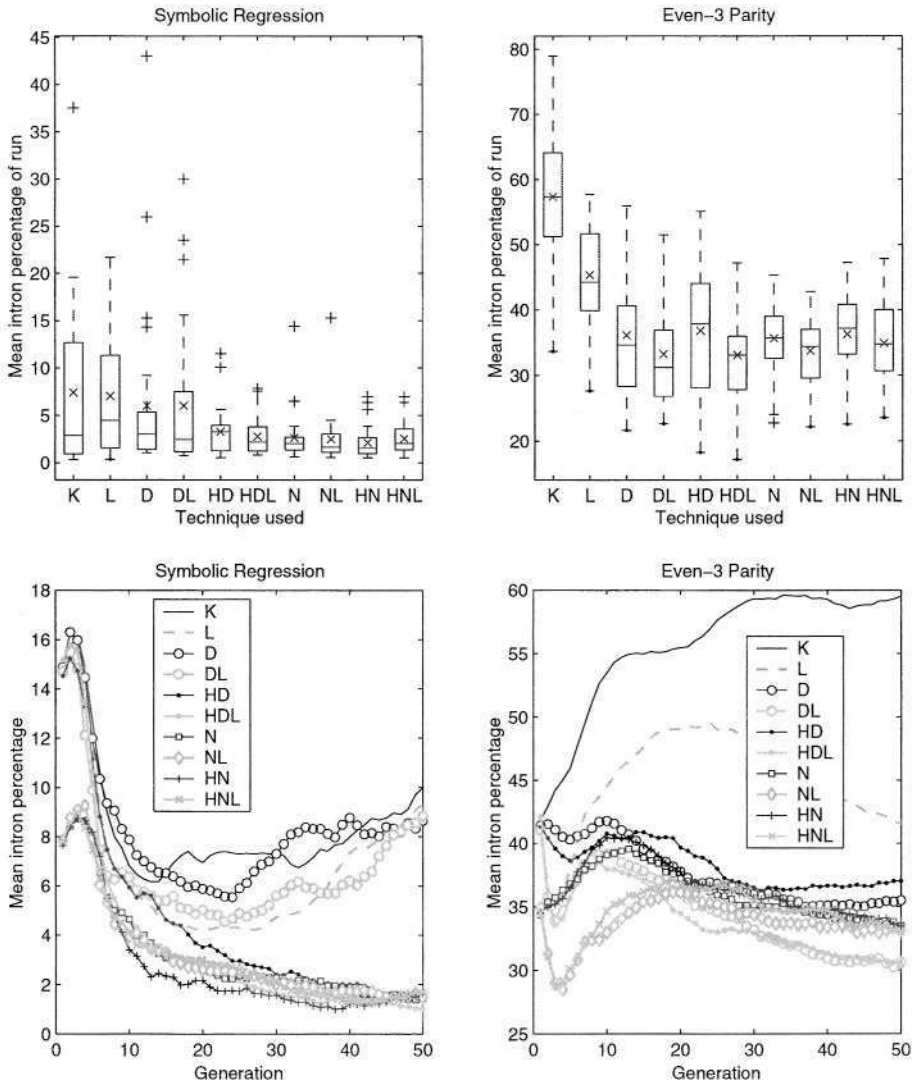


Fig. 2. Boxplots and evolution curves of mean intron percentage

5.2 Mean Percentage of Introns

Figure 2 shows the results concerning the mean percentage of introns. The tree initialization procedure used with the dynamic size techniques produces comparatively lower percentages of introns in both problems, although tree sizes are very similar (Fig. 1). In the Regression problem we realize that, as in [12], code growth is not caused by introns. The differences between the several techniques using dynamic limits are not considered significant, but the evolution curves show that, by the end of the run, only the new variations (heavy factor and dynamic size) continue lowering the percentage of introns, unlike in [12]. In the Parity problem, there are also no significant differences between any of the techniques using dynamic limits, and no marked tendency is apparent in the evolution curves either.

5.3 Population Diversity

Figure 3 shows the results concerning the population diversity. In the Regression problem, the techniques based on size are all capable of sustaining significantly higher diversity than the ones based on depth, but still perform worse than K and L. The influence of the heavy factor and the L tournament is considered significant in only half of the cases. The evolution curves show a marked decrease in diversity when the run starts (like in [12]), and by the end of the run the heavy depth techniques (HD, HDL) distinguish themselves by the lower diversity, particularly HDL that ends with a value almost as low as 50%.

In the Parity problem, once again the techniques based on size can sustain significantly higher diversity than the ones based on depth. The L tournament is responsible for significant differences in population diversity, but only because of a sharp initial decrease observed in the evolution curves, similar to what happened in the Regression problem. This phenomenon was also apparent in [12], where a plausible explanation has been proposed.

5.4 Best Fitness

Table 2 shows the p-values concerning the best fitness of run. Although this table contains more detailed information than the plots would, these values should not be interpreted as anything more than mere indicators of performance and suitability of the new techniques under study.

In the Regression problem, it is worth noting that both DL and HD yield significantly *better* results than L or most of the dynamic nodes techniques. The heavy dynamic depth did not decrease the ability to find individuals with good fitness. Although the dynamic limit on nodes only fails when compared to the winners DL and HD, we also notice substantially lower p-values when comparing to K and D.

In the Parity problem, once again the heavy dynamic depth performed as well as any of the previously published techniques (the first four). On the contrary, the dynamic limit on nodes performed significantly *worse* than the rest, except N (although with substantially lower p-values).

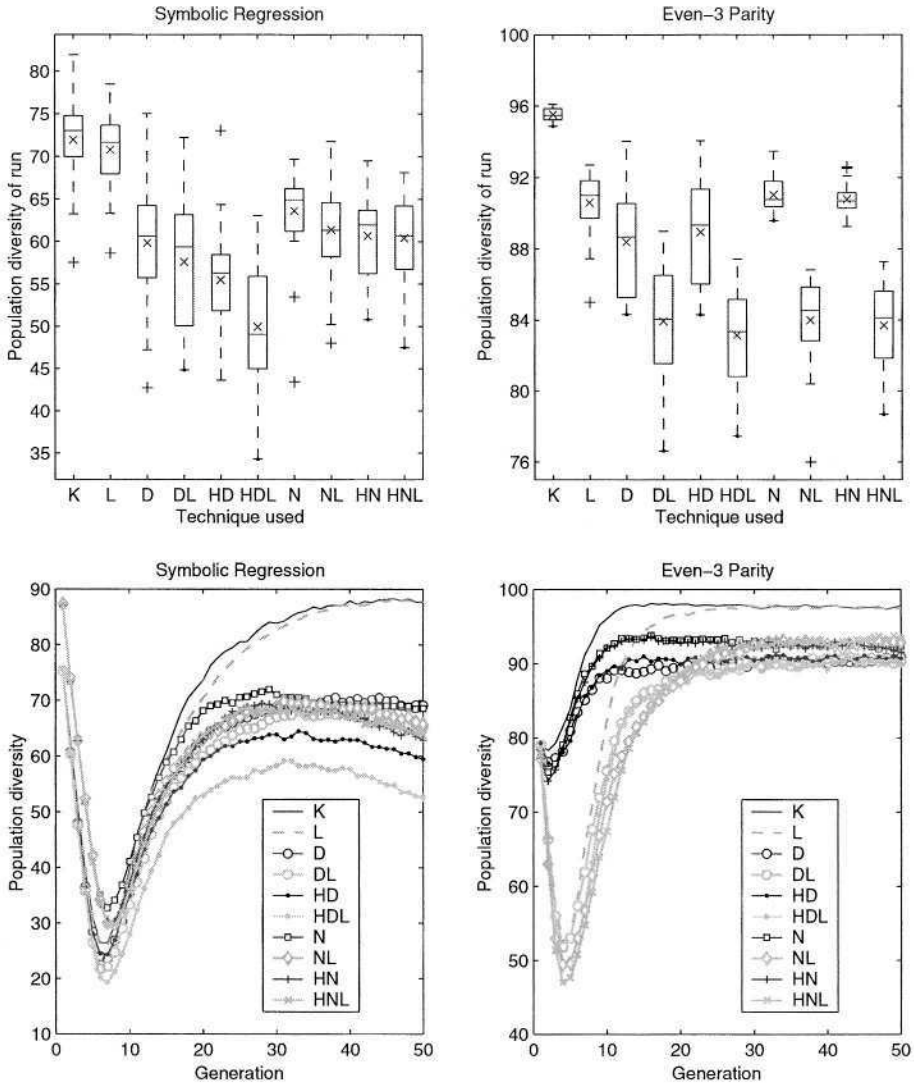


Fig. 3. Boxplots and evolution curves of population diversity

6 Discussion and Conclusions

The introduction of the heavy factor into the dynamic limit adds parsimony pressure during the run. Even without taking drastic measures towards the individuals that suddenly break the new limit, the results show that the heavy limit is capable of achieving even lower mean tree sizes than the original limit, particularly when applied to depth in the Regression problem. This decrease in

Table 2. p-values concerning the best fitness of run, using non-parametric ANOVAs. Statistical significance is considered where $p<0.01$ (in bold). See Sect. 5.4 for details

Even-3 Parity

	K	L	D	DL	HD	HDL	N	NL	HN	HNL	
K		1.000	0.393	0.720	0.690	0.492	0.120	**0.000**	**0.001**	**0.000**	K
L	0.144		0.393	0.720	0.690	0.492	0.120	**0.000**	**0.001**	**0.000**	L
D	0.719	0.057		0.232	0.643	0.132	0.021	**0.000**	**0.000**	**0.000**	D
DL	0.057	**0.000**	0.063		0.451	0.741	0.226	**0.000**	**0.003**	**0.000**	DL
HD	0.271	**0.001**	0.332	0.165		0.282	0.055	**0.000**	**0.000**	**0.000**	HD
HDL	0.599	0.306	0.443	0.014	0.110		0.375	**0.000**	**0.008**	**0.000**	HDL
N	0.126	0.682	0.089	**0.002**	0.011	0.369		**0.002**	0.069	**0.005**	N
NL	0.033	0.467	0.012	**0.000**	**0.000**	0.101	0.669		0.188	0.783	NL
HN	0.084	0.700	0.021	**0.000**	**0.001**	0.216	0.844	0.752		0.296	HN
HNL	0.052	0.431	0.031	**0.001**	**0.002**	0.179	0.935	0.976	0.724		HNL
	K	L	D	DL	HD	HDL	N	NL	HN	HNL	

Symbolic Regression

mean tree size is, however, accompanied by a proportional decrease in population diversity, although this makes no difference on the ability to converge to the optimal solution. The concerns regarding how depth limitations may adversely affect GP performance [3,7] do not seem to apply here.

On the other hand, the dynamic size limit did not perform so well. In spite of achieving similarly low mean tree sizes, and without losing so much diversity, it did not perform better than the best depth techniques on the Regression problem. On the Parity problem it clearly failed, with the last three techniques achieving significantly worse results, as if the amount of pressure that the problem can withstand was finally surpassed.

It is arguable whether the dynamic limit on size instead of depth increases the amount of pressure. It is clear that, when dealing with depth, there is usually the opportunity to add more nodes without breaking the limit, because trees are always far from full (see Fig. 4). On the other hand, looking at size regardless of depth removes a very important restriction from the search: the shape of the trees. It is true that when using the size limit instead of the depth limit, trees are comparatively less full. Could this account for the higher diversity achieved? A comparison with Fig. 3 reveals that mean tree fill rate seems to be in fact inversely related to population diversity – even the sudden loss of diversity observed in the beginning of the run is accompanied by a small increase in fill rate. So it may not be the case that a higher number of parent clones, resulting from their offspring being rejected, is the sole reason for diversity loss. The relationship between the two is not even clear (plots not shown), although this is still work in progress. Quite surprisingly, diversity loss (or higher tree fill rate) does not seem to jeopardize performance. The general poor fitness results achieved by the dynamic size techniques have a different cause, and we speculate that it

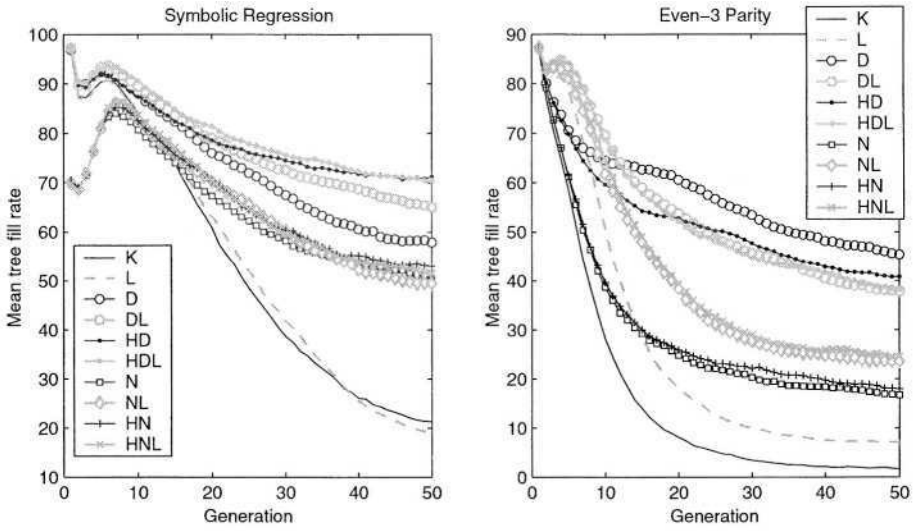


Fig. 4. Evolution curves of mean tree fill rate. Tree fill rate is the percentage of tree nodes in relation to the expected number of nodes of a full tree, given its depth and the function and terminal sets being used

might be related to the difficulty in slightly altering the number of nodes without conflicting with the current limit. This small window of freedom, present in all dynamic depth techniques (see previous paragraph), may be the crucial factor that the dynamic size lacks. If this supposition is true, a simple modification to the dynamic size techniques may easily solve the problem, for example by allowing individuals to always grow a little beyond the limit, regardless of their fitness.

7 Future Work

The whole idea of using dynamic limits should be tested in different, harder problems, and different sets of parameters should also be used to ensure the robustness of the technique. Considering the high applicability that the dynamic size limit may have in other domains, some effort should be made in order to fully understand and possibly eliminate the reasons behind its apparent failure, by introducing the elements responsible for the excellent results achieved by the best dynamic depth techniques. Some insight may be obtained by further studying how the number of parent clones relates to population diversity (where a different, possibly phenotypical, diversity measure may be more informative), and how do tree shape restrictions affect this relationship.

Acknowledgements. We acknowledge grants SFRH/BD/14167/2003 and POCTI/1999/BSE/34794 from Fundação para a Ciência e a Tecnologia, Portugal. We would also like to thank the members of the ECOS – Evolutionary and Complex Systems Group (Universidade de Coimbra) and Pedro J.N. Silva (Universidade de Lisboa) for the valuable suggestions provided throughout this research.

References

1. Banzhaf, W., Nordin, P., Keller, R.E., Francone, F.D.: Genetic programming – an introduction, San Francisco, CA. Morgan Kaufmann (1998)
2. Brameier, M., Bankhaf, W.: Neutral variations cause bloat in linear GP. In C. Ryan *et al.,* editors, Proceedings of EuroGP-2003, Berlin. Springer (2003) 286–296
3. Gathercole, C., Ross, P.: An adverse interaction between crossover and restricted tree depth in genetic programming. In J.R. Koza *et al.,* editors, Proceedings of GP'96, Cambridge, MA. MIT Press (1996) 291–296
4. Koza, J.R.: Genetic programming – on the programming of computers by means of natural selection, Cambridge, MA. MIT Press (1992)
5. Langdon, W.B.: Genetic Programming + Data Structures = Automatic Programming!, Boston, MA. Kluwer (1998)
6. Langdon, W.B.: The evolution of size in variable length representations. In D. Fogel, editor, Proceedings of ICEC'98. IEEE Press (1998) 633-638
7. Langdon, W.B., Poli, R.: An analysis of the MAX problem in genetic programming. In J.R. Koza *et al.,* editors, Proceedings of GP'97, San Francisco, CA. Morgan Kaufman (1997) 222–230
8. Luke, S., Panait, L.: Lexicographic parsimony pressure. In W.B. Langdon *et al.,* editors, Proceedings of GECCO-2002, San Francisco, CA. Morgan Kaufmann (2002) 829–836
9. Luke, S., Panait, L.: Fighting bloat with nonparametric parsimony pressure. In J.J. Merelo Guervós *et al.* , editors, Proceedings of PPSN-2002, Berlin. Springer Verlag (2002) 411–421
10. Poli, R.: A simple but theoretically-motivated method to control bloat in genetic programming. In C. Ryan *et al.,* editors, Proceedings of EuroGP-2003, Berlin. Springer (2003) 204–217
11. Silva, S.: GPLAB – a genetic programming toolbox for MATLAB. (2004) http://gplab.sourceforge.net
12. Silva, S., Almeida, J.: Dynamic maximum tree depth – a simple technique for avoiding bloat in tree-based GP. In E. Cantú-Paz *et al.,* editors, Proceedings of GECCO-2003, Berlin. Springer Verlag (2003) 1776–1787
13. Soule, T.: Code growth in genetic programming. PhD thesis, University of Idaho (1998)
14. Soule, T.: Exons and code growth in genetic programming. In J.A. Foster *et al.,* editors, Proceedings of EuroGP-2002, Berlin. Springer (2002) 142–151
15. Soule, T., Foster, J.A.: Effects of code growth and parsimony pressure on populations in genetic programming. Evolutionary Computation, 6(4) (1999) 293–309
16. Soule, T., Heckendorn, R.B.: An analysis of the causes of code growth in genetic programming. Genetic Programming and Evolvable Machines, 3 (2002) 283–309
17. Streeter, M.J.: The root causes of code growth in genetic programming. In C. Ryan *et al.,* editors, Proceedings of EuroGP-2003, Berlin. Springer (2003) 449–458

On Naïve Crossover Biases with Reproduction for Simple Solutions to Classification Problems

M. David Terrio and Malcolm I. Heywood

Dalhousie University, Faculty of Computer Science
6040 University Avenue, Halifax, NS. B3H 1W5 Canada
{mterrio, mheywood}@cs.dal.ca

Abstract. A series of simple biases to the selection of crossover points in tree-structured genetic programming are investigated with respect to the provision of parsimonious solutions. Such a set of biases has a minimal computational overhead as they are based on information already used to estimate the fitness of individuals. Reductions to code bloat are demonstrated for the real world classification problems investigated. Moreover, bloated solutions provided by a uniform crossover operator often appear to defeat the application of MAPLE™ simplification heuristics.

1 Introduction

The variable length representation employed by tree-structured Genetic Programming (GP) [1], whilst satisfying the goal of providing a machine-learning paradigm with a minimum number of *a priori* constraints, is also known to lead to approximately square law increases in code length [2], or what has typically become known as code bloat [3-7]. The ideal would therefore be to evolve fit – as measured by the application performance objective – yet parsimonious solutions. By doing so, the computational requirements necessary to evolve solutions would be significantly reduced – fitness evaluation is the most computationally expensive process, where this is proportional to the size of the data set and chromosome length of a candidate solution – whilst increasing the acceptance rate of GP solutions in the application domain. Moreover, the causes or biases behind code bloat are known to vary depending on the GP structure [8]. Here, we concentrate on tree structured as opposed to linearly structured GP.

In this work our motivation is to investigate the applicability of a series of naïve biases introduced to the crossover operator using fitness information freely available during evaluation of individuals. The basic objective is to direct the identification of crossover points such that parsimonious solutions are identified with minimal impact on solution performance. In addition, crossover is only allowed to produce one child. The second is reproduced, the motivation being to let mutation provide further investigation of the current shape [2]. In relation to previous works, Iba and de Garis considered the case of collecting information during fitness evaluation to *deterministi-*

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 678–689, 2004.

cally select sub-trees for application of mutation and crossover [9]. However, the principle motivation appears to have been improvements to the fitness of an individual and not to encourage parsimony in the solutions found.

2 Methodology and Approach

The initial motivation for this work was to provide transparent solutions using tree structured individuals in medical applications. Such a requirement is particularly important in this area due to the need to not just solve the problem, but also to win the confidence of the patient and medical personnel [10]. In practice, however, this might be seen as a general requirement for all solutions produced by machine learning systems. As will become apparent from Section 3, it is not in general possible to apply off-line simplification of GP solutions and expect a succinct solution.

In order to mitigate this effect we introduce two properties. Firstly, a degree of determinism is introduced into the crossover operator such that both stochastic selection and 'directed' selection of crossover points is provided. Figure 1 summarizes this process in terms of a generic GP algorithm with steady state tournament selection. Secondly, only one of the children is a result of the crossover operation, the other is reproduced, Figure 2.

The 'uniform' sub-tree crossover operator used here selects a branch of the tree representing an individual using a uniform probability density function. In providing suitable definitions for the selection of crossover operators, we limit ourselves to investigating the applicability of the following naïve crossover definitions,

Fitness Directed Crossover: This crossover operator selects the node in the individual with highest fitness. Node fitness is the error between the target value and node value (including any wrapper). The node value naturally includes the contribution of any attached sub-tree;

Fitness Difference Directed Crossover: Directed crossover now selects the node with greatest change in fitness as evaluation progresses from terminal to root node. This may or may not help protect the individual from repeatedly selecting the branches near the root node for crossover;

Roulette-Fitness Directed Crossover: In this case each node of an individual is given space in a roulette wheel in proportion to node fitness. In this way it is possible to provide an additional path for the stochastic selection of nodes.

With respect to single child crossover with reproduction of one parent, our principle motivation is to let the reproduced individual be modified by mutation, so emphasizing the further investigation of the current shape [2]. Note that mutation is applied in addition to crossover in the algorithm of Figure 1, and at a relatively high probability of 50%, Table 1. The test for producing a single child is summarized by Figure 2, and affects both the directed and undirected process for sub-tree crossover.

```
1. Initialize Population of size M.
2. Uniform random selection of N individuals (N<<M).
3. Evaluate fitness of the N individuals.
4. Rank the N individuals in descending order of fitness.
5. IF one of the N individuals satisfies the stop criteria, END.
6. Overwrite worst N/2 individuals from tournament with best N/2.
7. Call these the children.
8. For each N/4 pair of child individuals,
9. IF (apply crossover == TRUE)
10. THEN IF (apply directed crossover == TRUE)
        THEN (apply directed crossover)
        ELSE (apply uniform crossover)
11. For each of the N/2 child individuals
12. IF (apply mutation == TRUE)
13.       THEN (apply mutation operator)
14. Replace worst N/4 members selected from population with children of N/4 best.
15. Return to point (2).
```

Fig. 1. Basic algorithm for Genetic Program with Steady-State Tournament selection and directed crossover.

```
identify crossover points.
IF (number of nodes[child#1] > K)
THEN (use child#2 and parent#1)
ELSE (use child #1 and parent#2)
```

Fig. 2. Single child crossover with reproduction.

In each case, the directed form of crossover is applied in conjunction with the uniform selection of crossover points, Figure 1, point 10. That is to say, by adjusting the ratio governing the application of uniform verses directed crossover, it is possible to empirically assess the sensitivity of the approach to different problems. This issue in particular will be investigated in Section 3. Should the directed crossover definitions prove appropriate in practice, the principle benefit of such operators is that they do not require any additional information than that calculated during the course of fitness evaluation.

Mutation may take one of two forms: single-point or multi-point (applied with equal likelihood should the test for mutation prove true, Figure 1, point 12). A single-point mutation selects a node with uniform probability and replaces it with an alternative operator with the same arity. The multi-point mutation operator selects a node with uniform probability and recursively applies the single-point operator over the sub-tree. Neither mutation operator therefore changes the size or shape of the parent.

3 Evaluation

Two benchmark problems are used for the purpose of evaluating the forms of naïve directed crossover: Breast Cancer and Liver Disease Classification [11], where the latter is known to be particularly difficult. Table 1 summarizes the problem and GP parameters used. In all the experiments the parameter K from Figure 2 remains at 200, where no particular significance appears to be attached to different values for 'K' i.e. the parents are chosen uniformly. The data for the classification problems are separated into training and test data, where test data are only used to evaluate performance once the termination criterion is met.

As indicated in Section 2 one of the purposes of the following study is to identify the significance of applying different degrees of directed crossover. To this end we conduct 50 trials for each of the following crossover conditions,

- Uniform selection of crossover points – represents the performance baseline against which crossover points are selected with uniform probability. Hereafter this is referred to as 'Uniform';
- Fitness Directed (FD) crossover – applied at a ratio to the uniform selection of crossover points: 25%, 50%, 75% and 100%; Figure 1, point 10;
- Fitness Difference Directed (FDD) crossover – applied at a ratio to the uniform selection of crossover points: 25%, 50%, 75% and 100%; Figure 1, point 10;
- Roulette-Fitness Directed (R-FD) crossover – applied at a ratio to the uniform selection of crossover points: 25%, 50%, 75% and 100%; Figure 1, point 10.

Table 1. Tableau of GP parameters.

Objective	Find a program that classifies the Breast Cancer (Liver Disease) Classification problem.
Terminal Set	$x_0, \ldots, x_8$ ($x_0, \ldots, x_5$)
Functional Set	+, –, *, %, sin, cos, sqrt
Fitness Cases	524 training, 175 test (259 training, 86 test)
Fitness (and Hits)	Number of correct classifications.
Selection	Steady State Tournament of size 4.
Wrapper	IF GP output > 0.0 and data label is '1' THEN correct classification ELSE IF output ≤ 0.0 and data label is '0' THEN correct classification
Population size	500
Initial Population	Created using "ramped half and half" with depths between 2 and 6.
Parameters:	90% crossover, 50% mutation.
Termination:	15,000 tournaments.
Experiments:	50 independent trials.

Thus, Fitness Directed crossover applied at a ratio of 25% results in the application of uniform crossover 75% of the time, when the test for crossover returns true. Naturally, the case of a 100% ratio implies that only the directed crossover operator is applied.

Performance is expressed in terms of training and test classification accuracy of the best individual in the last tournament, resulting in each of the 50 trials contributing to the classification accuracy. By way of comparison the c5.0 Induction System [12] produced best-case classification errors on test data of 4.6% in the Beast Cancer Classification problem and 34.9% in the Liver Disease Classification problem. From Tables 2 and 3 it is apparent that the medians of all GP solutions are better than those from c5.0 for the two problems. Moreover, c5.0 produced solutions with 8 rules for Breast Cancer and 27 rules for Liver Disease. The median of all biased GP solutions for Liver Disease using FD and FDD were significantly simpler whilst retaining lower median error rates. Under Breast c5.0 solution was simpler, albeit at the expense of classification accuracy.

Table 2. Breast Cancer Classification Performance.

Crossover Type		Median Test Error	Median depth per solution	Median # nodes per solution	Average # nodes per individual
Uniform (baseline)		1.41	7	22	18.23
FD	0.25	***1.15***	7	20	16.35
	0.5	***1.15***	6.5	21	19.59
	0.75	1.73	6	16	15.76
	1.0	2.3	***4***	***10***	9.654
R-FD	0.25	***1.15***	8	21.5	18.68
	0.5	1.73	7	20.5	18.3
	0.75	1.73	7	24	17.76
	1.0	***1.15***	7	20.5	16.29
FDD	0.25	***1.15***	7	20	18.53
	0.5	***1.15***	7	21	18.19
	0.75	***1.15***	7	24	20.97
	1.0	1.44	5	12	11.97

3.1 Breast Cancer Classification Problem

Table 2 summarizes performance of uniform sub-tree crossover and the three forms of naïve directed crossover on the Breast Cancer Classification problem. It is apparent that the application of fitness directed crossover, irrespective of the form, results in

decreases to code bloat while classification accuracy remains within one percent of the (median) baseline established by uniform sub-tree crossover. It is also apparent that Fitness Directed (FD) and Fitness Difference Directed (FDD) crossover provide the greatest levels of bloat reduction. Roulette-Fitness Directed (R-FD) crossover appeared to be the least sensitive to specific ratios of directed and uniform sub-tree crossover, but also never improved on the nodes per solution provided by uniform crossover.

3.2 Liver Disease Classification

Table 3 summarizes performance of uniform sub-tree crossover and the three forms of naïve directed crossover on the Liver Disease Classification problem.

The pattern of performance remains consistent with that experienced for the Breast Cancer problem. Fitness Directed and Fitness Difference Directed crossover still provide the highest levels of bloat reduction, with increasing cost to classification accuracy as the ratio of directed crossover increases. Roulette-Fitness Directed crossover again provides the most competitive classification performance (with respect to the sub-tree crossover baseline), but does not appear to provide a particular trend in terms of code bloat reduction with increasing ratios of directed crossover (clarified further in next sub-section).

Table 3. Liver Disease Classification Performance.

Crossover Type		Median Test Error	Median depth per solution	Median # nodes per solution	Average # nodes per individual
Uniform (baseline)		33.33	8	34.5	26.77
FD	0.25	33.33	6.5	19.5	18.89
	0.5	33.33	6	21	18.84
	0.75	34.52	5.5	18	15.98
	1.0	33.33	***3***	***10***	***10.17***
R-FD	0.25	***30.95***	8	22.5	18.71
	0.5	32.14	8	28	21.79
	0.75	33.33	8	24	21.03
	1.0	32.14	9	29.5	24.62
FDD	0.25	31.55	9	24	20.48
	0.5	32.14	7	21	19.414
	0.75	32.14	6	17.5	15.68
	1.0	34.52	4	11.5	11.2

3.3 Analysis of Dynamic Properties

Given the encouraging results for the classification benchmark problems an analysis is conducted into the evolution of fitness and node count on a tournament-by-tournament basis. To do so, the classification accuracy and node count of each tournament winner is averaged over a 300-tournament window for each of the 50 trials. This information is then plotted as an *average* with respect to tournament for the 15,000 tournaments comprising a trial. For conciseness we will consider the case of Liver Disease alone, with similar results being produced for Breast Cancer.

Figures 3, 4 and 5 summarize the evolution of training classification accuracy for Fitness Directed, Fitness Difference Directed, and Roulette-Fitness Directed crossover respectively. In each case the uniform crossover classification accuracy is included to establish the base line (×). The clear distinction between increasing ratios of directed crossover and classification accuracy is readily apparent for Fitness Directed and Fitness Difference Directed crossover, Figures 3 and 4; whereas Roulette-Fitness Directed crossover closely mimics the performance of uniform sub-tree crossover throughout, Figure 5.

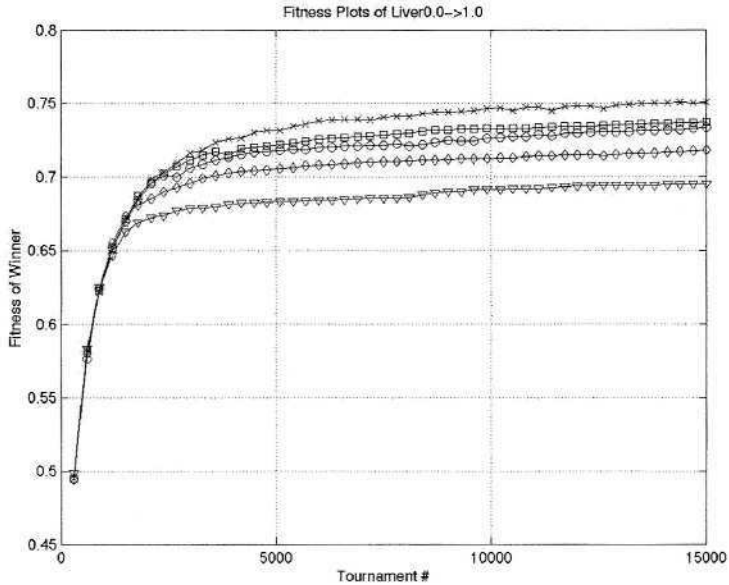


Fig. 3. Average Training Classification Accuracy - Fitness Directed Crossover. Crossover types: × - 100% uniform; ○ - 25% directed; □ - 50% directed; ◊ - 75% directed; ∇ - 100% directed

Figures 6, 7 and 8 repeat the analysis in terms of average node counts as evolution progresses. Here again, the contrast between different forms of directed crossover is readily apparent. Each increase in the ratio of Fitness Directed and Fitness Difference Directed crossover results in step reductions to the rate of bloat, such that by the 100% application of directed crossover, (average) node count is constant with increasing evolutionary cycles, Figures 6 and 7. Moreover, even in the case of Roulette-

Fitness Difference crossover, the contribution of each increase to the directed crossover ratio provides a clear decrease to the rate of average node count, Figure 8.

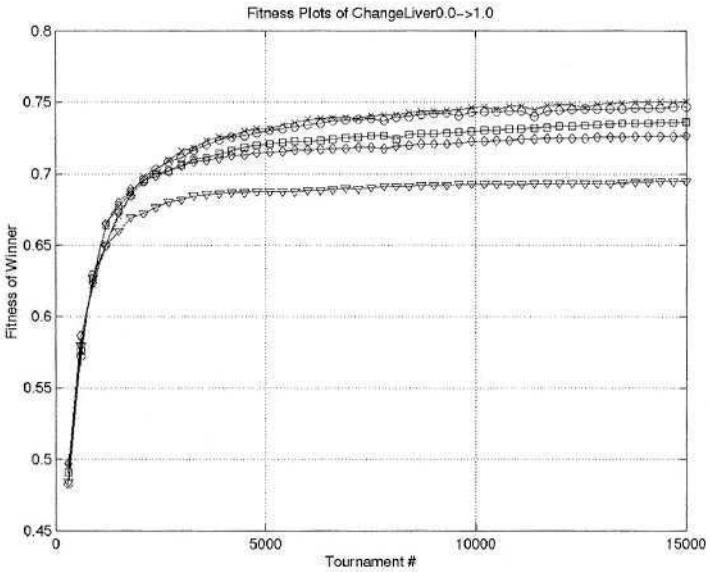


Fig. 4. Average Training Classification Accuracy – Fitness-Difference Directed Crossover. Crossover types: × - 100% uniform; O - 25% directed; □ - 50% directed; ◊ - 75% directed; ∇ - 100% directed

3.4 Solution Transparency

As indicated in the introduction, one of the desired properties assumed to coincide with parsimonious solutions is an increased transparency in programs provided by GP. In order to assess this hypothesis we utilize the simplification engine provided as part of the MAPLE™ system for symbolic math [13]. The motivation is to use the MAPLE™ symbolic simplification tool to represent the case of an average user who is used to judge the solutions provided by GP.

A scatter plot is used to summarize the effect of applying the simplification engine of MAPLE™ to each of the 50 solutions to the Liver Disease data set for both uniform sub-tree crossover and the Fitness Directed crossover (100%), Figure 6. That is, we compare solutions from 100% uniform selection of crossover points with 100% deterministic selection of crossover points. Any points along a line *"y = x"* indicate that no simplification took place; points above this line indicate that an expression became more complicated after application of MAPLE™; and points below indicate that an expression was simplified after application of MAPLE™. Note also that a log scale is employed on both axes.

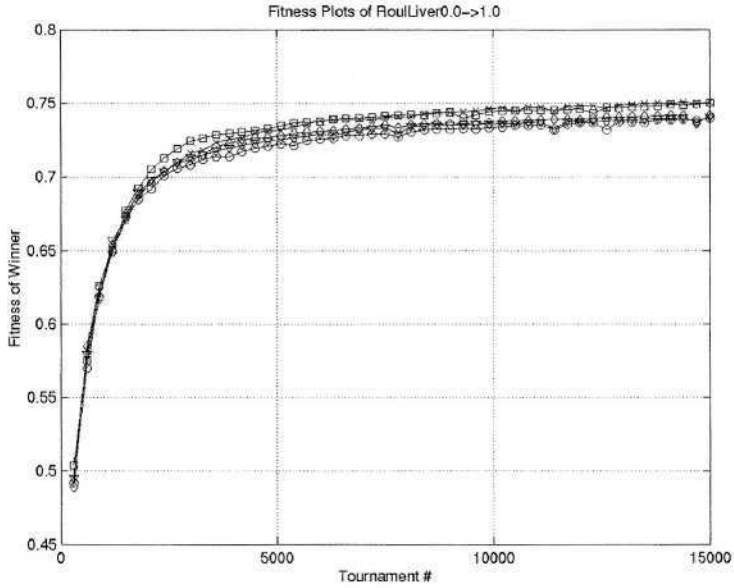


Fig. 5. Average Training Classification Accuracy – Roulette Fitness Directed Crossover. Crossover types: × - 100% uniform; ○ - 25% directed; □ - 50% directed; ◊ 75% directed; ∇ - 100% directed

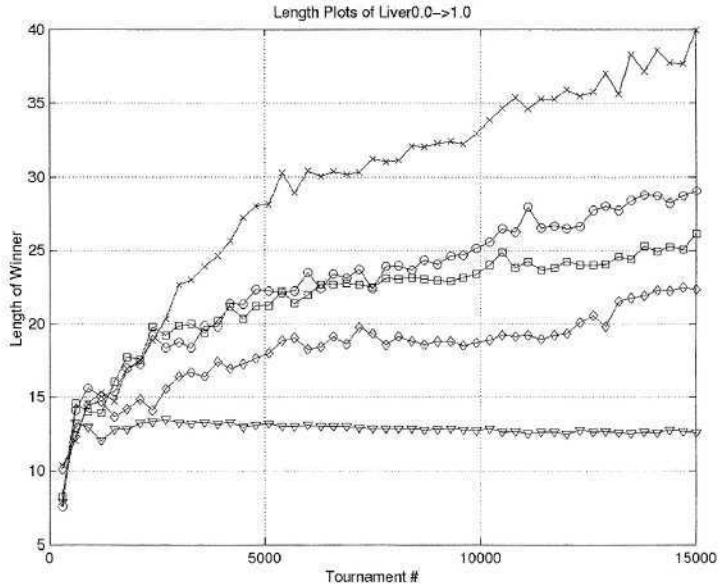


Fig. 6. Average Node Count - Fitness Directed Crossover. Crossover types: × - 100% uniform; ○ - 25% directed; □ - 50% directed; ◊ - 75% directed; ∇ - 100% directed

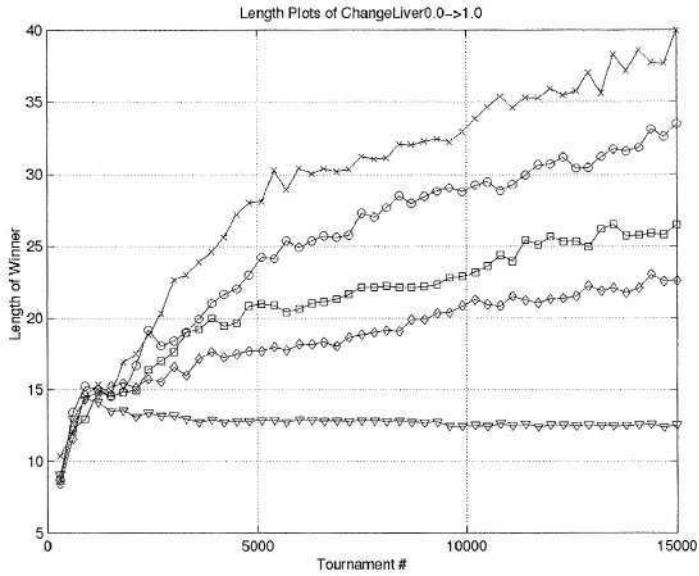


Fig. 7. Average Node Count - Fitness-Difference Directed Crossover. Crossover types: × - 100% uniform; ○ - 25% directed; □ - 50% directed; ◊ - 75% directed; ∇ - 100% directed

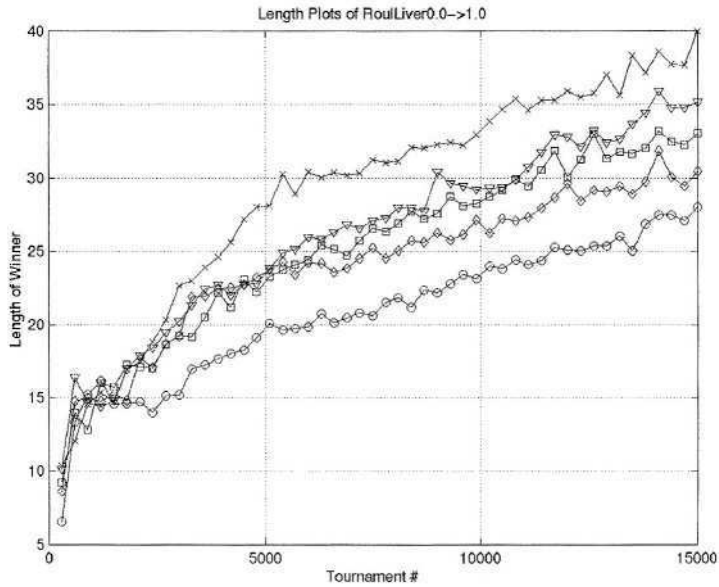


Fig. 8. Average Node Count - Roulette-Fitness Directed Crossover. Crossover types: × - 100% uniform; ○ - 25% directed; □ - 50% directed; ◊ - 75% directed; ∇ - 100% directed

In the case of uniform sub-tree crossover it is apparent that on this problem, solutions actually became more complicated following application of MAPLE™. Specifically, an average increase to term count of 168.7, over the 33 cases that increased after simplification, verses an average reduction in term count of 6.2 over 5 cases, with respect to the GP solution provided before simplification. In the case of 100% Fitness Directed crossover there is an average increase of 6.8 terms over 30 cases whereas no cases produced a decrease in code length with respect to the GP solution before simplification. It is also apparent that cases resulting in an increase following 'simplification' also predominate instances of longer programs with uniform crossover. Thus, the uniform application of crossover appears to result in individuals that are both 'bloated' and do not simplify on application of our 'base user' (the MAPLE™ symbolic simplification tool).

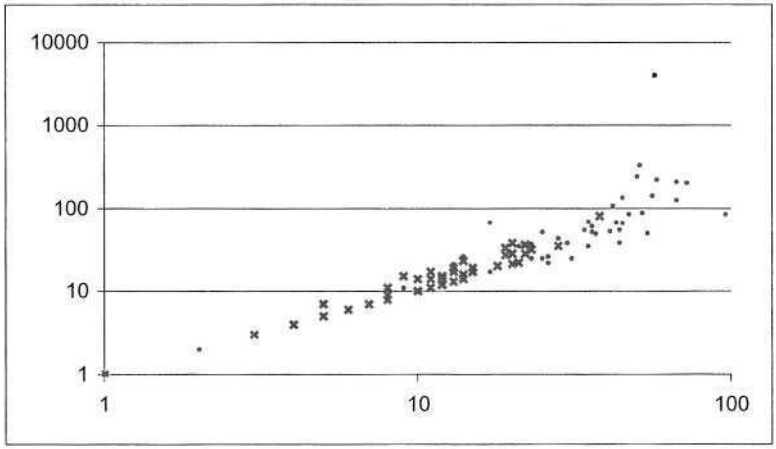


Fig. 9. Scatter plot - Solution Simplicity on Liver Disease Classification Problem, x-axis denotes before simplification; y-axis after simplification; × - case of 100% fitness directed crossover; o - uniform crossover.

4 Conclusion

Several naïve biases were defined to provide a set of directed crossover operators using only the fitness information readily available during fitness evaluation. Increasing levels of determinism associated with selection of crossover points steadily reduces the complexity of an individual at the same time that absolute fitness may also decrease. Moreover, in the case of the classification problems investigated, it appears that node count was proportional to the ratio of uniform to directed crossover operators. Finally, it is also apparent that 'bloated' GP solutions (case of no crossover bias) did not necessarily result in significant post evolution simplification when using the simplification heuristics of the MAPLE™ symbolic simplification tool. Future work will evaluate the biases under a wider range of problems.

Acknowledgements. The authors gratefully acknowledge the support of a Discovery Grant from the Natural Sciences and Engineering Research Council (NSERC) of Canada.

References

1. Koza J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, ISBN 0-262-11170-5 (1992)
2. Langdon W.B., Poli R.: Foundations of Genetic Programming. Springer-Verlag. ISBN 3-540-42451-2 (2002)
3. Blickle T., Thiele L.: Genetic Programming and Redundancy, Genetic Algorithms within the Framework of Evolutionary Computation. Workshop at KI-94, Max-Planck-Institut fur Informatik, MPI-1-94-241 (1994) 33-38
4. McPhee N.F., Miller J.D.: Accurate Replication in Genetic Programming, Proceedings of the 6th International Conference on Genetic Algorithms, Morgan Kaufmann, (1995) 303-309
5. Soule T., Foster J.A., Dickinson J.: Code Growth in Genetic Programming, Proceedings of the 1st Annual Conference on Genetic Programming, MIT Press (1996) 215-223
6. Langdon W.B., Poli R.: Fitness Causes Bloat, 2nd On-line World Conference on Soft Computing in Engineering Design and Manufacturing (WSC2) 1997.
7. Soule T., Foster J.A.: Effects of Code Growth and Parsimony Pressure on Populations in Genetic Programming, Evolutionary Computation, 6(4) (1998) 293-309
8. Smith P.W.H., Harries K.: Code Growth, Explicitly Defined Introns, and Alternative Selection Schemes, Evolutionary Computation, 6(4) (1998) 339-360
9. Iba H., de Garis H.: Extending Genetic Programming with Recombinative Guidance. In: Angeline P.J., and Kinnear K.E., eds., Chapter 4, Advances in Genetic Programming II, MIT Press, (1996) 69-88
10. Borjarczuk C.C., Lopes H.S., Freitas A.A.: Genetic Programming for Knowledge Discovery in Chest-Pain Diagnosis, IEEE Engineering in Medicine and Biology Magazine. 19(4), July-August (2000) 38-44
11. Universal Problem Solvers Inc., Machine Learning Data Sets. http://pages.prodigy.com/upso/datasets.htm
12. Quinlan J.R.: C4.5: Programs for Machine Learning. Morgan Kaufman. ISBN 1-55860-238-0. (1993) (for c4.5 v c5.0 comparison see http://www.rulequest.com/)
13. Waterloo Maple, Maple 8. http://www.mapleapps.com/

Fitness Clouds and Problem Hardness in Genetic Programming

Leonardo Vanneschi[2], Manuel Clergue[1], Philippe Collard[1], Marco Tomassini[2], and Sébastien Vérel[1]

[1] I3S Laboratory, University of Nice, Sophia Antipolis, France
[2] Information Systems Department, University of Lausanne, Lausanne, Switzerland

Abstract. This paper presents an investigation of genetic programming fitness landscapes. We propose a new indicator of problem hardness for tree-based genetic programming, called *negative slope coefficient,* based on the concept of fitness cloud. The negative slope coefficient is a predictive measure, i.e. it can be calculated without prior knowledge of the global optima. The fitness cloud is generated via a sampling of individuals obtained with the Metropolis-Hastings method. The reliability of the negative slope coefficient is tested on a set of well known and representative genetic programming benchmarks, comprising the binomial-3 problem, the even parity problem and the artificial ant on the Santa Fe trail.

1 Introduction

Genetic Programming (GP) has had an indeniable practical success in its fifteen years of existence [13,14]. However, it is still difficult to understand why some problems are easily solved by GP, while others resist solution or require massive amounts of computational effort. It would thus be of interest if we were able to somehow classify problems according to some measure of difficulty. To start with, it might be useful to take a look at what has been done in the older field of *genetic algorithms* (GAs). Difficulty studies in GAs have been pioneered by Goldberg and coworkers [4,6,8]. Their approach is focused on the construction of functions that should *a priori* be easy or hard for GAs to solve. These ideas have been followed by many others, for instance [18,5] and have been at least partly successful in the sense that they have been the source of many ideas as to what makes a problem easy or difficult for GAs. One concept that underlies many of these approaches is based on the notion of *fitness landscape.* The metaphor of a fitness landscape [22], although not without faults, has been a fruitful one in several fields. In particular, a statistic called *fitness distance correlation* (FDC) [10] has been studied in detail in the past in the context of GAs. Its suitability for GP has been investigated in [2,24,23]. As far as GP is concerned, but also in the GA field, the general conclusion of these studies was that the FDC can be considered as a rather reliable indicator of problem hardness. However, it has some severe drawbacks: sometimes the measure does not give any indication, and problems can be constructed for which the FDC leads to contradictory conclusions. The first consideration is not really serious since it manifests itself rarely and other tools, such as the analysis of the fitness-distance scatterplot can be brought to bear in these cases. On the other hand, the existence of counterexamples casts a shadow on the

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 690–701, 2004.

usefulness of the FDC, although such cases are typically contrived ones and they do not seem to appear often among "natural" problems. But the really annoying fact about FDC, and its main weakness in our opinion, is that the optimal solution (or solutions) must be known beforehand, which is obviously unrealistic in applied search and optimization problems, and prevents us from applying FDC to many GP benchmarks and real-life applications. Thus, although the study of FDC is an useful first step, we present another approach based on quantities that can be measured without any explicit knowledge of the genotype of optimal solutions, such as different kinds of fitness distributions. A second consideration concerns the way in which space is sampled: uniform random sampling has the merit of being simple and algorithm-independent (only random search is implied), but it would be more useful to have a sample of the landscape as "seen" by a specific algorithm for it is the latter the one that is really relevant. In this way, more weight can be given to particular points in the space. Thus, sampling according to a given stationary probability distribution, like Markov chain Monte Carlo, appears to be more appropriate.

This paper is structured as follows: section 2 summarizes some techniques used by other researchers in the past few years, which inspired this work. Section 3 introduces the concept of fitness cloud. This concept is used in section 4 to define a new measure, called *negative slope coefficient* (NSC), that is proposed as an indicator of problem hardness. Section 5 briefly introduces the test problems used to verify the reliability of NSC. Section 6 shows experimental results. Finally, section 7 offers our conclusions and hints for future work.

2 Fitness-Fitness Correlation: Previous Work

In genetic algorithms, plotting fitness against some features is not a new idea. Manderick *et al.* [17] study the correlation coefficient of genetic operators: they compute the correlation between the fitnesses of a number of parents and the fitnesses of their offspring. Grefenstette [7] uses fitness distribution of genetic operators to predict GA behaviour. Rosé *et al.* [20] develop the *density of states* approach by plotting the number of genotypes with the same fitness value. Smith *et al.* [21] focus on notions of *evolvability* and *neutrality;* they plot the average fitness of offspring over fitness according to Hamming neighbouring. *Evolvability* refers to the efficiency of evolutionary search. It is defined by Altenberg as "the ability of an operator/representation scheme to produce offspring that are fitter than their parents" [1].

Fitness correlation measures are almost absent in GP but there are a few precursors. The first work we are aware of is the article of Kinnear [12] in which GP difficulty is analysed through the use of the fitness *autocorrelation function,* as first proposed by Weinberger [26]. While fitness autocorrelation analysis has been useful in the study of NK landscapes [26], Kinnear found his results inconclusive and difficult to interpret: essentially no simple relationship was found between correlation length values and GP hardness. The work of Nikolaev and Slavov [19] also makes use of the fitness autocorrelation function with the main purpose of determining which mutation operator, among a few that they propose, "sees" a smoother landscape on a particular problem. Their analysis however, does not lead to a general study of problem difficulty in GP, which is our aim here. More recently, a much more detailed study of fitness correlation in GP has

been presented by Igel and Chellapilla [9]. In this work, several fitness-fitness probability distributions are analyzed on four typical GP problems as tools for understanding the effect of variation operators on the search. They favor using a combination of variation operators rather than a single one, and suggest that fitness distributions measures might be used at execution time to dynamically set the operator probabilities in order to be more exploitative or explorative. An off-line analysis of the results is also hinted at, leading to a possibly better operator choice for similar problems. Although interesting, this work does not really deal with GP difficulty; it is rather an attempt to automatically tune the evolutionary search and determine the most efficient variation operators for a given problem class. Some GP lansdcapes have been systematically investigated by Langdon and Poli [15]. Their results will be compared with our findings in some cases (see section 6).

3 Fitness Cloud

In this section we use the fitness cloud metaphor, first introduced in [25] by Vérel and coworkers for binary landscapes. In order to get a visual rendering of evolvability, for each string x in the genotype space a point is plotted, the abscissa of which is its fitness value f, and the ordinate the fitness $\tilde{f}$ of a particular neighbour that can be chosen in many different ways. The result is a scatterplot, that was called the *fitness cloud* in [25].

3.1 Evolvability

One feature that is intuitively linked, although not exactly identical, to problem difficulty is evolvability, i.e. the capacity of genetic operators to improve fitness quality. The most natural way to study evolvability is to plot the fitness values of individuals against the fitness values of their neighbors, where a neighbor is obtained by applying one step of a genetic operator to the individual. The genetic operator used here is standard subtree mutation [13], i.e. a random subtree of a selected individual is replaced by a randomly generated tree.

Formally, let Γ be the whole search space of a GP problem and $V(\gamma)$ the set of all the neighbors of individual $\gamma \in \Gamma$, obtained by applying one step of standard subtree mutation to it. The subtree mutation operator is not allowed to choose the root node as its insertion point, thus $V(\gamma)$ is different from the entire search space.

Now let f be the fitness function of the problem at hand. We define the following set of points on a bidimensional plane: $S = \{(f(\gamma), f(\nu)), \forall \gamma \in \Gamma, \forall \nu \in V(\gamma)\}$. The graphical representation of S is the scatterplot of the fitness of all the individuals belonging to the search space vs. the fitness of all their neighbors. We hypothesize that the shape of this scatterplot can give an indication of the evolvability of the genetic operators used and thus some hints about the difficulty of the problem at hand.

The fitness cloud implicitly gives some insight on the genotype to phenotype map. The set of genotypes that all have equal fitness is a *neutral set* [11]. Such a set corresponds to one abscissa in the fitness/fitness plan; according to this abscissa, a vertical slice from the cloud represents the set of fitnesses that could be reached from this set of neutrality. For a given offspring fitness value $\tilde{f}$, an horizontal slice represents all the fitness values from which one can reach $\tilde{f}$.

3.2 Sampling Methodology

In general, the size of the search space doesn't allow to consider all the possible individuals. Thus, we need to use samples. Sampling the program space according to a uniform probability distribution gives the same weight to all the sampled points. However, as it happens, many of those points are not really significant from the problem space point of view. For example, we might be repeatedly sampling points belonging to the same plateau of fitness, which may be wasted effort. For this reason, we prefer to sample the space according to a distribution that gives more weight to "important" values in the space, for instance those at a higher fitness level. This is also the case of any biased searcher such as an evolutionary algorithm, simulated annealing and other heuristics, and thus the sampling process more closely simulates the way in which the program space would be traversed by a searcher. This is a standard problem and it is well known that the sampling can be done by using Metropolis method [16] or any other equivalent importance sampling technique employed in simulation. Here we use *Metropolis-Hastings* sampling, that is an extension of Metropolis to non-symmetric stationary probability distributions. This technique can be defined as follows. Let α be the function defined as:

$$\alpha(x,y) = min\{1, \frac{y}{x}\},$$

and $f(\gamma_k)$ be the fitness of individual γ_k. A sample of GP individuals $\{\gamma_1, \gamma_2, \ldots, \gamma_n\}$ is built with the following algorithm:

begin
  γ_1 is generated uniformly at random;
  for $k := 2$ **to** n **do**
    1. an individual δ is generated uniformly at random;
    2. a random number u is generated from a
       uniform $(0,1)$ distribution;
    3. **if** $(u \leq \alpha(f(\gamma_{k-1}), f(\delta)))$
         then $\gamma_k := \delta$
         else goto 1.
       endif
    4. $k := k+1$;
  endfor
end

For each sampled point, the neighbors of that point must be generated. The number of all the possible neighbors of a tree depends on the particular genetic operator used. For standard subtree mutation, as used here, this number is huge. On the other hand, studying all possible neighbors of each sampled individual is worthless since most of them will be discarded by selection as soon as they are created. Thus, instead of studying the set $V(\gamma)$ of all the neighbors of each sampled individual γ, we only consider a subset of size $q \ll |V(\gamma)|$, obtained by applying tournament selection to its elements: to obtain each one of these q neighbors, a set of r random neighbors is generated (where r is the tournament size), by applying one step of subtree mutation to γ, and the best one is then chosen.

The terminology of section 3.1 is thus updated as follows: we refer to Γ as a sample of individuals obtained with the Metropolis-Hastings technique and, for each γ belonging to Γ, we refer to $V(\gamma)$ as a subset of size q of its neighbors, obtained by the application of the tournament selection mechanism.

4 Negative Slope Coefficient

The fitness cloud can be of help in determining some characteristics of the fitness landscape related to evolvability and problem difficulty. But the mere observation of the scatterplot is not sufficient to quantify these features. In this section, we present an algebraic measure, called *negative slope coefficient* (NSC), that we propose as a new indicator of problem difficulty for GP.

The abscissas of a scatterplot can be partitioned into m segments $\{I_1, I_2, \ldots, I_m\}$ of the same length. Analogously, a partition of the ordinates $\{J_1, J_2, \ldots, J_m\}$ can be done, where each segment J_i contains all the ordinates corresponding to the abscissas contained in I_i.

Let $M_1, M_2, \ldots, M_m$ be the averages of the abscissa values contained inside the segments $I_1, I_2, \ldots, I_m$ and let $N_1, N_2, \ldots, N_m$ be the averages of the ordinate values in $J_1, J_2, \ldots, J_m$. Then we can define the set of segments $\{S_1, S_2, \ldots, S_{m-1}\}$, where each S_i connects the point (M_i, N_i) to the point (M_{i+1}, N_{i+1}). For each one of these segments S_i, the *slope* P_i can be calculated as follows:

$$P_i = \frac{N_{i+1} - N_i}{M_{i+1} - M_i}$$

Finally, we can define the negative slope coefficient as:

$$\text{NSC} = \sum_{i=1}^{m-1} min\ (P_i,\ 0)$$

We hypothesize that NSC is an indicator of problem difficulty in the following sense: if NSC= 0, the problem is easy, if NSC< 0 the problem is difficult and the magnitude of NSC quantifies this difficulty: the more negative its value, the more difficult the problem. In other words, according to our hypothesis, a problem is difficult if at least one of the segments $S_1, S_2, \ldots, S_{m-1}$ has a negative slope and the sum of all the negative slopes gives a measure of problem hardness. The idea is that the presence of a segment with negative slope indicates a bad evolvability for individuals having fitness values contained in that segment. Note that, for the time being, we didn't try to normalize NSC values to a given range. This means that NSC results for different problems are not comparable among them.

In the following sections, NSC is tested as a measure of problem hardness on a set of well known GP benchmarks. To be able to validate difficulty predictions, we define a *performance* measure, as being the proportion of GP runs for which the global optimum has been found in less than 500 generations over 100 independent executions. Good or bad performance values correspond to our intuition of what "easy" or "hard" means in practice. All GP runs executed to calculate performance values have used the same

set of GP parameters: generational GP, population size of 200 individuals, standard GP mutation used as the sole genetic operator with a rate of 95%, tournament selection of size 10, ramped half and half initialization, maximum depth of individuals specified in the following case by case, elitism (i.e. survival of the best individual into the newly generated population).

For the sake of comparison we also measure the *fitness-fitness correlation* (FFC) [17]. Given the set $X = \{x_1, x_2, ..., x_n\}$ of all the abscissas of a scatterplot and the set $Y = \{y_1, y_2, ..., y_n\}$ of all the ordinates of the same scatterplot, the FFC is defined as: $C_{XY}/\sigma_X\sigma_Y$, where $C_{XY} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})$ is the covariance of X and Y, and σ_X, σ_Y, $\overline{x}$, $\overline{y}$ are the standard deviations and means of X and Y.

5 Test Problems

Problems used in this work are briefly described below. For a more detailed description, see [3,13]. Except for the ant problem, which is included because of the large body of knowledge accumulated on it, the other problems have been chosen because they are representative of important problem classes (respectively symbolic regression and boolean), and their difficulty can be tuned.

The binomial-3 problem. This benchmark (first introduced by Daida *et al.* in [3]) is an instance of the well known symbolic regression problem. The function to be approximated is $f(x) = 1 + 3x + 3x^2 + x^3$. Fitness cases are 50 equidistant points over the range [–1, 0). Fitness is the sum of absolute errors over all fitness cases. A hit is defined as being within 0.01 in ordinate for each one of the 50 fitness cases. The function set is $\mathcal{F} = \{+, -, *, //\}$, where // is the protected division, i.e. it returns 1 if the denominator is 0. The terminal set is $\mathcal{T} = \{x, \mathcal{R}\}$, where x is the symbolic variable and $\mathcal{R}$ is the set of ephemeral random constants (ERCs). ERCs are uniformly distributed over a specified interval of the form $[-a_{\mathcal{R}}, a_{\mathcal{R}}]$, they are generated once at population initialization and they are not changed in value during the course of a GP run. According to Daida and coworkers [3], difficulty tuning is achieved by varying the value of $a_{\mathcal{R}}$.

The even parity k problem. The boolean even parity k function [13] of k boolean arguments returns *true* if an even number of its boolean arguments evaluates to true, otherwise it returns *false*. The number of fitness cases to be checked is 2^k. Fitness is computed as 2^k minus the number of hits over the 2^k cases. Thus a perfect individual has fitness 0, while the worst individual has fitness 2^k. The set of functions we employed is $\mathcal{F} = \{NAND, NOR\}$. The terminal set is composed of k different boolean variables. Difficulty tuning is achieved by varying the value of k.

Artificial ant on the Santa Fe trail. In this problem, an artificial ant is placed on a 32×32 toroidal grid. Some of the cells from the grid contain food pellets. The goal is to find a navigation strategy for the ant that maximizes its food intake. We use the same set of functions and terminals as in [13] and the same trail definition. As fitness function, we use the total number of food pellets lying on the trail (89) minus the amount of food eaten by the ant during his path. This turns the problem into a minimization one, like the previous ones.

6 Experimental Results

6.1 The Binomial-3 Problem

Figure 1 shows the scatterplots and the set of segments $\{S_1, S_2, \ldots, S_m\}$ as defined in section 4 (with $m = 10$) for the binomial-3 problem with $a_{\mathcal{R}} = 1$ (figure 1(a)), $a_{\mathcal{R}} = 10$ (figure 1(b)), $a_{\mathcal{R}} = 100$ (figure 1(c)) and $a_{\mathcal{R}} = 1000$ (figure 1(d)). Parameters used are as follows: maximum tree depth = 26, $|\Gamma| = 40000$, i.e. a sample of 40000 individuals has been used, obtained with the Metropolis-Hastings sampling, $\forall \gamma \in \Gamma \ \ |V(\gamma)| = 1$, i.e. for each sampled individual, only one neighbor has been considered. It has been obtained by one step of tournament selection and standard subtree mutation has been used as the operator to generate the neighborhood.

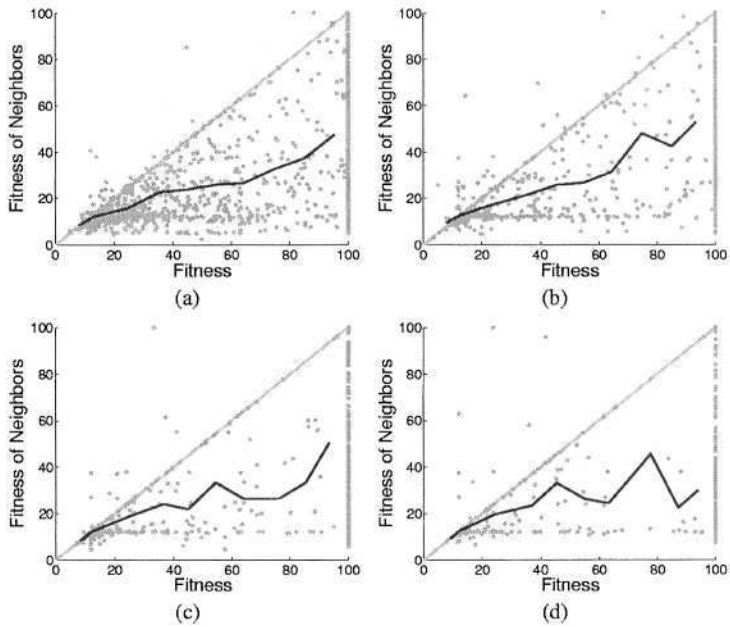


Fig. 1. Binomial-3 results. (a): $a_{\mathcal{R}} = 1$. (b): $a_{\mathcal{R}} = 10$. (c): $a_{\mathcal{R}} = 100$. (d): $a_{\mathcal{R}} = 1000$. Fitness clouds have been obtained by a sample of 40000 individuals.

Table 1 shows some data about these experiments. Column one of table 1 represents the corresponding scatterplot in figure 1. Column two contains the $a_{\mathcal{R}}$ value. Column three contains performance. Columns four and five contain values of NSC and FFC respectively.

Table 1. Binomial-3 problem. Some data related to scatterplots of figure 1.

scatterplot	$a_{\mathcal{R}}$	p	NSC	FFC
fig. 1(a)	1	0.89	0	0.70
fig. 1(b)	10	0.42	-0.53	0.74
fig. 1(c)	100	0.35	-1.01	0.75
fig. 1(d)	1000	0.29	-3.39	0.75

Table 2. Even parity. Indicators related to scatterplots of figure 2.

scatterplot	problem	p	NSC	FFC
fig. 2(a)	even parity 3	0.98	0	0.56
fig. 2(b)	even parity 5	0.01	-0.11	0.39
fig. 2(c)	even parity 7	0	-0.49	0.25
fig. 2(d)	even parity 9	0	-0.55	0.23

These results show that NSC values get smaller as the problem becomes harder, and it is zero when the problem is easy ($a_{\mathcal{R}} = 1$). On the other hand, FFC doesn't seem to give any indication about problem difficulty, which confirms Kinnear's observations [12]. Moreover, the points in the scatterplots seem to cluster around good (i.e. small) fitness values as the problem gets easier (remark: points having a fitness value above 100 have not been visualized in the scatterplots of figure 1 for the sake of clarity, even though, of course, they have been used to calculate the NSC). In conclusion, the presence or absence of segments with negative slope (quantified by the NSC), in conjunction with the graphical representation of the scatterplot, seems to be useful to estimate problem hardness.

6.2 The Even Parity k Problem

Figure 2 shows the scatterplots and the set of segments $\{S_1, S_2, \ldots, S_m\}$ (where m has been set to 6) for the even parity 3, even parity 5, even parity 7 and even parity 9 problems. Parameters used are as follows: maximum tree depth = 10, $|\Gamma| = 40000$ obtained with the Metropolis-Hastings sampling, $\forall \gamma \in \Gamma \ \ |V(\gamma)| = 1$. Tournament has been used as a selection mechanism and standard subtree mutation as the operator for generating the neighborhood.

Table 2 shows some data about these experiments with the same notation and meaning as in table 1, except that column two now refers to the problem rank. Analogously to what happens for the binomial-3 problem, NSC values get smaller as the problem becomes harder, they are always negative for hard problems, and zero for easy ones. Once again the points in the scatterplots seem to cluster around good fitness values as the problem gets easier. These results are in qualitative agreement with intuition and with those found by other researchers (e.g. [13]). Thus, the utility of NSC as an indicator of problem hardness, together with the graphic representation of the fitness cloud, is confirmed.

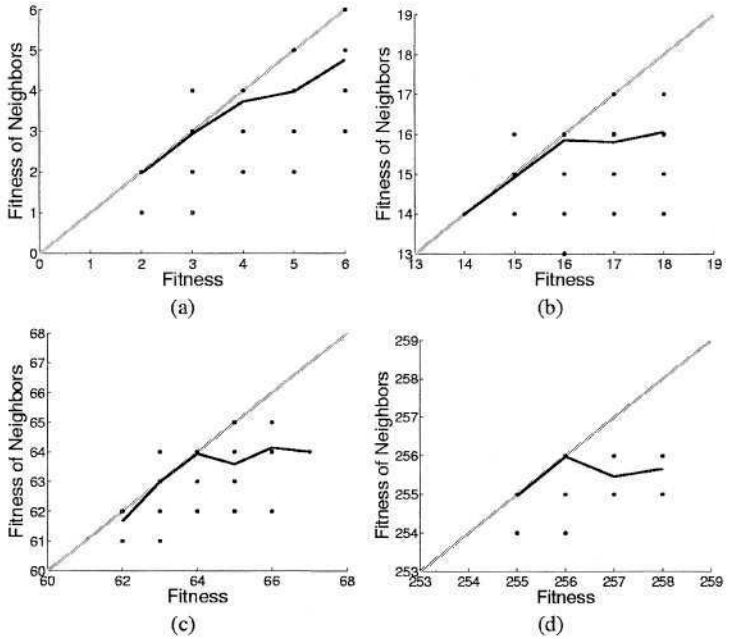


Fig. 2. Results for the even parity k problem. (a): Even parity 3. (b): Even parity 5. (c): Even parity 7. (d): Even parity 9. Fitness clouds have been obtained by a sample of 40000 individuals. Note that many sampled individuals may have the same fitness value.

6.3 The Artificial Ant on the Santa Fe Trail

This problem, among others, has been studied in depth by Langdon and Poli. In [15] they did a detailed analysis of the problem's fitness landscape by enumeration for small programs, and by sampling for bigger program sizes. They also studied the problem from the point of view of the schema theory and found it to be deceptive. Figure 3 shows the scatterplots and the set of segments $\{S_1, S_2, \ldots, S_m\}$ (where m has been set to 10) for the artificial ant problem with maximum tree depths equal to 10 and 6 respectively. The other parameters used are: $|\Gamma| = 40000$ obtained with the Metropolis-Hastings sampling, $\forall \gamma \in \Gamma \ \ |V(\gamma)| = 1$. Tournament has been used as a selection mechanism and standard subtree mutation as the operator to generate the neighborhood.

Table 3 shows the results of the experiments, where column two indicates the maximum program depth allowed, and the other values have the usual meaning. Both problems turn out to be difficult and NSC is negative for both problem instances. Indeed, there are many local optima in the ant space and the search can become easily trapped in one of them. Larger programs appear to be slightly easier to search for a solution than smaller ones. This is in agreement with Langdon's and Poli's findings which say that the number of solutions increases exponentially with program size. The unusual shape of the scatterplot and the fact that some mean segments are horizontal can also be accounted for in terms of Langdon's and Poli's analysis. They found that, for a given program, most

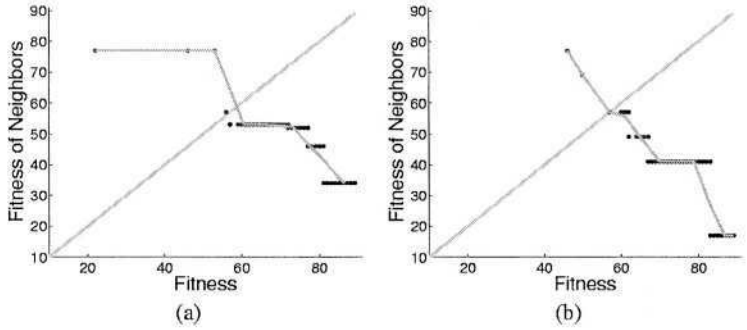


Fig. 3. (a): Artificial Ant problem with maximum tree depth equal to 10. (b): Artificial Ant problem with maximum tree depth equal to 6. Fitness clouds have been obtained by a sample of 40000 individuals. Note that many sampled individuals may have the same fitness value.

neighbors have the same fitness or are worse; a fact that is confirmed here. Moreover, the phenomenon is more marked for longer programs, again confirmed by figure 3. We also see that the fitter the individual, the more difficult becomes to improve it, a fact that is represented by the large negative slope segment at low fitness (remember that we have defined fitness so that low values are better). This too is in agreement with the analysis in [15].

Table 3. Ant problem results. Some data related to scatterplots of figure 3.

scatterplot	max. tree depth	p	NSC	FFC
fig. 3(a)	10	0.05	-6.06	-0.88
fig. 3(b)	6	0	-11.42	-0.82

7 Conclusions and Future Work

A new indicator of problem hardness for GP, called negative slope coefficient, is proposed in this work. This measure is based on the concept of fitness cloud, that visualizes on a plane the relationship between the fitness values of a sample of individuals and the fitness of some of their neighbors. Sampling has been done with the Metropolis-Hastings technique, so as to give more weight to individuals with good fitness and to preserve the original distribution of the fitness function over all the search space. Contrary to fitness distance correlation, NSC is predictive, i.e. it can be used without prior knowledge of the global optima. Thus, NSC can be used to quantify difficulty of standard GP benchmarks, where the cardinality and shape of global optima is not known *a priori*. Studies over three well-known GP benchmarks (binomial-3, even parity and artificial ant) have been presented here, confirming the suitability of NSC as measure of problem

hardness, at least for the cases studied. Such a systematic study of GP problem hardness, performed using a well defined algebraic measure, had, to our knowledge, never been applied to these benchmarks before. Other experiments on these benchmarks using a different sampling technique (standard Metropolis) and on other classes of tunably difficult problems (like trap functions and royal trees) have been done. Fitness clouds using more than one neighbor for each sampled individual have been analysed. Moreover, fitness clouds obtained by applying more than once the mutation operator at each sampled individual have been studied. Finally, a less disruptive mutation operator (structural mutation introduced in [24]) has been used. All these results (not shown here for lack of space) confirm the suitability of NSC as an indicator of problem hardness.

Future work includes a more exhaustive study of NSC and other measures based on fitness clouds over a wider set of GP benchmarks, the use of more sophisticated sampling techniques and the study of techniques to automatically define some quantities that in this paper have been chosen somehow arbitrarily, such as the number of segments in which the fitness cloud is partitioned and their size. Furthermore, we will study NSC from a statistical point of view, on order to understand the significance of the results obtained and how they change from one sampling to the other. Since we don't believe NSC to be infallible, as is true for any statistic based on samples, these studies should also lead to the discovery of some drawbacks of this measure that should inspire future extensions.

Acknowledgments. L. Vanneschi and M. Tomassini gratefully acknowledge financial support by the Fonds National Suisse pour la recherche scientifique under contract 200021-100112/1.

References

1. L. Altenberg. The evolution of evolvability in genetic programming. In K. Kinnear, editor, *Advances in Genetic Programming,* pages 47–74, Cambridge, MA, 1994. The MIT Press.
2. M. Clergue, P. Collard, M. Tomassini, and L. Vanneschi. Fitness distance correlation and problem difficulty for genetic programming. In W. B. Langdon et. al., editor, *Proceedings of the genetic and evolutionary computation conference GECCO '02,* pages 724–732. Morgan Kaufmann, San Francisco, CA, 2002.
3. J. M. Daida, R. Bertram, S. Stanhope, J. Khoo, S. Chaudhary, and O. Chaudhary. What makes a problem GP-hard? analysis of a tunably difficult problem in genetic programming. *Genetic Programming and Evolvable Machines,* 2:165–191, 2001.
4. K. Deb and D. E. Goldberg. Analyzing deception in trap functions. In D. Whitley, editor, *Foundations of Genetic Algorithms, 2*, pages 93–108. Morgan Kaufmann, 1993.
5. S. Forrest and M. Mitchell. What makes a problem hard for a genetic algorithm? some anomalous results and their explanation. *Machine Learning,* 13:285–319, 1993.
6. D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Boston, MA, 1989.
7. J. Grefenstette. Predictive models using fitness distributions of genetic operators. In D. Whitley and M. Vose, editors, *Foundations of Genetic Algorithms, 3,* pages 139–161. Morgan Kaufmann, 1995.

8. J. Horn and D. E. Goldberg. Genetic algorithm difficulty and the modality of the fitness landscapes. In D. Whitley and M. Vose, editors, *Foundations of Genetic Algorithms, 3,* pages 243–269. Morgan Kaufmann, 1995.
9. C. Igel and K. Chellapilla. Fitness distributions: Tools for designing efficient evolutionary computations. In L. Spector, W. B. Langdon, U.-M. O'Reilly, and P. Angeline, editors, *Advances in Genetic Programming 3,* pages 191–216, Cambridge, MA, 1999. The MIT Press.
10. T. Jones. *Evolutionary Algorithms, Fitness Landscapes and Search.* PhD thesis, University of New Mexico, Albuquerque, 1995.
11. M. Kimura. *The Neutral Theory of Molecular Evolution.* Cambridge University Press, Cambridge, UK, 1983.
12. K. E. Kinnear. Fitness landscapes and difficulty in genetic programming. In *Proceedings of the First IEEEConference on Evolutionary Computing,* pages 142–147. IEEE Press, Piscataway, NY, 1994.
13. J. R. Koza. *Genetic Programming.* The MIT Press, Cambridge, Massachusetts, 1992.
14. J. R. Koza, M. J. Streeter, and M. A. Keane. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence.* Kluwer Academic Publishers, Boston, MA, 2003.
15. W. B. Langdon and R. Poli. *Foundations of Genetic Programming.* Springer, Berlin, 2002.
16. N. Madras. *Lectures on Monte Carlo Methods.* American Mathematical Society, Providence, Rhode Island, 2002.
17. B. Manderick, M. de Weger, and P. Spiessens. The genetic algorithm and the structure of the fitness landscape. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms,* pages 143–150. Morgan Kaufmann, 1991.
18. M. Mitchell, S. Forrest, and J. Holland. The royal road for genetic algorithms: fitness landscapes and ga performance. In F. J. Varela and P. Bourgine, editors, *Toward a Practice of Autonomous Systems, Proceedings of the First European Conference on Artificial Life,* pages 245–254. The MIT Press, 1992.
19. N. I. Nikolaev and V. Slavov. Concepts of inductive genetic programming. In W. Banzhaf et. al., editor, *Genetic Programming, Proceedings of EuroGP'1998,* volume 1391 of *LNCS,* pages 49–59. Springer-Verlag, 1998.
20. H. Rosé, W. Ebeling, and T. Asselmeyer. The density of states - a measure of the difficulty of optimisation problems. In H.-M. Voigt et al., editor, *Parallel Problem Solving from Nature - PPSN IV,* volume 1141 of *Lecture Notes in Computer Science,* pages 208–217. Springer-Verlag, Heidelberg, 1996.
21. Smith, Husbands, Layzell, and O'Shea. Fitness landscapes and evolvability. *Evolutionary Computation,* 1(10): 1–34, 2001.
22. P. F. Stadler. Fitness landscapes. In M. Lässig and Valleriani, editors, *Biological Evolution and Statistical Physics,* volume 585 of *Lecture Notes Physics,* pages 187–207, Heidelberg, 2002. Springer-Verlag.
23. L. Vanneschi, M. Tomassini, M. Clergue, and P. Collard. Difficulty of unimodal and multimodal landscapes in genetic programming. In E. Cantú-Paz et. al., editor, *Genetic and Evolutionary Computation – GECCO-2003,* volume 2724 of *LNCS,* pages 1788–1799. Springer-Verlag, Berlin, 2003.
24. L. Vanneschi, M. Tomassini, P. Collard, and M. Clergue. Fitness distance correlation in structural mutation genetic programming. In C. Ryan et al., editor, *Genetic Programming, 6th European Conference, EuroGP2003,* volume 455–464 of *Lecture Notes in Computer Science.* Springer-Verlag, Heidelberg, 2003.
25. S. Vérel, P. Collard, and M. Clergue. Where are bottleneck in nk-fitness landscapes ? In *CEC 2003: IEEE International Congress on Evolutionary Computation. Canberra, Australia,* pages 273–280. IEEE Press, Piscataway, NJ, 2003.
26. E. D. Weinberger. Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biol. Cybern.,* 63:325–336, 1990.

Improving Generalisation Performance Through Multiobjective Parsimony Enforcement

Yaniv Bernstein, Xiaodong Li, Vic Ciesielski, and Andy Song

School of Computer Science and Information Technology
RMIT University, VIC 3001, Melbourne, Australia

Abstract. This paper describes POPE-GP, a system that makes use of the NSGA-II multiobjective evolutionary algorithm as an alternative, parameter-free technique for eliminating program bloat. We test it on a classification problem and find that while vastly reducing program size, the technique does improve generalisation performance.

Program Bloat, the phenomenon of ever-increasing program size during a GP run, is a recognised and widespread problem. Traditional techniques to combat program bloat are program size limitations or parsimony pressure (penalty functions). These techniques suffer from a number of problems, in particular their reliance on parameters whose optimal values it is difficult to *a priori* determine. In this work we study the performance of POPE-GP, a new algorithm that uses the NSGA-II multiobjective algorithm as the basis for parsimony enforcement. We are especially interested in finding out if small solutions generalise better than large solutions. To achieve this, we compare the performance of POPE-GP on a real-world classification problem with that of a GP with more traditional parsimony control and a GP with no control at all, paying particular attention to the performance of solutions on the unseen testing set as a measure of generalisation performance.

The Pseudo-Objective Parsimony Enforcement GP (POPE-GP) uses the NSGA-II multiobjective optimisation algorithm [1] as a base for its operation. The two objectives are defined as being the actual objective of the GP run (the fitness) and the size of the program. Once these objectives have been defined, the NSGA-II algorithm attempts to find the Pareto Front for these two objectives. We compared the generalisation performance of classifier programs generated by the POPE-GP algorithm with those generated by a standard GP with a depth limit of eight and one with no limits at all. We used the Wisconsin Breast Cancer Database[1], which has been widely used as a testbed for classification. We divided the 699 instances in the data set randomly into training and testing sets, so that 70% (479 instances) of the data made up the training set and the remaining 30% (204 instances) constituted the testing set. We used the RMITGP[2] GP programming library with strongly-typed GP. The fitness of an individual was taken to be the gross classification error – ie. the number of instances in the training set that are misclassified.

[1] *http://www.ics.uci.edu/~mlearn/MLSummary.html*
[2] *http://yallara.cs.rmit.edu.au/~dylanm/rmitgp.html*

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 702–703, 2004.

Table 1. (a) End-of-run average values for the algorithms tested. (b) Mean classification accuracy on the testing set.

Algorithm	AvDepth	AvFitness	AvSize	BestDepth	BestFitness	BestSize
POPE-GP (500)	6.71	0.9586	18.77	9.40	0.9865	31.50
POPE-GP (50)	5.40	0.9467	13.12	8.00	0.9811	23.20
Depth-Limited (500)	7.99	0.9388	300.79	8.00	0.9845	282.72
Depth-Limited (50)	7.97	0.9175	270.27	7.86	0.9753	261.06
No Parsimony Pressure (500)	33.99	0.9658	1266.01	21.30	0.9858	691.56

(a)

	POPE 500	POPE 50	DL 500	DL 50	No Pressure
Mean Accuracy (%)	95.971	95.932	95.463	94.537	95.151
Standard Deviation	1.065	0.954	1.466	2.442	1.268

(b)

The classification accuracies (Table 1) on the testing data lend support to the hypothesis that enforcing parsimony does lead to improved generalisation performance. In particular the two POPE-GP algorithms (50 and 500 individuals) clearly outperformed all other algorithms in terms of generalisation performance. The most interesting result was the excellent generalisation performance of POPE-GP with 50 individuals. In fact, the classification accuracy of programs generated by this algorithm on the test data was statistically indistinguishable from the POPE-GP with 500 individuals, and clearly superior to the programs generated by other algorithms. This is an impressive result considering that the algorithm makes only 10% of the evaluations made by the POPE-GP with 500 individuals. Running a standard depth-limited GP with the reduced population produced poor results. This is a vindication of the hypothesis that parsimonious solutions tend to generalise better and of the approach of using multiobjective techniques for parsimony enforcement.

The excellent performance of the POPE-GP with 50 individuals warrants further investigation and analysis, but we do have some clues to why it performed as it did. Firstly, the average size of the best performing individual in this algorithm was substantially smaller — by almost 30% — than the best individuals generated by POPE-GP with 500 individuals. Also, their classification error on the testing data was significantly lower than those produced by the 500 individual algorithm. In other words, the programs were more general and 'fit' the training data less tightly, using only the strongest predictors in the underlying data for classification purposes.

References

1. Deb, K., Pratap, A., Agarwal, S. and Meyarivan,T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation,* 6(2):182–197,(2002)

Using GP to Model Contextual Human Behavior

Hans Fernlund and Avelino J. Gonzalez

Intelligent Systems Laboratory
University of Central Florida
Orlando, FL 32816-2450
hfe@du.se, gonzalez@pegasus.cc.ucf.edu

Abstract. This paper describes a new approach that automatically builds human behavior models for simulated agents by observing human performance. This research synergistically combines Context-Based Reasoning, a paradigm especially developed to model tactical human performance within simulated agents, with Genetic Programming that is able to construct the behavior knowledge in accordance to the Context-Based Reasoning paradigm.

1 Introduction

The use of a learning system that could automatically extract knowledge and construct human behavior models could reduce development complexity and cost of building simulated agents with human characteristics. This paper presents an approach to building human behavior models automatically. The approach employs Context-Based Reasoning (CxBR) and Genetic Programming (GP) to implement the learning artifact of a methodology called Learning by Observation, which will learn the behavior of a human merely by observation. The intent is not only to use the observations to learn, but also to learn the behavior of the observed entity. The research described in this paper defines learning by observation as follows: *The agent adopts the behavior of the observed entity only through the use of data collected through observation.*

2 The Genetic Context Learning Approach

CxBR is a modeling technique that can efficiently represent the behavior of humans in software agents [2]. CxBR encapsulates into hierarchically-organized contexts the knowledge about appropriate actions and/or procedures as well as compatible new situations. Mission Contexts define the mission to be undertaken by the agent. While it does not control the agent per se, the Mission Context defines the scope of the mission, its goals, the plan, and the constraints imposed (time, weather, rules of engagement, etc). The Major Context is the primary control element for the agent. It contains functions, rules and a list of compatible next Major Contexts. Identification of a new situation can now be simplified because only a limited number of all situations are possible under the currently active context. Sub-Contexts are

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 704–705, 2004.

abstractions of functions performed by the Major Context which may be too complex for one function, or that may be employed by other Major Contexts. This encourages re-usability. Sub-Contexts are activated by rules in the active Major Context. Transitions between contexts are triggered by events in the environment – some planned, others unplanned.

The new learning algorithm presented here by merging CxBR and GP is called Genetic Context Learning (GenCL). Instead of creating the contexts by hand, we use GP to build the contexts. The GP's evolutionary process provides the CxBR frame with appropriate context's action rules and sentinel rules. The individuals in the genetic population are components of the context base and a micro simulator is used to simulate an individual's behavior. The behavior from the micro simulator is then compared with the human performance, and a fitness measure is established to evaluate the models appropriateness. The evolutionary process will strive to minimize the discrepancies between the performances of the contexts created by GP and the human performance. A complete description of GenCL can be found in Fernlund [1].

3 Results

The initial experiments used a full scale automobile driving simulator to collect data used to automatically model personalized human driving behavior by GenCL. Five different drivers were used to collect data and five different agents were evolved with different driver behavior. Further, in order to determine how useful the automatic creation of simulated agents through GenCL is, two agents were developed by an independent source in the traditional, manual way. Here a knowledge engineer interviewed and rode an automobile with two drivers.

The results show that the new GenCL learning algorithm is able to evolve human contextual knowledge in all parts of the CxBR paradigm (i.e. both the actions within a specific context and the context transitions). The agents evolved, show the ability to generalize the behavior and they inhibit long term reliability performance. The performance of the evolved simulated agents showed consistent behavior even after 40 minutes of driving. Further, their performances were competitive with agents developed manually by human developers. To develop this algorithm further, and to apply it in other applications, it could probably ease the future development of human behavior models in simulated agents.

References

1. Fernlund, H. (2004), "Evolving Models from Observed Human Performance", Doctoral dissertation, Department of Electrical and Computer Engineering, University of Central Florida, Spring, 2004.
2. Gonzalez A. J. and Ahlers, R. H. (1998) "Context-Based Representation of Intelligent Behavior in Training Simulations" Transactions of the Society for Computer Simulation International, volume 15, no 4, December 1998.

A Comparison of Hybrid Incremental Reuse Strategies for Reinforcement Learning in Genetic Programming

Scott Harmon, Edwin Rodríguez, Christopher Zhong, and William Hsu

Department of Computing and Information Sciences
Kansas State University
{sjh4069,edwin,czh9768,bhsu}@cis.ksu.edu

Easy missions is an approach to machine learning that seeks to synthesize solutions for complex tasks from those for simpler ones. ISLES (Incrementally Staged Learning from Easier Subtasks) [1] is a genetic programming (GP) technique that achieves this by using identified goals and fitness functions for subproblems of the overall problem. Solutions evolved for these subproblems are then reused to speed up learning, either as automatically defined functions (ADF) or by seeding a new GP population. Previous positive results using both approaches for learning in multi-agent systems (MAS) showed that incremental reuse using easy missions achieves comparable or better overall fitness than single-layered GP. A key unresolved issue dealt with hybrid reuse using ADF with easy missions. Results in the keep-away soccer (KAS) [2] domain (a test bed for MAS learning) were also inconclusive on whether compactness-inducing reuse helped or hurt overall agent performance. In this paper, we compare reuse using single-layered (with and without ADF) GP and easy missions GPs to two new types of GP learning systems with incremental reuse.

In our research we performed six experiments. The first experiment used standard, conventional GP without any enhancement. We will refer to this as single-layered GP. The second used using standard GP enhanced with ADF. We will refer to this as single-layered ADF. The rest of the experiments used double-layered (two stages of evolution). The third used ISLES with Standard GP in the first and second stage. We will refer to this as ISLES - SGP/SGP. The fourth used ISLES with Standard GP in the first stage and ADF in the second stage. We will refer to this as ISLES - SGP/ADF. The fifth used ISLES with ADF in the first stage and Standard GP in the second stage. We will refer to this as ISLES - ADF/SGP. The sixth and final experiment used ISLES with ADF in the first and second stage. We will refer to this as ISLES - ADF/ADF.

Each experiment was done using ECJ [3] and a KAS simulator created by S. Gustafson [1]. For both single-layered experiments, the target concept was to minimize the number of turn overs. For all of the experiments with ISLES, the first stage goal was to maximize the number of successful passes between two teammates in the absence of takers. The second stage goal was to minimize the number of turnovers from keepers (3 keepers) to takers (1 taker).

We took the average of ten runs for each experiment. The population size for all the experiments was 4000. For the single-layered experiments, we stopped at

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 706–707, 2004.

the 101th generation. For the ISLES experiments we stopped the first stage at the 10th generation and the second stage at the 90th generation. When going from the first stage to the second stage, we used the individuals from the first stage to seed the population of the second stage. For ISLES -SGP/ADF that involved putting the GP trees into the ADF portions and initializing the main trees to random sequences. For ISLES - ADF/SGP, it involved taking the ADF trees and placing then into the main trees of the GP (discarding the main trees of the ADF). For the others a simple transfer of the individuals was performed.

The results we obtained show evidence of the possible existence of a well defined spectrum of behaviors that ranges from techniques that use compactness inducing reuse (such as ADF) to those that use totally dynamic reuse (ISLES). This spectrum is defined by a trade off between high fitness and space efficiency. Our preliminary results allow us to hypothesize that for problems that surpass a certain threshold of complexity, compactness inducing techniques, like ADF, will tend to yield solutions that are far more efficient in terms of space. However, this space saving comes at the cost of suboptimality in terms of fitness. This conjecture is very interesting and poses several questions: is this phenomenon problem independent? Is this phenomenon independent of the design of the system, parameter tuning, inductive bias and other characteristics of the underlying evolutionary engine? Unfortunately, given the premature nature of our results and lack of diversity in terms of problems studied (as well as some uncertainty introduced by the small amount of repetitions), it is impossible, for now, to answer these questions. However, despite the weakness of these preliminary results in terms of support for our conjecture, the issue posed herein is a very interesting one and we believe that it should beintegrated into the current efforts of our research community.

The ideas presented in this work have a potential effect on other well developed areas such as Schema Theory and Code Bloat Theory. Our long term goal is to develop a theory that characterizes the behavior of GP in the presence of different types of reuse. This theory should have both a qualitative and quantitative value. In this sense it should not only give insight about how reuse affect the way GP searches for problems, but also serve as a tool to determine under what conditions reuse is beneficial, what kind of reuse should be applied or if it should be avoided altogether.

References

1. Hsu, W.H., Gustafson, S.M.: Genetic programming and multi-agent layered learning by reinforcements. In: GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, New York, Morgan Kaufmann Publishers (2002) 764–771
2. McAllester, D., Stone, P.: Keeping the ball from CMUnited-99. In: RoboCup-2000: Robot Soccer World Cup IV. Springer Verlag, Berlin (2001)
3. Luke, S.: Issues in Scaling Genetic Programming: Breeding Strategies, Tree Generation, and Code Bloat. PhD thesis, Department of Computer Science, University of Maryland, A. V. Williams Building, University of Maryland, College Park, MD 20742 USA (2000)

Humanoid Robot Programming Based on CBR Augmented GP

Hongwei Liu[1,2] and Hitoshi Iba[1]

[1] Graduate School of Frontier Science, The University of Tokyo,
Hongo 7-3-1, Bunkyo-ku, Tokyo 113-8656, Japan
{Lhw, Iba}@iba.k.u-tokyo.ac.jp
[2] School of Computer and Information, Hefei University of Technology,
Hefei 230009 China

1 Introduction

Humanoid robots are designed as companions for human beings to operate autonomously in various environments with people, and they need to adapt to noisy, cluttered environments. In order not only to look like but also to behave like human beings, humanoid robots need a vastly richer set of primitive behaviors; thus they must be able to produce dynamically temporal sequences of behaviors to accomplish a task in various human environments. As a consequence, it is difficult to evolve control programs for a humanoid robot.

In this paper, as an improvement of our previous researches [3], we introduced adaptive and developmental mechanisms by augmenting Genetic Programming [2] with Case-Based Reasoning [1], thus endowing humanoid robots with online adaptive and developmental abilities. Experimental results show that this approach can generate robust control programs; although we use a highly simplified simulation, which is rather crude, the robot can easily overcome the gaps between simulation and real world environments. Furthermore, the robot can develop new strategies according to the properties of new environments which it never encountered in simulation.

2 Experiments and Results

We employed humanoid robot HOAP-1 of Fujitsu Inc. The HOAP-1 robot was designed for a wide application in research and the development of robotic technologies. It is 48cm high, 6kg, has 20 DOFs, and is a light and compact humanoid robot.

In figure 1, the left panel traces one typical trajectory of a simulated robot, in which the circle *O* denotes the object and the circle *G* denotes the goal. From the trajectory, we can see that in simulation, the robot is able to select a rational path to approach the object while avoiding obstacles. The right panel shows the curve of average fitness (thin line) and of best fitness (thick line).

Figure 2 demonstrates one step of an adaptation process. The robot received an instruction of "ML" (move left forward) at that position. After interpreting this abstract behavior into a concrete behavior using CBR, the robot retrieved

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 708–709, 2004.

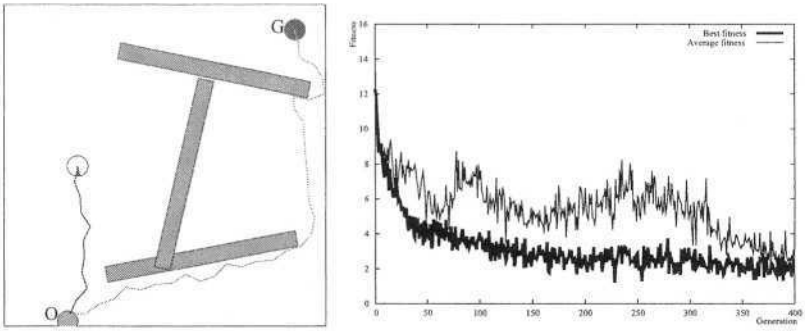


Fig. 1. The result of evolution stage. The left panel shows one example of the simulated robot's trajectory; and the right panel shows the curves of best fitness (thick line) and average fitness (thin line) over generations.

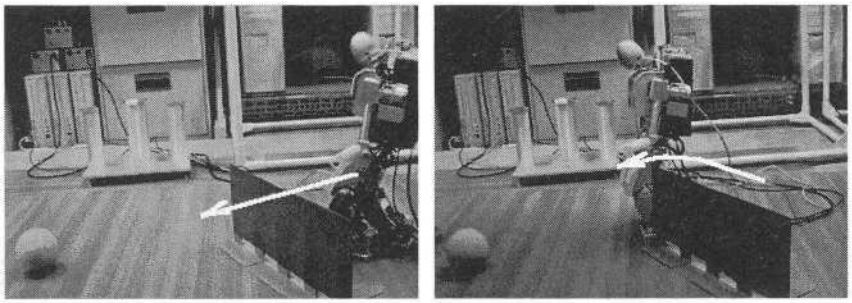


Fig. 2. In the real world environment, the humanoid robot shows adaptability by retrieving and reusing a corresponding case.

a similar case, which was to move forward slightly then turn left and move to left forward. By reusing this case, the robot shows flexibility when it performs in the real world.

Although we used highly simplified simulation, which is only a coarse mimic of real environments, by employing these two mechanisms, the humanoid robot can overcome the gaps between the simulation and the real world and flexibly adapt to real world environments.

References

1. Janet Kolodner. *Case-based reasoning.* Morgan Kaufmann Publishers Inc., 1993.
2. J. R. Koza. Evolution of subsumption using genetic programming. In F. J. Varela and P. Bourgine, editors, *Proceedings of the First European Conference on Artificial Life. Towards a Practice of Autonomous Systems,* pages 110–119, Paris, France, 11-13 Dec 1992. MIT Press.
3. Hongwei Liu and Hitoshi Iba. Multi-agent learning of heterogeneous robots by evolutionary subsumption. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2003),* pages 1715–1728, Chicago, IL, USA, 2003.

Genetic Network Programming with Reinforcement Learning and Its Performance Evaluation

Shingo Mabu, Kotaro Hirasawa, and Jinglu Hu

Graduate School of Information, Production and Systems, Waseda University,
Hibikino 2-7, Wakamatsu-ku, kitakyushu, Fukuoka, Japan,
mabu@asagi.waseda.jp, {hirasawa, jinglu}@waseda.jp

Abstract. A new graph-based evolutionary algorithm named "Genetic Network Programming, GNP" has been proposed. GNP represents its solutions as directed graph structures, which can improve the expression ability and performance. Since GA, GP and GNP already proposed are based on evolution and they cannot change their solutions until one generation ends, we propose GNP with Reinforcement Learning (GNP with RL) in this paper in order to search solutions quickly. Evolutionary algorithm of GNP makes very compact directed graph structure which contributes to reducing the size of the Q-table and saving memory. Reinforcement Learning of GNP improves search speed for solutions because it can use the information obtained during tasks.

Recently, a new directed graph-based evolutionary algorithm named "Genetic Network Programming (GNP)[1]" has been proposed. Basically, GA and GP represent solutions as string and tree structures, respectively, but GNP represents its solutions as graph structures composed of a number of judgement nodes and processing nodes [Fig. 1]. Judgement nodes are *if-then* type branch decision functions, and Processing node determines an action/processing an agent should do. The GNP we used never causes bloat because of the predefined number of nodes although GNP can evolve the programs having variable sizes. The graph structure of GNP has some distinguished abilities inherently. 1) Since the graph structure of GNP inherently have the ability to re-use nodes unlike GA and tree GP, GNP can use certain Judgement/Processing nodes repeatedly to achieve tasks. Therefore, even if the number of nodes is predefined and smaller than GP programs, GNP can perform well by making effective programs based on re-using nodes. So we do not have to prepare the excessive number of nodes, as a result, the structures of GNP become very compact, which contributes toward saving the calculation time and physical memory. 2) GNP starts its node transition from a start node and continues its node transition according to the node connections without any terminal. So a current node is selected influenced by the past node transition. Therefore, GNP can have the implicit memory function which memorizes the past action sequences of agents in the network flow. The node transition ends when the end condition is satisfied, for example, the time step reaches the maximum one or the GNP program completes the given tasks.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 710–711, 2004.

However, conventional GNP are based on evolution, i.e., after the programs are carried out to some extent, they are evaluated and evolved based on their fitness values, so many trials must be done again and again. To overcome this problem and search solutions quickly, we have proposed a new algorithm of GNP[2] which combines evolution and reinforcement learning (RL). Since RL is done when agents carry out their task, GNP can search for better solutions every judgement/processing besides the evolution executed every generation. The aim of the combination of RL and evolution is to take advantage of the sophisticated search abilities of evolution and online learning of RL.

In this paper, a new algorithm of GNP with Reinforcement Learning is proposed. This method is a extension of the previous GNP, so it becomes more general framework of GNP with RL in terms that 1) we defined the new state and action pairs used by RL which are different from the previous method, 2) the proposed method can change node functions in addition to node connections and 3) save the number of parameters used by RL, so the size of the Q-table becomes small.

In order to confirm the ability of the proposed method, the simulations using tileworld and maze problem which are the typical benchmark problems of agents are carried out, and the results of the proposed method are compared with those of standard GNP (GNP using evolution only) and standard tree GP. From the simulation results, it is clarified that the proposed method can obtain the best results and learn faster than the previous algorithm [Fig. 2, 3].

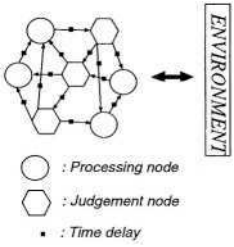


Fig. 1. Basic structure of GNP

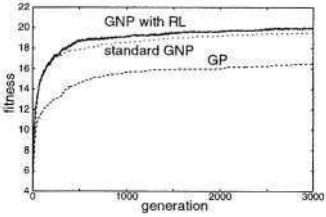


Fig. 2. Tileworld problem

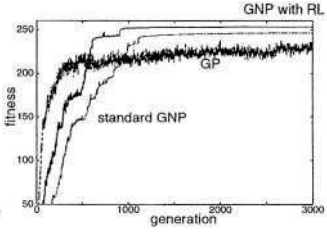


Fig. 3. Maze problem

References

1. H.Katagiri, K.Hirasawa, J.Hu and J.Murata, "Network structure Oriented Evolutionary Model - Genetic Network Programming - and Its comparison with Genetic Programming", *in 2001 Genetic and Evolutionary Computation Conference, Late Breaking Papers,* pp. 219-226, (2001).
2. S. Mabu, K. Hirasawa, and J. Hu, "Genetic Network Programming with Learning and Evolution for Adapting to Dynamical Environments", *in Proc. of 2003 Congress on Evolutionary Computation,* pp. 69-76, (2003).

Multi-agent Cooperation Using Genetic Network Programming with Automatically Defined Groups

Tadahiko Murata and Takashi Nakamura

Department of Informatics, Kansai University, Takatsuki, 569-1095, Osaka, Japan
murata@res.kutc.kansai-u.ac.jp,
http://www.res.kutc.kansai-u.ac.jp/~murata/

Abstract. In this paper, we propose a genetic network programming (GNP) architecture using a coevolution model called automatically defined groups (ADG). The GNP evolves networks for describing condition-action relations for agents. By applying ADG to GNP, we evolve different networks in order to realize the cooperation of multiple agents with different abilities. Computational experiments on a load transportation problem show that appropriate networks are obtained with taking account of the ability of agents.

1 GNP with ADG

Genetic Network Programming (GNP) has been proposed [1] for describing successions of processing. It is inspired from GP, however it does not have a tree architecture, but has a network architecture. Fig. 1 shows an example of networks obtained by the GNP. There are three types of nodes in a network: a start node, judgment nodes (diamonds), and processing nodes (circles). Agents make decisions on their behaviors according to the obtained network. For example, an agent examines whether it carries a load or not in the first judgment node "Carry?" If it is carrying a load, it moves to a goal according to the processing node "Move to goal." Otherwise, it will go to a place with heavy loads. Using network architecture GNP can obtain a succession of actions as shown in Fig. 1 while GP realizes a succession of actions by returning to a root node from a terminal node in which an agent can take an action.

When each agent has a different ability, a different network should be obtained according to the ability of the agent. We employ a coevolution model called automatically defined groups (ADG) [2]. While the model was originally proposed for GP, we apply it to GNP in order to develop different networks according to the ability of agents. We show the effectiveness of the GNP with ADG by computational experiments on a load transportation problem [2].

2 Simulations on Load Transportation Problem

We applied a GP, a GNP, a GNP with a heterogeneous model (GNP-H) and the proposed method, that is a GNP with ADG (GNP-A) to a load transportation problem.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 712–714, 2004.

In the GNP-H, each agent has its own network while the original GNP has only a network for all agents. In the load transportation problem, there are light and heavy loads in a two-dimensional grid world. The number of agents is 20. Among them only five agents can bring a heavy or light load. The other 15 agents can carry only a light load at a time. When an agent brings a heavy load, five points are obtained. Bringing a light load results in one point. The task of the problem is to maximize the total point within a time limit. Since we set a time limit for each agent to bring a load to the goal three times, the best total point becomes 120. Appropriate action rules for each of the 20 agents should be obtained according to their carrying ability. We employ the total point as a fitness value to be maximized in the evolutionary process.

Table 1 shows average results of computational experiments over 100 times. From Table 1, we can see that the GNP and the GNP-A can attain the best fitness value while the other two methods can not do. Fig. 1 shows the best network obtained by the GNP. The fitness value becomes 120 when each of the 20 agents acts according to this network. Fig. 2 depicts the maximum fitness values over generations obtained by the four methods. The GP could find a tree which enables the five agents to carry a heavy load three times. Since every network should be optimized in the GNP-H, it can not converge during 500 generations. Table 1 and Fig. 2 show that the proposed method (GNP-A) could obtain networks with the best fitness over all 100 trials and converge faster than the GNP. The proposed method shows better performance on the load transportation problem because it obtained two simpler networks for the agents.

Table 1. Average fitness values.

Fitness	GP	GNP	GNP-H	GNP-A
Max	75	120	118	120
Average	75.0	105.8	110.9	120.0
Min	75	75	99	120

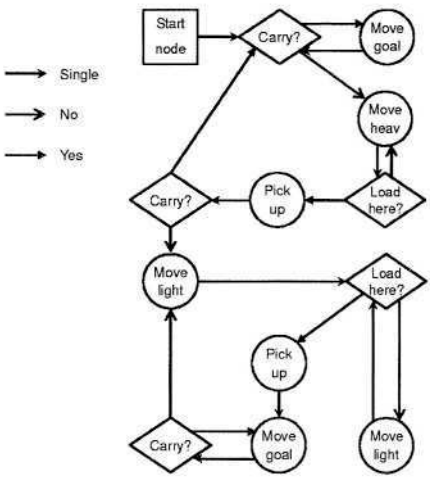


Fig. 1. The best network obtained by GNP.

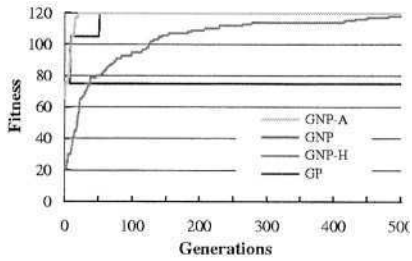


Fig. 2. Average results over generations.

References

1. H. Katagiri, K. Hirasawa, J. Hu, and J. Murata, "Network structure oriented evolutionary model -Genetic Network Programming- and its comparison with Genetic Programming," *Proc. of 2001 GECCO,* p. 219 (2001).
2. A. Hara and T. Nagao, "Emergence of cooperative behavior using ADG; Automatically defined groups," *Proc. of 1999 GECCO,* pp. 1039-1046 (1999).

Chemical Genetic Programming – Coevolution Between Genotypic Strings and Phenotypic Trees

Wojciech Piaseczny[1], Hideaki Suzuki[1], and Hidefumi Sawai[2]

[1] ATR Human Information Science Labs., Kyoto, Japan
{wojtek,hsuzuki}@atr.jp,
[2] Communications Research Laboratory, Kobe, Japan
sawai@crl.go.jp

Abstract. Chemical Genetic Programming (CGP) is a new method of genetic programming that introduced collision-based biochemical processes and realized dynamic mapping from genotypic strings to phenotypic trees.

Chemical Genetic Programming (CGP) [1,2] proposes a new method of genetic programming that enables evolutionary optimization of the mapping from genotypic strings to phenotypic trees. The phenotypic tree of an individual is created by rewriting an initial expression using a series of grammatical rewriting rules. The information stored on DNA serves two purposes. First, it determines which rewriting rules to apply in building an individual's phenotypic tree. Secondly, the DNA is used to create this set of rewriting rules. CGP uses a collision-based method for translating DNA into the set of rewriting rules. Fig. 1 shows the structure of a cell in CGP.

An initial set of rewriting rules is supplied that provides the system with the syntactical structure within which all programs must be generated. These rules allow only executable programs to be generated, but do not place any artificial limitations on the complexity or structure of these programs. Individuals are initially randomly generated, then evolved to form the population of successive generations. The fitness value of an individual is calculated using a correlation function that compares the target function with the individual's generated function. Based on their fitness values, cells, including both the DNA and the rewriting rules, are selected and then evolved. Evolution consists of mutation of DNA units, single point DNA crossover and molecular exchange between two cells, and selection of cells. Evolution adaptively changes not only the DNA information, but also the rewriting rules in cells. This enables the evolutionary optimization of the genotype (DNA) to phenotype (tree) mapping, by which we expect CGP to find the best translation grammar for creating a solution from the one-dimensional genotypic information.

To examine the effectiveness of CGP, it was applied to a selected symbolic regression problem, and the results were compared against those of Grammatical Evolution (GE) [3], which uses static translation from genotype to phenotype.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 715–716, 2004.

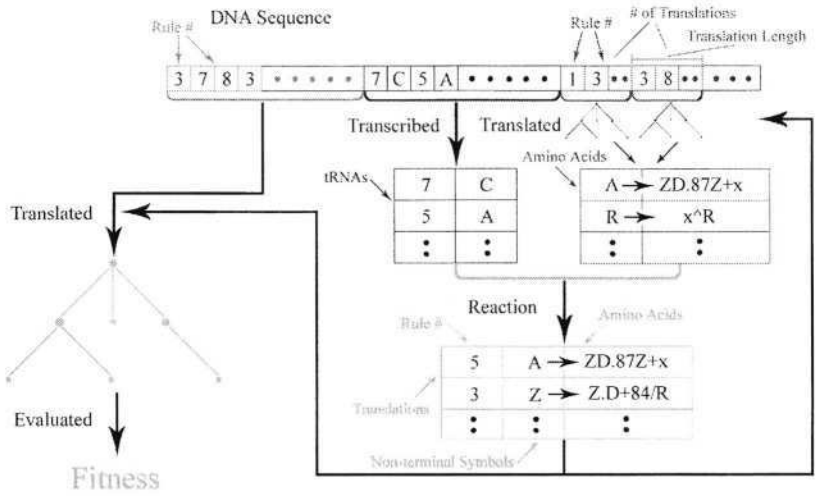


Fig. 1. A cell's architecture in chemical genetic programming. The DNA sequence is translated into amino acids, and transcribed into tRNAs. Amino acids and tRNAs react together using a collision-based method to form an individual's rewriting rules. This revised grammar set is used for translation of the DNA to form the final evaluation tree.

The results obtained to date have shown CGP to be effective in this problem domain, as 8 of 10 trials were able to find a good solution while only 2 of 10 GE trials were able to find a good solution.

This research was supported by the Telecommunications Advancement Organization of Japan. The second author's work was also partly supported by Doshisha University's Research Promotion Funds.

References

1. Piaseczny, W., Suzuki, H., Sawai, H.: Chemical genetic programming - Evolutionary optimization of the translation from genotypic strings to phenotypic trees. In: Sugisaka, M., Tanaka, H. (eds.): *Proceedings of the Ninth International Symposium on Artificial Life and Robotics (AROB 9th '04)* Vol. 2 (2004) 571-574
2. Piaseczny, W., Sawai, H., Suzuki, H.: Chemical genetic programming - Evolution of amino acid rewriting rules used for genotype-phenotype translation. In: *Proceedings of the 2004 Congress on Evolutionary Computation (CEC-2004)* To be published.
3. Ryan, C., O'Neill, M., Collins, J.: Grammatical Evolution: Solving Trigonometric Identities. In: *Proceedings of the 4th International Mendel Conference on Genetic Algorithms, Optimization Problems, Fuzzy Logic, Neural Networks, Rough Sets (MENDEL)* (1998)

A Study of the Role of Single Node Mutation in Genetic Programming*

Wei Quan and Terence Soule

Department of Computer Science
University of Idaho
Moscow, ID 83844-1010
tsoule@cs.uidaho.edu

Abstract. In this paper we examine the effects of single node mutations on trees evolved via genetic programming. The results show that neutral mutations are less likely for nodes nearer the root and that as evolution proceeds neutral mutations of nodes near the root are progressively less likely.

Studies of crossover in tree based GP have shown that when smaller and/or deeper branches are selected for crossover the resulting change in fitness is smaller and the probability of fitness neutral crossover is larger [3,1,4,2]. In this paper we continue this research for mutations by studying the relationship between the depth of single node mutations and the probability of fitness neutral mutations.

Our GP is steady-state, population size 100, 0.7 crossover rate, 90/10 crossover, 4 member tournament selection. Results are the average of 100 trials. The initial population is generated with full tree with depth of 4. The test problem is symbolic regression; the target function is $f(x) = x^3 + 2x^2 - 5x + 3$ in the range $[0, 2\pi]$ or $[-\pi, \pi]$. Fitness is the square root of the sum of the squared errors at every test point. The function and terminal set is $\{+, -, *, /(\text{protected})\}$. Mutation changes a single node into another node with the same arity. One mutation is applied per offspring per iteration.

Results are generated by copying the population after 50, 100 and 500 evaluations and exhaustively testing every possible single node mutation on the copied populations. Figure 1 shows the percentage of non-neutral mutations as a function of depth of mutation and the number of evaluations/iterations. There is a strong correlation between the depth of a mutation point and the probability of a fitness neutral mutation. Additionally, after a longer period of evolution neutral mutations are less likely, for all mutation depths studied.

We tested two hypotheses to explain the affect of additional iterations on the percentage of non-neutral mutations:
1) As evolution proceeds, the increasing tree size makes mutation near the root less likely to be fitness neutral. This seems reasonable because for larger trees a mutation near the root effects absolutely and proportionally more of the tree.
2) As evolution proceeds, the improving fitness makes mutation near the root

* This work supported by NSF EPSCoR EPS-0132626.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 717–718, 2004.

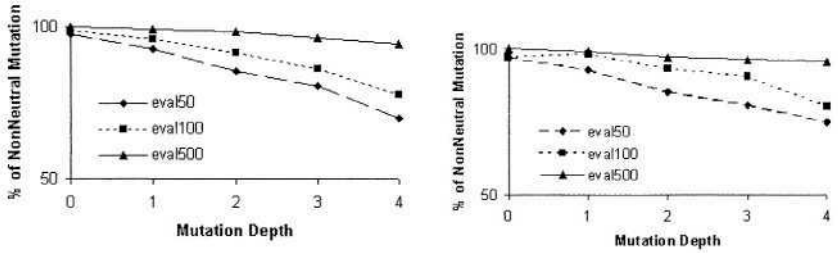


Fig. 1. Average percentage of mutations that are non-neutral as a function of depth after 50, 100 and 500 evaluations for the domains $[-\pi, \pi](\text{left})$ and $[0, 2\pi](\text{right})$. Mutations at the root (depth 0) are never neutral. As depth increases neutral mutations become more likely. As evolution proceeds neutral mutations become progressively less likely.

less likely to be fitness neutral. This seems reasonable because as trees become more fit they are presumably less random, thus a random mutation may be more likely to effect their fitness.

To test these hypotheses we compared overall tree depth to the probability of neutral mutations and compared the tree fitness to the probability of neutral mutations. The correlations of both overall depth and of fitness to the probability of neutral mutations were too weak to support the above hypotheses.

Our results show that the affects of single node mutations are similar to the affects of crossover in tree based GP: the probability of a neutral mutation is correlated with the depth of the mutation, at least for mutations near the root. Additionally, the more a program has evolved, the more likely it is that a mutation near the root will change the tree's fitness.

References

1. Christian Igel and Kumar Chellapilla. Investigating the influence of depth and degree of genotypic change on fitness in genetic programming. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference 1999*, pages 1061–1068. Morgan Kaufmann, 1999.
2. Sean Luke. Code growth is not caused by introns. In *Late Breaking Papers, Proceedings of the Genetic and Evolutionary Computation Conference 2000,* pages 228–235, 2000.
3. Terence Soule and Robert Heckendorn. An analysis of the causes of code growth in genetic programmin. *Genetic Programming and Evolvable Machines,* 3:283–309, 2002.
4. Matthew J. Streeter. The root causes of code growth in genetic programming. In Conor Ryan, Terence Soule, Maarten Keijzer, Edward Tsang, Riccardo Poli, and Ernesto Costa, editors, *Genetic Programming, 6th European Conference, EuroGP 2003*, pages 443–454, 2002.

Multi-branches Genetic Programming as a Tool for Function Approximation

Katya Rodríguez-Vázquez and Carlos Oliver-Morales

IIMAS-UNAM, Circuito Escolar, CU, 04510 Coyoacón, Mexico City, MEXICO
katya@uxdea4.iimas.unam.mx, oliver_carlos_99@yahoo.com

Abstract. This work presents a performance analysis of a Multi-Branches Genetic Programming (MBGP) approach applied in symbolic regression (e.g. function approximation) problems. Genetic Programming (GP) has been previously applied to this kind of regression. However, one of the main drawbacks of GP is the fact that individuals tend to grow in size through the evolution process without a significant improvement in individual performance. In Multi-Branches Genetic Programming (MBGP), an individual is composed of several branches, each branch can evolve a part of individual solution, and final solution is composed of the integration of these partial solutions.

1 Introduction

In function approximation problems based on GP approaches, two relevant aspects are considered. On the one hand, evolved functions must be accurate approximations; on the other hand, solutions complexity must be also kept as simple as possible without incurring in a deterioration in accuracy. This paper presents an alternative GP encoding based on multiple branches, showing the evolution of accurate and simple solution.

The multi-branches (MB) representation work consists of four parts: a root node, *N* branches, *N+1* coefficients and an output. The number of coefficients is *N+1,* a coefficient for each branch plus the constant term. Expression of section 2 provides details of multi-branches representation. In this example, rooted-node has been defined as the addition operation (ADD). The first branch only consists of $1/n^2$, while the second branch is defined as log(2n). Coefficients for each branch are also expressed. The last coefficient refers to constant term into polynomial expression.

2 Experiments

The harmonic number defined in equation (1) is used in order to approximate the well-known function expansion given in equation (2), an asymptotic expansion where γ is the Euler's constant ($\gamma \approx 0.57722$).

$$H_n \equiv \sum_{i=1}^{n} \frac{1}{i} \tag{1}$$

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 719–721, 2004.

$$H_n = \gamma + \ln(n) + \frac{1}{2n} - \frac{1}{12n^2} + \frac{1}{120n^4} + O(\frac{1}{n^6}) \quad (2)$$

where n = 1, 2, ..., 50 (the first 50 harmonic numbers defined as fitness cases). The function set used in this problem is F = { +, -, *, %, -x, plog, psqrt, cos}, where %, plog and psqrt are the protected division, the protected natural logarithm and the protected square root, respectively. Then, the aim of this experiment is to rediscover terms of an already known approximation. In this case, the following approximation emerged,

```
(ADD
   (divd (divd n n) (.* n n))
   (log_p (+ n n))
   (divd (ngt n) (.* n n))
   -0.072444
   2.302
   -0.49414
   -0.1147)
)
```

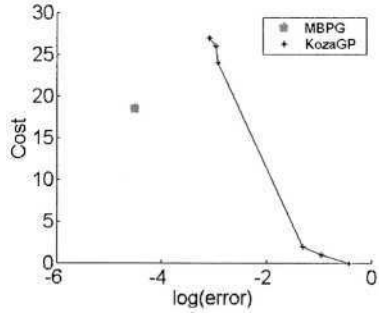


Fig. 1. Harmonic number, [1, 50].

Translating this MBGP model, the function expansion is expressed as,

$$H_n = -0.072444\frac{1}{n^2} + 2.302\ln(2n) - 0.49414\frac{-n}{n^2} - 0.1147$$

ordering this equation,

$$H_n = [\ln(2) - 0.1147] + \ln(n) + \frac{1}{2.0237n} - \frac{1}{13.8n^2} + 1.302\ln(2n)$$

$$= 0.5784 + \ln(n) + \frac{1}{2.0237n} - \frac{1}{13.8n^2} + 1.302\ln(2n)$$

It is observed that MBGP approximated the first four terms given in equation (3), showing an approximation error of 3.18742×10^{-5}. Figure 1 shows the Padé approximations and GP Pareto front (Streeter and Becker, 2003) as well as the solution with smallest error generated by means of MBGP.

3 Conclusions

The multi-branches representation for genetic programming was proved to be powerful. It has been tested on a function approximation problem and results showed to be promising. It was also observed that complexity tends to be reduced by using this representation. Further studies will focus on both the flexibility of this representation in diverse domains and the effects and control of introns.

Acknowledgements. Authors would like to thank the financial support of CONACyT, México, under the project 40602-A and PAPPIT-UNAM under the project EN100201.

Reference

STREETER M. AND L.A. BECKER (2003) Automated Discovery of Numerical Approximation Formulae via Genetic Programming. *Genetic Programming and Evolvable Machines,* **4**(3), pp. 255-286.

Hierarchical Breeding Control for Efficient Topology/Parameter Evolution

Kisung Seo[1,2], Jianjun Hu[1], Zhun Fan[1], Erik D. Goodman[1], and Ronald C. Rosenberg[1]

[1]Genetic Algorithms Research and Applications Group (GARAGe), Michigan State University, East Lansing, MI 48824, USA
[2]Department of Electronics Engineering, Seokyeong University
Seoul, 136-749, Korea
ksseo@skuniv.ac.kr,
{hujianju, fanzhun, goodman, rosenber}@egr.msu.edu

1 The Approach

This paper adopts a hierarchical breeding control mechanism to obtain better search performance based on differential balancing of topology-altering operations and parameter-altering operations according to fitness level, in a fitness-structured multi-population model. The basic idea for this control mechanism arises from observing the human design process. Usually, preliminary or conceptual design involves more structural modification, and final or detailed design involves more parameter tuning – i.e., there is greater concentration on design topology in the early stage and more on parameter tuning in the later stage. Therefore, the key concept is to provide different breeding probabilities for topology-altering and parameter-altering operations according to fitness level of the subpopulation (Figure 1).

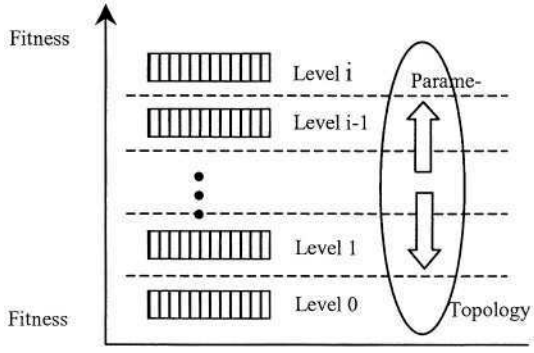


Fig. 1. Hierarchical breeding control structure. Subpopulations are organized in a hierarchy with ascending fitness levels.

In other words, topology-altering operations are given higher probability than parameter-altering operations at low fitness levels, and vice-versa at higher levels. To implement, switched modular primitives, which separate GP functions into a three-level hierarchy – operations affecting topology, intermediate topology, and parameters

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 722–723, 2004.

– are combined with topology/parameter control in Hierarchical Fair Competition GP (HFC-GP), which can strongly reduce premature convergence and enable scalability with smaller populations.

2 Results of the Approach

As a proof of concept for this approach, the eigenvalue assignment problem, which is to synthesize bond graph models with minimum distance errors from pre-specified target sets of eigenvalues, was used. The fitness function calculates the sum of distance errors between each target eigenvalue and the solution's corresponding eigenvalue, divides by the order, and performs hyperbolic scaling.

The results of 6- and 10-eigenvalue runs are provided in Figure 2, showing average distance error for each set across 10 experiments. Figure 2, left, illustrates the comparison between the basic approach (without topology/parameter control) and the hierarchical topology/parameter breeding control on typical complex conjugate and real 6-eigenvalue target sets. For all four cases, the average error in the hierarchical topology/parameter breeding control approach is smaller than that of the basic approach. The right side of Figure 2 represents the results on two 10-eigenvalue sets, and shows that the new approach outperforms the basic approach on these problems.

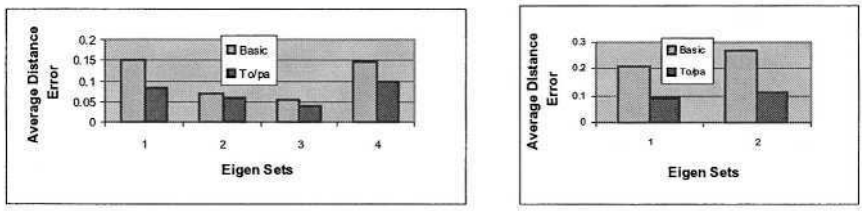


Fig. 2. Results for 6-eigenvalue sets (left, four cases) and 10-eigenvalue sets (right, two cases)

Results showed better performance for all tested eigenvalue sets when the new topology/parameter control method was used. This tends to support the conjecture that a carefully tailored representation and sophisticated topology/parameter control method will improve the efficiency of GP search. This, in turn, offers promise that much more complex multi-domain systems with more complex performance specifications can be designed efficiently.

Acknowledgment. The authors gratefully acknowledge the support of the National Science Foundation through grant DMII 0084934.

Keeping the Diversity with Small Populations Using Logic-Based Genetic Programming

Ken Taniguchi and Takao Terano

Graduate School of Business Sciences, University of Tsukuba,
3-29-1, Otsuka, Bunkyo-ku, Tokyo 112-0012, Japan
{taniguti, terano}@gssm.otsuka.tsukuba.ac.jp
http://www.gssm.otsuka.tsukuba.ac.jp/staff/terano/

Abstract. We present a new method of Logic-Based Genetic Programming (LBGP). Using the intrinsic mechanism of backtracking in Prolog, we utilize large individual programs with redundant clauses, and apply them to small populations. Our method is validated by the experiments about Symbolic Regression, and XOR problems.

Genetic programming is a method to automate programming tasks in which programs are evolved through Genetic Algorithm (GA) techniques. One of the difficulties of GP [2] is that I) they must have a large number of individuals in order to keep the population diversity, and that II) they cannot reuse sub-structures of program codes destroyed by improper uses of genetic operators. To cope with the issues, we will focus on the Prolog-based logic programming framework.

In this paper, we propose a new Logic-Based Genetic Programming (LBGP) and show the effectiveness via computer experiments. In our research, we follow the ideas: (1) Each program as an individual consists of a set of clauses, which are redundant like the representation of individuals in Messy GAs [1] and we allow them to execute only some parts of clauses, and (2) Instead of using large populations, flexible genetic operations are designed to exchange the roles of clause sets, which are directly executable or not in the run.

In our LBGP, we directly handle tree structures of program codes. We use Prolog language to represent an individual as a set of clauses. The order of clauses corresponds to the locus of chromosomes. Multiple clauses with same heads are executed in the ascending order similar.

We define the two concepts: exon and intron parts, which are similar to the ones in the literature [4]. The exon part is a clause directly executed and evaluated by means of a given fitness function. Exon parts are usually the most top clauses among the same heads. The introns part is the other clauses, however, by the backtracking mechanism, they will become candidates for the execution.

The exonizing operation stochastically activates the intron or inactive parts generated by sub-tree crossover or intronizing operations. By the operation, a selected Horn clause is replaced to the most upper line from the other places. The restructured sub-

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 724–725, 2004.

trees are expected to be effective in the later phases of genetic cycles, because they may contain indirectly executable and potentially usable components.

The procedure of our LBGP is summarized as follows:

Step 1: Initialize
Generate the initial population with random trees
Step 2: Selection of Reproduction
Select two parents randomly from the populations.
Step 3: Genetic Operation
Stochastically apply the one of the following genetic operations:
3-1) Sub-tree Crossover;
3-2) One Point Crossover;
3-3) Mutation Operation
3-4) Activate/Inactivate Operation:
3-4-1) If there are Intron Parts, Apply Exonizing Operation;
3-4-2) Ohterwise, Apply Intronizing Operation to the Elitist Individual.
Step 4: Selection of Survival
Apply Generation Model for the survival.
Step 5: Loop
If the terminal conditions are satisfied, then stop. Else go to Step 1.

We have carried out experiments about Symbolic Regression and XOR problems to validate the performance by means of program execution. Also, we have compared our implementation with the conventional LBGP code from CMU AI Repository [3]. The results have suggested the proposed method outperform ten times by means of the number of program evaluation.

The performance of the method are attained by (1) variable length chromosome representation of codes in Prolog programs, (2) employments of exonizing and intronizing operations, (3) integration of several generation models, and (4) introduction of the stack counter. Our future work includes the further integration of LBGP with multi-modal and multi-objective problems, the validation of applicability to large scale practical problems.

References

1. Goldberg, D. E., Korb, B., and Deb, K.: Messy Genetic Algorithms: Motivation, Analysis, and First Results, *Complex Systems,* Vol. 3, pp. 493-530 (1989)
2. Koza, J. R.: *Genetic Programming: On the Programming of Computers by Means of Natural Selection,* MIT Press (1992)
3. Raik, S.: Genetic Programming in Prolog, CMU-AI-Reposiroty (1993)
4. Terao, M. and Iba, H.: Controlling Effective Introns for Multi-Agent Learning by Genetic Programming, *Proc. of GECCO 2000,* pp. 419-426 (2000)

Analysis and Improvements of the Adaptive Discretization Intervals Knowledge Representation

Jaume Bacardit and Josep Maria Garrell

Intelligent Systems Research Group, Universitat Ramon Llull,
Psg. Bonanova 8, 08022-Barcelona, Catalonia, Spain, Europe.
{jbacardit,josepmg}@salleURL.edu

Abstract. In order to handle classification problems with real-valued attributes using discretization algorithms it is necessary to obtain a good and reduced set of cut points in order to learn successfully. In recent years a discretization-based knowledge representation called *Adaptive Discretization Intervals* has been developed that can use several discretizers at the same time and also combines adjacent cut points. In this paper we analyze its behavior in several aspects. From this analysis we propose some fixes and new operators that manage to improve the performance of the representation across a large set of domains.

1 Introduction

Genetic Algorithms (GA) [1] have been applied extensively in recent years to Classification and Machine Learning tasks [2,3,4,5]. The most popular family of systems performing these tasks is known as Learning Classifier Systems (LCS), having two main approaches: the Pittsburgh LCS [6] and the Michigan LCS [2]. The classical knowledge representations [6,2] used in these systems are nominal. Therefore, a discretization algorithm is needed in order to handle problems with real-valued attributes with these representations, treating the resulting intervals as nominal values. The performance of these systems is tied to the right election of these intervals. A good discretization algorithm has to balance the loss of information intrinsic to this kind of process and generating a reasonable number of cut points, that is, a reasonable search space.

In recent years a discretization-based knowledge representation has been developed which can handle these issues, called *Adaptive Discretization Intervals (ADI) rule representation* [7]. This representation is used inside a Pittsburgh *LCS* and uses rules that contain intervals built joining together the low level intervals provided by the discretization algorithm, thus collapsing the search space when it is possible. Also, this representation can use several discretization algorithms at the same time allowing the system to choose the correct discretization for each problem and attribute.

In this paper we analyze the behavior of this representation from various points of view. This analysis identifies some problems that lead us to propose a

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 726–738, 2004.

new recombination operator and also a fix to another one. These improvements increase the ADI representation performance in most domains (even significantly sometimes based on statistical tests). Several combinations of the improvements proposed in this paper are tested, comparing them to the original configuration and also to two Machine Learning systems across several domains.

The paper is structured as follows. Section 2 describes the framework of the classifier system used for this paper and the *ADI* knowledge representation. Then, the ADI representation analysis and proposal of new operators is explained in section 3. Next, section 4 describes the test suite used in the comparison. The results obtained are summarized in section 5, section 6 discusses the conclusions and some further work and, finally, section 7 presents some related work.

2 Framework and ADI Knowledge Representation

The *LCS* used for this paper is called GAssist *(Genetic Algorithms based claSSIfier sySTem)* [7], and is a Pittsburgh style classifier system descendant on *GABIL*. A detailed description of the system is found in a previous paper [8].

2.1 Adaptive Discretization Intervals (ADI) Knowledge Representation

This subsection describes the main characteristics of the *ADI* knowledge representation. For this paper we have used the second revision of the representation (named *ADI2* in previous work [7]).

The semantical structure of each rule in ADI is taken from **GABIL** [6]: Each rule consists of a condition part and a classification part: $condition \rightarrow decision$. Each condition is a Conjunctive Normal Form (CNF) predicate defined as:

$$((A_1 = V_1^1 \vee \ldots \vee A_1 = V_m^1) \bigwedge \ldots \bigwedge (A_n = V_2^n \vee \ldots A_n = V_m^b))$$

Where A_i is the ith attribute of the problem and V_i^j is the jth value that can take the ith attribute. This kind of predicate can be encoded into a binary string in the following way: if we have a problem with two attributes whose values can be {1,2,3}, a rule of the form "If the first attribute has value 1 or 2 and the second one has value 3 then we assign class 1" will be represented by the string 110|001|1.

In *GABIL* for each attribute we would use a set of static discretization intervals instead of nominal values. The intervals of the ADI representation are not static, but they evolve through the iterations splitting and merging among them (having a minimum size called *micro-interval*). Thus, the binary coding of the *GABIL* representation is extended as represented in figure 1, also showing the split and merge operations.

The inner workings of the representation and the exact definition of the operators can be read in [7].

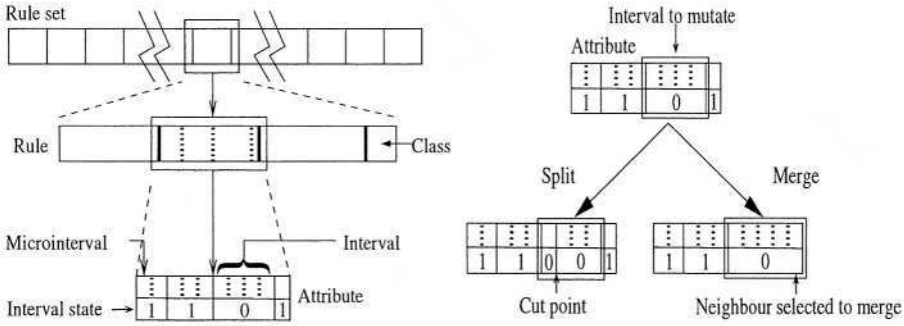


Fig. 1. Adaptive intervals representation and the split and merge operators.

3 Analysis and Improvements of the ADI Knowledge Representation

In this section we analyze the behavior of the *ADI* representation presented in previous papers and, after, we propose some fixes and a new operator in order to fix some identified flaws.

3.1 Discretizers in the Population. Finding the Ideal Discretizer

The first issue we want to examine is the distribution of discretizers in the population. In the ADI representation, the initialization stage assigns a random discretizer from our predefined pool to each attribute term of each rule of each individual. Through the iterations, the discretizers of the best individuals survive, but no new discretizers are inserted into the population. This arises the question of how does the number of attributes in the population assigned to each discretizer evolve through the iterations. We extracted this information from the population and it is represented in figure 2. This figure show the evolution of the discretizer proportions for 4 problems (*bre,iris,mmg,pim,* detailed in section 4). The discretizers chosen for these tests are the most frequently used previously in the *ADI* research: uniform-width discretizer of 4,5,6,7,8,10,15,20,25 intervals. We show this figure to compare the behavior among the datasets. Therefore, the same Y scale is used in all plots.

Figure 2 shows that all discretizers start the evolutionary process with a proportion approximately of 1/number of discretizers. Later on, the proportions change through the iterations. We can see that the proportions for all the datasets do not diverge too much from their initial value, with the exception of the *iris* dataset. This behavior makes us wonder if the system has managed to identify the ideal discretizer for this dataset. Thus we repeated the tests for these four datasets but using only the discretizer most frequent for each problem. The results are detailed in table 1.

We can see that the only dataset where there is accuracy increase when we are using only one discretizer is *iris.* It would be interesting to determine if there

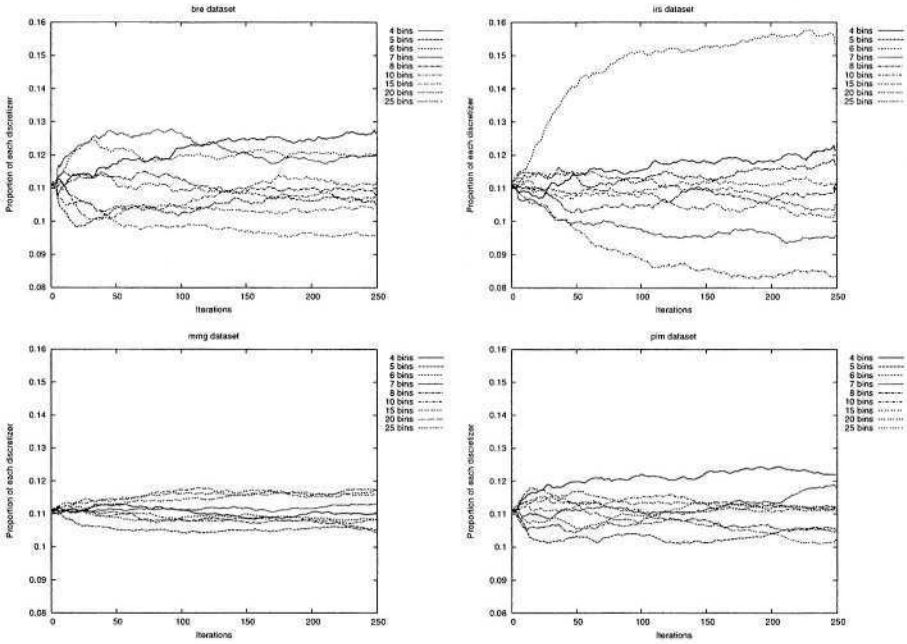


Fig. 2. Evolution of the discretizer proportions in the population for the *bre,iris,mmg,pim* datasets

Table 1. Results of the experiment of using only the discretizer with more proportion in the population

Dataset	Original accuracy	Accuracy with one discretizer	Discretizer
bre	95.6±2.2	95.5±1.8	4 intervals
irs	95.9±3.9	97.8±3.1	6 intervals
mmg	65.0±9.0	63.1±8.8	25 intervals
pim	74.4±4.7	74.2±3.7	4 intervals

are other datasets where the evolution of the discretizer proportions presents the same behavior, and check if they manage also to identify the ideal discretizer. Unfortunately, we could not find any more dataset presenting this behavior.

3.2 Discretizers in the Population. Survival of the Discretizers

The next issue to analyze of the *ADI* representation is also related to the discretizer proportions in the population. In figure 3 we show the evolution of the average proportion of the 15 intervals discretizer for the *bre* dataset, but this time using error bars. We can extract an important observation: In some runs this discretizer disappeared from the population in less than 30 iterations. Other discretizers and datasets show the same behavior. Is this effect good or bad? Ideally the *GA* should choose the ideal discretizer for each domain and attribute. However, in most situations the system is not prepared to choose correctly in

few iterations because it has not learned enough. It is clear that, in order to avoid this situation, we have to create some kind of mechanism that is able to introduce new discretizers into the population through the evolutionary process.

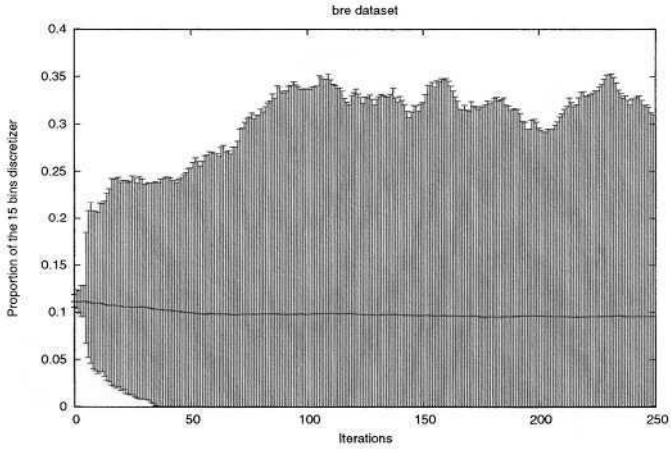


Fig. 3. Evolution of the proportions of the uniform-width discretizer with 15 intervals in the population for the bre dataset

What form does it take such survival mechanism? Probably the most suitable form would be an operator that changes the discretizer used by an attribute but maintaining, as much as possible, the semantical structure of the attribute. That is, finding a set of intervals build over the new discretizer as close as possible to the old ones. Unfortunately, an operator like this can present a huge computational cost considering that it is frequent to deal with datasets that have hundreds of cut points. Therefore, for this paper we have designed a simpler operator, called **reinitialize.** This operator repeats the process done in the initialization stage of the *GA,* but only for the selected individual and attribute term, as represented in figure 4.

1. We have an attribute term of a rule in the population selected for reinitialize
2. We select randomly a discretizer from our predefined pool
3. We assign a number of intervals to the attribute. This number is randomly chosen between 2 and min(number of *micro-intervals*, maximum allowed intervals per attribute)
4. We assign randomly a number of consecutive *micro-intervals* to each interval of the attribute. Every interval must have at least one *micro-interval*
5. We assign a random truth value (0 or 1) to each interval

Fig. 4. Steps of the reinitialize operator

This operator is applied after the merge and split stages, and the probability controlling it is also defined for each attribute-term. In order to assign a

good value to this probability we did some tests with some probability values (0.0025,0.005,0.01,0.015). We reproduce only the results for the *mmg* and *pim* datasets because they illustrate two different kinds of behavior. The results are in table 2, where we can see a correlation between the probability increase and the a decrease of obtained train accuracy and more rules and intervals per attribute. However, test accuracy does not show these trends. While the *pim* dataset does not benefit from this operator, the *mmg* dataset has a notable accuracy increase, considering that we are comparing two versions of the same system.

Table 2. Short tests of the reinitialize operator

Dataset	Reinit. prob.	Train acc.	Test acc.	# of rules	Interv. per attr
mmg	0.0000	78.4±1.8	65.0±9.0	6.6±1.0	2.4±0.1
	0.0025	78.4±1.8	65.7±8.8	6.5±1.0	2.5±0.1
	0.0050	78.2±1.7	66.8±8.8	6.5±0.8	2.5±0.1
	0.0100	77.5±1.7	67.3±9.5	6.5±0.9	2.6±0.1
	0.0150	76.4±1.8	67.5±9.1	6.6±1.0	2.7±0.1
pim	0.0000	78.6±1.0	74.4±4.7	5.4±0.9	2.2±0.1
	0.0025	78.1±0.9	74.4±4.5	5.1±0.6	2.2±0.1
	0.0050	78.1±1.0	74.4±4.7	5.3±0.7	2.3±0.1
	0.0100	77.9±1.0	74.3±4.3	5.2±0.8	2.3±0.1
	0.0150	77.7±1.0	74.1±5.1	5.1±0.6	2.4±0.1

Therefore, we can see that the operator is beneficial in some domains but its effects are too much aggressive (creating poor solutions) when applied to other datasets. Reinitialize needs to be redefined to have a softer behavior. The simplest way to achieve this goal is to redefine the probability controlling the operator. The new probability decreases linearly through the iterations until it achieves value 0 at last iteration. This fix allows the system to explore more aggressively in the early iterations and later on, in the final iterations, refine the good solutions. We repeated the short test with the same datasets, using as initial probabilities the values 0.01,0.02,0.03,0.04. Results are in table 3.

Table 3. Short tests of the improved reinitialize operator

Dataset	Initial reinit. prob.	Train acc.	Test acc.	# of rules	Interv. per attr
mmg	0.00	78.4±1.8	65.0±9.0	6.6±1.0	2.4±0.1
	0.01	78.8±1.6	65.7±9.4	6.5±0.8	2.4±0.1
	0.02	78.4±1.7	66.2±9.4	6.5±1.0	2.5±0.1
	0.03	78.4±1.5	67.1±8.1	6.4±0.7	2.5±0.1
	0.04	78.0±1.8	67.2±8.0	6.6±1.0	2.5±0.1
pim	0.00	78.6±1.0	74.4±4.7	5.4±0.9	2.2±0.1
	0.01	78.7±1.0	74.6±4.5	5.3±0.9	2.3±0.1
	0.02	78.7±1.0	74.6±4.4	5.4±1.0	2.3±0.1
	0.03	78.6±1.0	75.1±4.5	5.3±0.7	2.3±0.1
	0.04	78.5±1.1	74.3±4.7	5.2±0.7	2.3±0.1

There are some interesting differences from the previous results. The train accuracy of tests with reinitialize operator is slightly higher that the original

configuration. This shows that we have achieved the objective of creating an operator that helps exploring the search space for better solutions while being soft enough that it does not destroy these solutions in the final iterations. Even more interesting is the test accuracy, because now we obtain an accuracy increase over the original *ADI* configuration in both domains.

3.3 Evolution of the Intervals per Attribute Ratio

Another important issue we want to analyze is how does it evolve the semantical structure of the rules through the iterations. That is, how many intervals per attribute do we have, and how are they distributed. We can see the evolution of the average number of intervals per attribute for the *mmg* problem in figure 5. All datasets present very similar behavior. The reason is the *hierarchical selection* operator [8] used to control the bloat effect. This operator promotes individuals that minimize the total sum of intervals of their rules, thus promoting individuals that have less rules and also less intervals per rule.

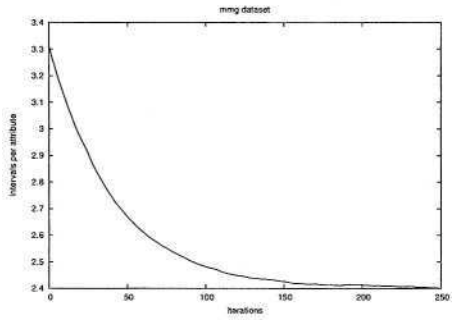


Fig. 5. Evolution of the average number of intervals per attribute

This behavior of the hierarchical selection is good for the system because it collapses the search space where it is possible and it helps creating generalized solutions that present low internal fragmentation of intervals. However, the difference between a well generalized attribute containing two intervals and an attribute too simple containing one (irrelevant) interval is only of one merge operation applied to the wrong attribute. In order to avoid part of this problem we introduce a **merge restriction** which cancels some merge operations if the attribute has only two intervals. The merge restriction process is represented in figure 6. The reader can see that now we have another probability to tune. For the sake of simplicity, in this paper we only test one value of this probability (0.5). The rationale of setting this value is to maintain an equilibrium between avoiding over-merging and creating too much specialized solutions.

1. An attribute has been selected for merge, according to the code in figure 1.
2. If the attribute has only one interval → we cancel the operation
3. If the attribute has two intervals then
   a) If random[0, 1] < probability of merge restriction → we cancel the operation
   b) Else → we apply the merge operation
4. If the attribute has more than two intervals → we apply the merge operation

Fig. 6. Merge operator with the new restriction included

4 Experimentation Design

4.1 Test Problems

For this paper we have selected 12 problems from the popular *UCI* repository [9] and also 3 real problems from private repositories (Biopsies [10], Mammograms [11] and Learning [7]). The characteristics of the problems are listed in table 4. The partition of the examples into the train and test sets was done using *stratified ten-fold cross-validation* [12].

Table 4. Characteristics of the test problems.

Name	ID.	Instances	Attributes	Classes
Bupa	bpa	345	6	2
Biopsies	bps	1027	24	2
Wisconsin Breast Cancer	bre	699	9	2
Glass	gls	214	9	6
Heart-Statlog	h-s	270	13	2
Ionosphere	ion	351	34	2
Learning	lrn	648	4	5
Mammograms	mmg	216	21	2
Pima-Indians-Diabetes	pim	768	8	2
Sonar	son	208	60	2
New-Thyroid	thy	215	5	3
Vehicle	veh	846	18	4
Wdbc	wdbc	569	30	2
Wine	wne	179	13	3
Wpbc	wpbc	198	33	2

4.2 Configurations of the GA to Test

We tested 6 configurations of the *ADI* representation: the original version of the representation and 5 combination of the two improvements presented in this paper. The configurations are detailed in table 5. We have chosen to test the **reinitialize operator** always combined with the **merge restriction** because we think that the benefits of reinitialize diminish if we have too much generalization pressure. The GA parameters are shown in table 6. In order to have an external reference of the system performance, we have also included results for the *C4.5* [13] and IB1 [14] systems (using the *WEKA* [12] implementation and default parameters).

Table 5. Configurations of the *ADI* representation to test

Name	Merge restriction prob.	Reinitialize prob.	Reinitialize prob. at end
Original ADI	0	0	0
New ADI1	0.5	0	0
New ADI2	0.5	0.005	0.005
New ADI3	0.5	0.010	0.010
New ADI4	0.5	0.02	0
New ADI5	0.5	0.03	0

Table 6. Common parameters of the GA.

Parameter	Value
General parameters	
Crossover probability	0.6
Selection algorithm	Tournament
Tournament size	3
Population size	300
Probability of mutating an individual	0.6
Iterations	A maximum of 1500. Depends on learning rate on each dataset
Rule Deletion operator	
Iteration of activation	5
Minimum number of rules	number of classes of domain + 3
Hierarchical Selection	
Iteration of activation	25
Threshold	0.01
ADI rule representation	
Number of intervals of the uniform discretizations	4,5,6,7,8,10,15,20,25
Split probability	0.05
Merge probability	0.05

5 Results

In this section we present the results of the test described in the previous section. For each of the 6 *ADI* configuration and the 2 external systems we show the average and standard deviation of the cross-validation accuracy. The *ADI* results are the average of repeating the tests with 15 different random seeds. The results are presented in table 7. We applied two-sided Student t-tests to the results [12] with a significance level of 1%, in order to determine if there were significant outperformances between the methods tested. The results of the t-tests are shown in table 8.

5.1 Analyzing Only the Performance of the ADI Configurations

If we look only at the *ADI* configurations we can see that the original *ADI* configuration was only the best method in 1 of the 15 tested domains, showing that the improvements introduced to the representation perform well. Furthermore, in the domain *(son)* where the new operators performed worse, the difference was not significant (according to the t-tests). Looking at the accuracy averages and also to the t-tests, we can see that there is a clear winner: the *New ADI4* configuration. This method has the top accuracy average, it is the method that

Table 7. Mean and deviation of the accuracy for each method tested. Bold entries show the best method for each test problem

Prob.	Original ADI	New ADI1	New ADI2	New ADI3	New ADI4	New ADI5	C4.5	IB1
bpa	63.7±7.4	63.7±7.9	63.9±7.3	62.6±8.1	63.3±7.3	63.2±7.4	**68.4±3.9**	64.5±8.5
bps	80.6±4.3	80.7±4.0	80.7±3.8	79.6±4.1	80.6±4.3	80.0±4.0	80.1±4.5	**83.2±3.0**
bre	95.6±2.2	95.8±2.2	**96.0±2.1**	95.7±2.2	95.8±2.2	95.9±2.1	95.4±1.5	**96.0±1.4**
gls	66.4±7.7	66.5±7.8	**67.9±7.0**	**67.9±7.2**	67.8±8.1	66.5±7.8	65.8±9.9	66.3±10.4
h-s	80.4±6.9	80.2±6.7	**80.7±6.6**	79.8±7.2	80.5±6.7	80.6±7.1	76.3±5.5	74.1±6.4
ion	91.6±4.4	90.9±4.5	92.2±3.8	91.7±3.7	**92.7±3.6**	92.0±3.9	89.8±4.8	86.9±4.6
lrn	68.1±4.8	68.0±5.5	68.6±5.5	67.8±4.8	68.9±5.4	**69.0±4.7**	68.6±4.4	61.4±5.8
mmg	65.0±9.0	66.1±8.9	66.0±7.5	**67.8±9.4**	67.0±8.1	**67.8±8.6**	64.8±6.0	63.5±11.5
pim	74.4±4.7	75.1±4.3	74.7±4.0	74.3±4.8	**75.3±4.4**	74.4±4.5	73.1±5.0	70.3±3.3
son	74.6±9.7	73.2±10.1	73.1±9.6	72.3±10.6	74.3±9.2	73.5±9.5	71.5±8.0	**87.3±9.2**
thy	91.9±5.6	92.0±5.7	91.4±6.4	91.6±5.8	92.0±5.2	91.5±5.9	92.6±3.7	**96.8±4.7**
veh	66.0±4.9	66.4±4.9	66.1±4.7	65.6±4.0	66.7±4.4	66.5±4.7	**73.6±5.0**	69.4±5.0
wdbc	93.8±3.2	93.7±3.0	93.8±3.1	93.9±3.1	94.0±3.1	93.7±3.1	93.7±2.1	**95.6±2.1**
wine	92.7±6.9	92.5±6.9	92.2±6.9	92.6±6.7	93.0±6.8	92.2±6.9	94.1±6.8	**95.6±4.8**
wpbc	75.7±7.4	75.5±6.4	76.1±7.7	75.9±6.5	**77.1±6.4**	76.8±7.2	73.7±7.4	68.8±10.1
average	78.7±11.4	78.7±11.2	78.9±11.2	78.6±11.3	**79.3±11.2**	78.9±11.1	78.8±10.9	78.6±13.0

Table 8. Summary of the statistical two-sided t-test performed at the 1% significance level. Each cell indicates how many times the method in the row outperforms the method in the column

Method	*Orig. ADI*	*New ADI1*	*New ADI2*	*New ADI3*	*New ADI4*	*New ADI5*	*C4.5*	*IB1*	*Total*
Orig. ADI	-	0	0	0	0	0	1	3	4
New ADI1	0	-	0	0	0	0	0	4	4
New ADI2	0	1	-	1	0	0	1	4	7
New ADI3	1	0	1	-	0	0	1	3	6
New ADI4	2	2	0	3	-	1	1	4	11
New ADI5	1	0	0	2	0	-	1	4	8
C4.5	2	1	1	2	2	2	-	0	10
IB1	3	4	4	3	3	3	2	-	22
Total	9	8	6	9	5	6	7	22	

outperforms significantly most often the other methods and, even more interesting, it has never been significantly outperformed.

Comparing *Original ADI* to *New ADI1* (where only the merge restriction improvement was used) we can see that they perform similarly in average, but *New ADI1* performs equal or better in the majority of domains. Thus, showing that it is a harmless fix that can be combined with the reinitialize operator without creating a bad interaction. The results of *New ADI3* show the dangers of the reinitialize operator where it is incorrectly tuned. It has the lowest accuracy average and it is the method most often significantly outperformed.

5.2 Comparing *ADI* to *C4.5* and *IB1*

If we now look at the results from a global point of view, we can see some big accuracy differences between the methods compared, specially between *IB1* and the other methods. These differences are quite logical, because they are a consequence of the *Selective Superiority Problem* [15]. Being *IB1* the only non axis-parallel classifier it was expected a change in the range of domains where this systems performs well.

Comparing the systems tested we can say that *New ADI4* is better (in average) than *C4.5* and *IB1*. Also, it is the system less times outperformed significantly, according to the T-Tests. This shows that this *ADI* configuration has a robust and reliable behavior.

6 Conclusions and Further Work

This paper was focused on an existing representation for real-valued attributes: the Adaptive Discretization Intervals (ADI) rule representation. This representation evolves rules that can use multiple discretizations, letting the evolution choose the correct discretization for each rule and attribute. Also, the intervals defined in each discretization can split or merge among them through the evolution process, reducing the search space where it is possible. We have analyzed the behavior of the representation, studying its evolution through the iterations in two aspects: the proportions of each discretizer in the population and the number of intervals per attribute. From studying the proportions we have discovered that all discretizers can disappear sometimes from the population. Studying the evolution of the intervals per attribute ratio we have seen that the pressure applied to reduce this ratio can easily transform well generalized solutions into too much simple ones.

This analysis has led us to do two proposals: The first one is an operator that can reintroduce fresh discretizers and sets of intervals into the population. This operator, if badly tuned, can be very destructive. Thus, we have tested a dynamic probability decreasing its value through the iterations. The second one is a restriction introduced into the merge operator, in order to avoid the over-reduction of the number of intervals in the population.

We have tested 5 combinations of these two improvements with various parameterizations across 15 datasets and two alternative Machine Learning systems. The comparison of these configurations to the original *ADI* version shows that the improvements done have better performance. It is interesting to remark that the configuration performing better in average *(New ADI4)* has been outperformed significantly very few times, showing that the improvements presented here are good and robust.

As further work it would be interesting to combine these improvements with alternative sets of discretizers, containing both uniform and non-uniform discretization algorithms, as an alternative to the pool of uniform-width discretizers used for this paper. Also, we should analyze in depth the interactions between the two kind of improvements studied.

7 Related Work

Discretization is not the only way to handle real-valued attributes in Evolutionary Computation based Machine Learning systems. Some examples are induction of decision trees (either axis-parallel or oblique), by either generating a full tree by means of genetic programming operators [3] or using an heuristic method to

generate the tree and using a Genetic Algorithm and an Evolution Strategy to optimize the test performed at each node [16]. Other examples are inducing rules with real-valued intervals [17,18] or generating an instance set used as the core of a *k-NN* classifier [3]. If our *ADI* method is able to find the correct cut points, the performance of the system should be quite competitive if compared to all the axis-parallel methods described above (all but the last one). Also, there are other systems that, like *ADI,* perform evolutionary induction of rules based on discretization [4,5]. A comparison of *ADI* with these two methods is found in [19].

Acknowledgments. The authors acknowledge the support provided under grant numbers 2001FI 00514, TIC2002-04160-C02-02, TIC 2002-04036-C05-03 and 2002SGR 00155. Also, we would like to thank Enginyeria i Arquitectura La Salle for their support to our research group.

References

1. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press (1975)
2. Wilson, S.W.: Classifier fitness based on accuracy. Evolutionary Computation **3** (1995) 149–175
3. Llorà, X., Garrell, J.M.: Knowledge-independent data mining with fine-grained parallel evolutionary algorithms. In: Proceedings of the Third Genetic and Evolutionary Computation Conference, Morgan Kaufmann (2001) 461–468
4. Giráldex, R., Aguilar-Ruiz, J., Riquelme, J.: Natural coding: A more efficient representation for evolutionary learning. In: GECCO 2003: Proceedings of the Genetic and Evolutionary Computation Conference, Springer (2003) 979–990
5. Divina, F., Keijzer, M., Marchiori, E.: A method for handling numerical attributes in GA-based inductive concept learners. In: GECCO 2003: Proceedings of the Genetic and Evolutionary Computation Conference, Springer (2003) 898–908
6. DeJong, K.A., Spears, W.M.: Learning concept classification rules using genetic algorithms. Proceedings of the International Joint Conference on Artificial Intelligence (1991) 651–656
7. Bacardit, J., Garrell, J.M.: Evolving multiple discretizations with adaptive intervals for a Pittsburgh rule-based learning classifier system. In: Proceedings of the Genetic and Evolutionary Computation Conference - GECCO2003, LNCS 2724, Springer (2003) 1818–1831
8. Bacardit, J., Garrell, J.M.: Bloat control and generalization pressure using the minimum description length principle for a Pittsburgh approach learning classifier system. In: Proceedings of the 6th International Workshop on Learning Classifier Systems, (in press), LNAI, Springer (2003)
9. Blake, C., Keogh, E., Merz, C.: UCI repository of machine learning databases (1998) (www.ics.uci.edu/mlearn/MLRepository.html).
10. Martínez Marroquín, E., Vos, C., et al.: Morphological analysis of mammary biopsy images. In: Proceedings of the IEEE International Conference on Image Processing. (1996) 943–947

11. Martí, J., Cufí, X., Regincós, J., et al.: Shape-based feature selection for microcalcification evaluation. In: Imaging Conference on Image Processing, 3338:1215-1224. (1998)
12. Witten, I.H., Frank, E.: Data Mining: practical machine learning tools and techniques with java implementations. Morgan Kaufmann (2000)
13. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann (1993)
14. Aha, D.W., Kibler, D.F., Albert, M.K.: Instance-based learning algorithms. Machine Learning **6** (1991) 37–66
15. Brodley, C.: Addressing the selective superiority problem: Automatic algorithm /model class selection (1993)
16. Cantu-Paz, E., Kamath, C.: Inducing oblique decision trees with evolutionary algorithms. IEEE Transactions on Evolutionary Computation **7** (2003) 54–68
17. Wilson, S.W.: Get real! XCS with continuous-valued inputs. In Booker, L., Forrest, S., Mitchell, M., Riolo, R.L., eds.: Festschrift in Honor of John H. Holland, Center for the Study of Complex Systems (1999) 111–121
18. Stone, C., Bull, L.: For real! xcs with continuous-valued inputs. Evolutionary Computation Journal **11** (2003) 298–336
19. Aguilar, J., Bacardit, J., Divina, F.: Experimental evaluation of discretization schemes for rule induction. In: GECCO 2004: Proceedings of the Genetic and Evolutionary Computation Conference, Springer (to appear) (2004)

Bounding Learning Time in XCS

Martin V. Butz, David E. Goldberg, and Pier Luca Lanzi

Illinois Genetic Algorithms Laboratory (IlliGAL)
University of Illinois at Urbana-Champaign
Urbana, IL, 61801
{butz,deg,lanzi}@illigal.ge.uiuc.edu

Abstract. It has been shown empirically that the XCS classifier system solves typical classification problems in a machine learning competitive way. However, until now, no learning time estimate has been derived analytically for the system. This paper introduces a time estimate that bounds the learning time of XCS until maximally accurate classifiers are found. We assume a domino convergence model in which each attribute is successively specialized to the correct value. It is shown that learning time in XCS scales polynomially in problem length and problem complexity and thus in a machine learning competitive way.

1 Introduction

Although the learning classifier system framework was proposed more than thirty years ago [1], the theoretical understanding is still rather sparse. Due to the complex interaction of several adaptive mechanisms, including the evolutionary learning method, the credit assignment mechanism, and the distributed problem representation, the exact analysis of the systems is hard. Recently, two classifier systems have reached most attention in the literature: the strength-based classifier system ZCS [2,3] and the accuracy-based classifier system XCS [4,5].

The XCS classifier system has shown to solve typical classification problems competitively to other machine learning algorithms [6,7]. Also the theoretic understanding is increasing [8,9,5,10]. In particular, problem bounds have been found that bound the minimal population size in order to assure that the evolutionary algorithm applies (the *covering challenge*), that necessary minimal order schemata are available (the *schema challenge*), that the found subsolutions (or schemata) have the chance to reproduce before being deleted (the *reproductive opportunity bound*), and that the different subsolutions have enough support to avoid the loss of a niche (the *support challenge,* in preparation). However, until now, no estimate has been found that approximates the time until complete knowledge is reached.

Specifically, we are interested in how many learning steps XCS needs to evolve the intended accurate, maximally general model of the applied task. Assuming that all the other challenges are met, we can estimate how long it takes to discover successively better classifiers until the maximally general, accurate classifiers are found. This paper focuses on mutation-driven evolution, taking

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 739–750, 2004.

into account time until reproduction and time until generation of the next best classifier via mutation. The experimental study confirms that XCS learning time is approximated by the derived time bound. The bound shows that learning time scales polynomially in the problem length and exponentially in the order of the problem (polynomially in problem complexity).

The next section gives an overview of the XCS classifier system and existing theory. Given that all other bounds are met, we then derive the time bound for XCS learning. The experimental study that follows shows that XCS learning scales in the specified time bound. Moreover, several operator and parameter influences are identified. Concluding remarks put the derived bound into a broader perspective of learning classifier system and machine learning research.

2 XCS Theory

The accuracy-based learning classifier system XCS was introduced elsewhere [4]. Due to the accuracy-based fitness approach, XCS learns not only the rules (or classifiers) that denote the best classification possible, but rather a complete situation-action-reward mapping. In short, XCS is designed to learn a complete, accurate, and maximally general *payoff map* of an environment (out of which an optimal behavioral/classification policy can be derived).

XCS knowledge is represented by a *population* of maximally N condition-action-reward prediction classifiers. Each classifier essentially specifies the expected reward given the specified conditions and executing the specified action. Rule evaluation is done via a credit assignment mechanism [11] with similarities to algorithms in reinforcement learning [4,12]. Rule generation and evolution is done via a steady-state, niched genetic algorithm [13,14]. If a GA is applied in a particular learning iteration, two classifiers are reproduced using tournament selection [15] with respect to their fitness in the current action set (the subset of classifiers that contains all classifiers that specify the executed action and whose conditions are satisfied by the current problem instance). To keep the population size constant, two classifiers are deleted via proportionate selection from the whole population.

The following XCS theory focuses on Boolean function problems in which each problem instance is represented by l binary features and belongs to one of n classes. The *specificity* of a classifier refers to the number of features that the classifier condition specifies over the total number of features l (usually unspecified features are represented by the don't care symbol #).

2.1 Evolutionary Pressures in XCS

After Kovacs analyzed the problem of *strong overgenerals* that shows that any strength-based learning classifier system (without fitness sharing techniques) suffers from the tragedy of the common [16], recent analysis investigated the evolutionary learning progress in XCS. In [9] five major evolutionary pressures in XCS, (1) set pressure, (2) mutation pressure, (3) deletion pressure, (4) subsumption pressure, and (5) fitness pressure, were identified.

Intuitively, the *set pressure* formalizes the intrinsic generalization pressure in XCS. Since the average specificity $s[A]$ of classifiers in an action set $[A]$ is lower than the average specificity $s[P]$ in the population and classifiers are reproduced in action sets but deleted in the population, the average offspring specificity is lower than the specificity of the replaced classifiers.

Also *mutation pressure* influences specificity. Generally, a random mutation process causes a tendency towards an equal number of symbols in a population. Thus, applying random mutations, the result will be a population with an approximately equal proportion of 0, 1, and # symbols in the condition parts of classifiers in a binary LCS. Thus, mutation results in a pressure towards an equal number and an equal distribution of symbols in classifier conditions.

Combining set and mutation pressure to a general *specificity equation* [10] a general estimate of the change in the population's specificity can be derived:

$$\Delta_{spe} s([P]) = f_{ga} \frac{2\,(s([A]) + \Delta_{mut} - s([P]))}{N}. \tag{1}$$

Parameter f_{ga} approximates the average frequency of GA application, $s([X])$ refers to the average specificity of the referred set X, Δ_{mut} quantifies the specificity change due to mutation, and N specifies the population size. The formula allows an accurate prediction of specificity change and convergence over time given no fitness influence [9,10]. It was also shown that given no fitness influence, the converged specificity in the population can be roughly approximated by twice the mutation rate.

The main part of the *deletion pressure* is already included in the set pressure. In addition to deleting classifiers from the population while reproducing classifiers in action sets, deletion is biased towards deleting classifiers that populate large niches and classifiers with a fitness value which is significantly smaller than the average value of the population [17].

Subsumption pressure is designed to decrease the population size boiling it down to the accurate, maximally general classifiers. It applies only if the noise in a problem is lower than the error threshold ε_0 below which a classifier is considered for subsumption. Subsumption must be applied with care. A too low experience threshold θ_{sub} as well as a too-high value of ε_0 can cause fundamental loss of information in the population (subsuming accurate classifiers by a temporarily accurate, over-general classifier).

Fitness pressure is needed to generate a major drive towards accuracy from the inaccurate (and thus mainly over-general) side. Essentially, fitness pressure causes the reproduction of higher accurate classifiers. Thus, fitness is the major pressure that guides the evolutionary process towards higher accuracy.

Over the last years it became clear that, given the problem provides appropriate fitness guidance, fitness pressure needs to be strong enough to overcome the set pressure. Since the traditionally applied proportionate selection mechanism is highly dependent on fitness scaling, a set-size proportionate *tournament selection* mechanism was introduced to XCS [15] that results in a more robust and more problem independent XCS learning system.

2.2 Problem Bounds

In addition to the above evolutionary pressure in XCS, several problem bounds have been identified. The following paragraphs give an overview over the derived problem bounds.

Covering Challenge. The first bound was formulated in [8] requiring a minimal population size to assure that the genetic algorithm actually applies and is not blocked by a continuous covering-deletion cycle. Given a current specificity of $s[P]$ in the population and assuming a problem in which any possible problem instance is equiprobable, the covering probability is determined by

$$P(cover) = 1 - \left[1 - \left(\frac{2 - s([P])}{2}\right)^l\right]^N, \tag{2}$$

where l specifies the number of binary features in the problem. In order to keep the covering probability initially sufficiently high to ensure the start of the evolutionary process, the initial specificity needs to be set sufficiently low (controlled by the don't care probability $P_\#$).

Schema Challenge. In addition to the assurance of input covering, it also needs to be assured that classifiers represent particular schemata. To characterize such a classifier, Holland's schema notion is used [13]. A *representative* of a particular schema of order o must have at least all o positions correctly specified. For successful evolution, the presence of a representative of a schema of order o needs to be highly probable. The probability of the existence of a representative can be determined by

$$P(representative) = 1 - \left[1 - \frac{1}{n}\left(\frac{s([P])}{2}\right)^o\right]^N, \tag{3}$$

assuming a binomial distribution of the specificity in the population. Parameter n denotes the number of classes. While the previous specificity measure is mainly relevant for the beginning of the run, the current specificity of the population directly affects the probability of the availability of a representative.

Reproductive Opportunity. To ensure successful evolution, it is necessary to assure gradual evolution ensuring *reproductive opportunities* for the better classifiers. Existing or generated higher-accurate classifiers need to have reproductive opportunities before being deleted. To ensure this, the expected time until a reproductive opportunity should be shorter than the expected time until deletion. This constraint effectively results in a population size bound since only a larger population size can increase the time until deletion in XCS [5].

$$N > n2^{k_d + (l - k_d)s([P]) + 1} \tag{4}$$

This bound ensures that classifiers necessary in a problem of order of minimal schema order k_d get reproductive opportunities. Once the bound is satisfied,

existing representatives of an order k_d schema have a high probability of reproduction. Thus, with a high probability, XCS will evolve a more accurate population.

Note that this population size bound is actually exponential in minimal schema order k_d and in string length times specificity $ls([P])$. However, it was shown that the necessary specificity in $[P]$ decreases with larger population sizes [5]. In particular, requiring that a representative of a particular schema order k_d exists, it can be shown that the required minimal specificity is bounded by

$$O(\left(\frac{n}{N}\right)^{\frac{1}{k_d}}). \tag{5}$$

Considering this, a general *reproductive opportunity bound (ROP-bound)* can be derived that shows that population size grows as

$$O(l^{k_d}). \tag{6}$$

The bound essentially determines that the populations size grows polynomially in the problem length l and exponentially in the problem difficulty. Thus, the computational complexity grows similar to any inductive machine learning algorithm such as for example the inductive decision tree learner C4.5 [18]. The specificity bound and population size bound will also be relevant for the following derivation of the learning time.

3 Bounding Learning Time in XCS

To derive our learning time bound, we estimate the time until reproduction of the current best classifier as well as the time until creation of the next best classifier via mutation given a reproductive event of the current best classifier. The model assumes a completely general initial population. First specializations are randomly introduced via mutation. Problem-specific initialization techniques or a higher initial specificity in the population may speed-up learning time (as long as the covering challenge is not violated). Further assumptions are that the current best classifier is not lost (assured by the ROP-bound) and that it is selected as the offspring when it is part of an action set (assured by the selection mechanism). The time model assumes domino convergence [19] in which each attribute is successively specified. This means that only once the first attribute is correctly specified in a classifier, then the second attribute influences fitness and so forth.

With the above assumptions, we can bound the learning time in the following way. First, we estimate the probability that mutation correctly specifies the next attribute

$$P(\text{perfect mutation}) = \mu(1-\mu)^{l-1} \tag{7}$$

where l specifies the number of attributes in a problem instance (i.e. condition length). This probability can be relaxed in that we only require that the k already

correctly set features are not unset (changed to don't care), the next feature is set, and we do not care about the others:

$$P(\text{good mutation}) = \mu(1-\mu)^k \tag{8}$$

Equation 7 specifies the lower bound on the probability that the next best classifier is generated whereas Equation 8 specifies an optimistic bound.

The probability of reproduction of a classifier is mainly influenced by the probability of being part of an action set. The probability of being part of an action set again, is determined by the current specificity of a classifier. Given a classifier which specifies k attributes, the probability of reproduction is

$$P(\text{reproduction}) = \frac{1}{n}\frac{1}{2}^k \tag{9}$$

where n denotes the number of actions in a problem. The best classifier has a minimal specificity of k/l. With respect to the current specificity in the population $s([P])$, the specificity of the best classifier may be expected to be $k+s([P])(l-k)$ assuming a uniform specificity distribution in the other $l-k$ attributes. Taking this expected specificity into account, the probability of reproduction is

$$P(\text{reproduction in [P]}) = \frac{1}{n}\frac{1}{2}^{k+s([P])(l-k)}. \tag{10}$$

Since the probability of a successful mutation assumes a reproductive event, the probability of generating a better offspring than the current best is determined by

$$P(\text{generation of next best cl.}) = P(\text{reproduction in [P]})\ P(\text{good mutation}) = \frac{1}{n}\frac{1}{2}^{k+s([P])(l-k)} \mu(1-\mu)^{l-1}. \tag{11}$$

Since this is a geometric distribution (memoryless property, each trial has an independent and equally probable distribution), the expected time til the generation of the next best classifier is

$$E(\text{time until generation of next best cl.}) = 1/P(\text{generation of next best cl.}) = \frac{1}{\frac{1}{n}\frac{1}{2}^{k+s([P])(l-k)}\mu(1-\mu)^{l-1}} = \frac{n2^{k+s([P])(l-k)}}{\mu(1-\mu)^{l-1}} \leq \frac{n2^{k+s([P])l}}{\mu(1-\mu)^{l-1}}. \tag{12}$$

Given now a problem in which o features need to be specified and given further the domino convergence property in the problem, the expected time until the generation of the next best classifier can be summed to derive the time until the generation of the global best classifier:

$$E(\text{time until generation of maximally accurate cl}) = \sum_{k=0}^{o-1} \frac{n2^{k+s([P])l}}{\mu(1-\mu)^{l-1}} = \frac{n2^{s([P])l}}{\mu(1-\mu)^{l-1}} \sum_{k=0}^{o-1} 2^k < \frac{n2^{o+s([P])l}}{\mu(1-\mu)^{l-1}}. \tag{13}$$

This time bound shows that XCS needs an exponential number of evaluations in the problem difficulty o. As argued above, the specificity and consequently also mutation needs to be decreased indirect proportional to the string length l. In particular, since specificity $s([P])$ grows as $O((\frac{n}{N})^{\frac{1}{k_d}})$ (Equation 5) and population size grows as $O(l^{k_d})$ (Equation 6), specificity essentially grows as $O(\frac{n}{l})$. Using the O-notation and plugging this behavior into Equation 13 we derive the following adjusted time bound.

$$O\left(\frac{l2^{o+n}}{(1-\frac{n}{l})^{l-1}}\right) = O\left(\frac{l2^{o+n}}{e^{-n}}\right) = O\left(l2^{o+n}\right) \tag{14}$$

Thus, learning time in XCS is bound mainly by the order of problem difficulty o and the number of problem classes n. It is linear in the problem length l. This derivation essentially also validates Wilson's hypothesis that XCS learning time grows polynomially in problem complexity as well as problem length [20]. The next section experimentally validates the derived learning bound.

4 Experimental Validation

In order to validate the derived bound, we evaluate XCS performance on an artificial problem in which domino convergence is forced to take place. Similar results are expected in typical Boolean function problems in which similar fitness guidance is available, such as in the layered multiplexer problem [4,5]. In other problems, additional learning influences may need to be considered such as the influence of crossover or the different fitness guidance in the problem [5].

To force domino convergence, instead of using the usual Widrow-Hoff delta rule to update classifier estimates, we set the reward prediction error directly to a fixed value according to the current specificity of the classifier. Given a problem of problem difficulty o the prediction error of a classifier is set to $500(o-k)/o$ where k denotes the number of successive relevant attributes specified in the classifier. Thus, given a problem of length $l = 6$ and $o = 3$ (and assuming a left-to-right order with the first three features being relevant), the classifier 1#1111 would be assigned an error of 333 whereas classifier 011#1# would be assigned an error of 0.

If not stated differently, the XCS classifier system is applied with a GA threshold $\theta_{GA} = 0$ (the GA is always applied), error instead of fitness-based selection, tournament selection with a action-set proportionate tournament size of $\tau = 0.4$, niche mutation, no action mutation, no crossover, an initial completely general population ($P_{\#} = 1.0$) and GA subsumption ($\theta_{sub} = 0, \epsilon_0 = 1$). The results are averaged over 20 experiments. The error-based selection approach eliminates the additional evolutionary influence due to fitness sharing.

4.1 Time Bound Validation

To validate the time bound, we monitor the specificity of the relevant attributes. According to the domino convergence theory, the system should successively detect the necessary specialization of each relevant attribute eventually converging

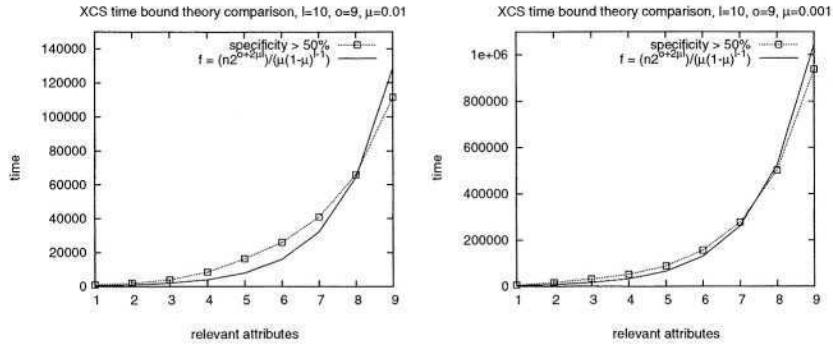


Fig. 1. The theory comparison shows that the time until the specificity of the successive attributes has reached 50% is approximated by the theoretical bound. Maximum population size N is set to 32000.

to a specificity of nearly 100%. The time bound estimates the expected time until all relevant attributes are detected. To evaluate the bound, we record the number of steps until the specificity of a particular attribute reaches 50%. This criterion indicates that the necessary specificity is correctly detected but it does not require full convergence.

Figure 1 shows the time until 50% specificity is reached in the successive attributes in the setting with $l = 10$ and $o = 9$. The comparison with the theoretical bounds matches approximating the specificity in the population $s([P])$ with 2μ which has been shown to be approximately correct [5]. As predicted by the theory, decreasing the mutation rate (Figure 1, right-hand side) increases the time until the required specificity is reached. Although nearly all interactions between the different niches are prevented by disallowing the mutation of the action part and by applying niche mutation only, the specificity in the later attributes still is learned slightly faster than predicted by the theory. Two-stage interactions might occur in which mutation first overgeneralizes a highly accurate classifier and then specializes it in another niche.

The second concern is the influence of the number of irrelevant attributes. Figure 2 shows that also in this case the theory closely matches the empirical results. Since a higher mutation rate results in a higher specificity, the influence of the number of irrelevant attributes is more significant in the setting with a mutation rate of $\mu = 0.01$. In the low mutation case, the bound is approximated in the settings with larger string length. Hereby, the specificity is approximated by twice the mutation rate. Using a smaller population size can delay or stall the evolutionary process due to the reproductive opportunity bound.

Figure 3 shows the behavior of the specificities of the six relevant positions and several of the other positions (that all behave similarly). It can be seen that complete convergence is delayed if population size is set not high enough. The reproductive opportunity bound slowly comes into play. With a string length of more than 150, evolution partially stalls completely since the overspecialized

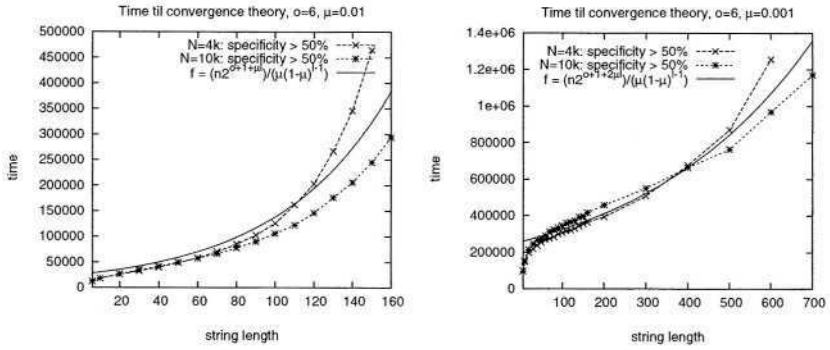


Fig. 2. The influence of problem length is properly bound by the theory. The reproductive opportunity bound increasingly outweighs the time bound when the population size is set too low.

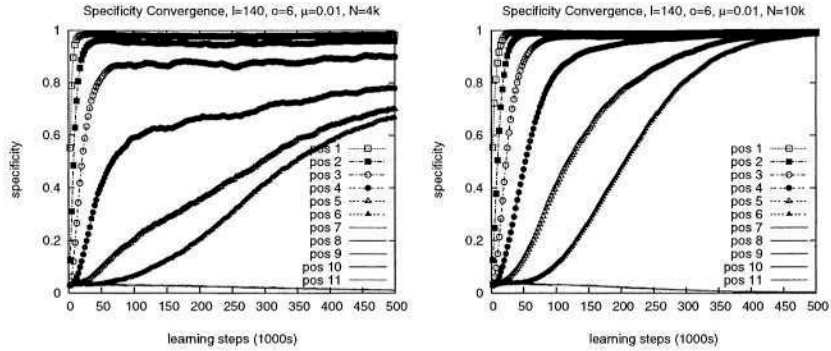


Fig. 3. Increasing the problem length further, the reproductive opportunity bound increasingly affects convergence stalling learning by preventing the reproduction of more-accurate classifiers. Increasing the population size remedies the reproductive opportunity bound giving more-accurate classifiers more time for reproduction.

irrelevant attributes prevent sufficient reproductive opportunities (see Section 2.2 and [5]). With a higher population size, the reproductive opportunity bound vanishes and all six specificities converge to one without delay. Similar behavior is found for the case with a lower mutation rate and larger string length as indicated in Figure 2 (right-hand side).

4.2 Parameter Influences

In addition to the above bound, we investigate the effects of several parameters and additional mechanisms in XCS. Figure 4 reveals dependences on several XCS mechanisms in the setting with a string length $l = 10$ and the number of relevant attributes $o = 4$ (left-hand side) and $o = 8$ (right-hand side). In the setting with four relevant attributes, we can see that the disallowance of action mutation as

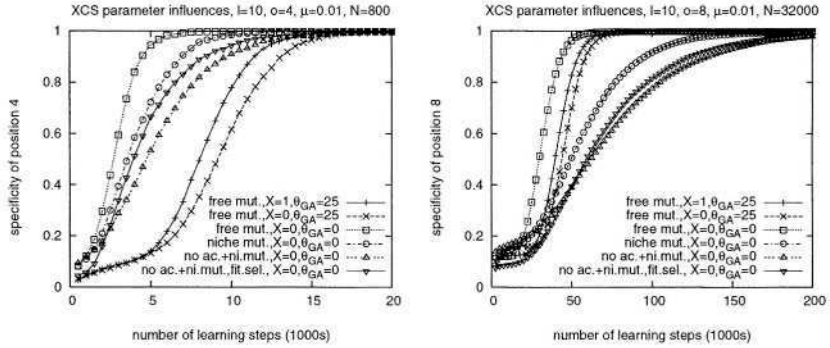


Fig. 4. Several mechanisms influence the learning speed: action mutation and free mutation facilitate the additional knowledge exchange between different problem niches and subsolutions. A higher GA threshold delays evolution especially given a more generalized population. Fitness-based selection slightly speeds-up learning. More relevant attributes and thus a more specialized population show similar performance influences (right-hand side).

well as the restriction to niche mutation decreases performance. Allowing action mutation or free mutation, subsolutions in one problem niche can propagate much easier to another niche (by mutating action 1 to 0 or specified attribute 1 to 0 or vice versa). Increasing the GA threshold θ_{GA} delays the evolutionary process. Uniform crossover has an additional beneficial effect enhancing the possibility of knowledge exchange between different niches. Fitness-based selection also slightly speeds-up learning. Due to the fitness decrease in offspring (fitness is set to 10% of the parental fitness), a slight generalization pressure [21] is generated that decreases specificity (in particular the specificity of the irrelevant attributes) and thus facilitates learning. In the setting with eight relevant attributes (right-hand side), the performance decrease due to restricted mutation overshadows the decrease due to a higher GA threshold.

5 Summary and Conclusions

This paper introduced a first learning time bound to the XCS classifier system. Assuming a domino convergence model in which each relevant attribute converges successively, we showed that learning time grows polynomially in problem complexity and linearly in the problem length. The provided experiments validated the basic assumptions in the derivations hold. The results confirm Wilson's original learning time estimation [20]. Additional learning mechanisms were shown to improve learning speed enabling a better knowledge exchange between different subsolutions or niches such as free mutation or crossover. Satisfying all problem bounds in XCS and given a problem structure that allows domino convergence, we can now assure that XCS learns a problem in time polynomial in problem complexity and problem length.

The research points out the strong dependence of successful learning on the underlying problem structure. Several problem difficulty measures were detected. The most trivial one is the problem length in which XCS scales polynomially. The second one is the order of the problem, which specifies the minimal number of attributes that need to be specified to be maximally accurate. We showed that XCS learning time scales exponentially in the problem order and thus polynomially in problem complexity. Finally, previously we showed that the minimal number of attributes that need to be specified to reach higher accuracy is a third problem bound (discussed in Section 2.2). Given a problem with a particular order of problem difficulty, we are now able to estimate the required population size and estimate the required learning time to solve the problem successfully.

The current time bound is applied in problems in which the domino convergence property holds. That is, each attribute progressively reduces the error estimate of a classifier and thus increases its accuracy and fitness. In problems in which this property does not hold, the convergence time may vary and other operators may be necessary to achieve successful learning. In particular, different substructures of accurate subsolutions may need to be recombined in a proper way applying intelligent recombination operators. Also, fitness guidance may be violated and a bilateral fitness approach may be necessary, as investigated in [5]. Nonetheless, this learning time bound shows that XCS is able to learn in a machine-learning competitive way. Future research will show the ways the bound can be modified and enhanced to account for recombinatory events as well as for other problem types.

Acknowledgment. We are grateful to Xavier Llorà, Kei Onishi, Kumara Sastry, and the whole IlliGAL lab for their help and the useful discussions. The work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant F49620-03-1-0129. The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon. Additional funding from the German research foundation (DFG) under grant DFG HO1301/4-3 is acknowledged. Additional support from the Computational Science and Engineering graduate option program (CSE) at the University of Illinois at Urbana-Champaign is acknowledged. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research or the U.S. Government.

References

1. Holland, J.H.: Processing and processors for schemata. In Jacks, E.L., ed.: Associative Information Techniques, New York, American Elsevier (1971) 127–146
2. Wilson, S.W.: ZCS: A zeroth level classifier system. Evolutionary Computation **2** (1994) 1–18
3. Bull, L., Hurst, J.: ZCS redux. Evolutionary Computation **10** (2002) 185–205

4. Wilson, S.W.: Classifier fitness based on accuracy. Evolutionary Computation **3** (1995) 149–175
5. Butz, M.V., Goldberg, D.E., Tharakunnel, K.: Analysis and improvement of fitness exploitation in XCS: Bounding models, tournament selection, and bilateral accuracy. Evolutionary Computation **11** (2003) 239–277
6. Bernadó, E., Llorà, X., Garrell, J.M.: XCS and GALE: A comparative study of two learning classifier systems and six other learning algorithms on classification tasks. In Lanzi, P.L., Stolzmann, W., Wilson, S.W., eds.: Advances in learning classifier systems: Fourth international workshop, IWLCS 2001 (LNAI 2321). Springer-Verlag, Berlin Heidelberg (2002) 115–132
7. Dixon, P.W., Corne, D.W., Oates, M.J.: A preliminary investigation of modified XCS as a generic data mining tool. In Lanzi, P.L., Stolzmann, W., Wilson, S.W., eds.: Advances in learning classifier systems: Fourth international workshop, IWLCS 2001 (LNAI 2321). Springer-Verlag, Berlin Heidelberg (2002) 133–150
8. Butz, M.V., Kovacs, T., Lanzi, P.L., Wilson, S.W.: How XCS evolves accurate classifiers. Proceedings of the Third Genetic and Evolutionary Computation Conference (GECCO-2001) (2001) 927–934
9. Butz, M.V., Pelikan, M.: Analyzing the evolutionary pressures in XCS. Proceedings of the Third Genetic and Evolutionary Computation Conference (GECCO-2001) (2001) 935–942
10. Butz, M.V., Kovacs, T., Lanzi, P.L., Wilson, S.W.: Toward a theory of generalization and learning in XCS. IEEE Transactions on Evolutionary Computation **8** (2004) 28– 46
11. Holland, J.H., Reitman, J.S.: Cognitive systems based on adaptive algorithms. In Waterman, D.A., Hayes-Roth, F., eds.: Pattern directed inference systems. Academic Press, New York (1978) 313–329
12. Lanzi, P.L.: Learning classifier systems from a reinforcement learning perspective. Soft Computing: A Fusion of Foundations, Methodologies and Applications **6** (2002) 162–170
13. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, MI (1975) second edition 1992.
14. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading, MA (1989)
15. Butz, M.V., Sastry, K., Goldberg, D.E.: Tournament selection in XCS. Proceedings of the Fifth Genetic and Evolutionary Computation Conference (GECCO-2003) (2003) 1857–1869
16. Kovacs, T.: Towards a theory of strong overgeneral classifiers. Foundations of Genetic Algorithms 6 (2001)
17. Kovacs, T.: Deletion schemes for classifier systems. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99) (1999) 329–336
18. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann, San Francisco, CA (1993)
19. Thierens, D., Goldberg, D.E., Pereira, A.G.: Domino convergence, drift, and the temporal-salience structure of problems. In: Proceedings of the 1998 IEEE World Congress on Computational Intelligence, New York, NY, IEEE Press (1998) 535–540
20. Wilson, S.W.: Generalization in the XCS classifier system. Genetic Programming 1998: Proceedings of the Third Annual Conference (1998) 665–674
21. Bull, L.: Investigating fitness sharing in a simple payoff-based learning classifier system. Technical Report UWELCSG03-009, University of Western England, Learning Classifier System Group, Bristol, UK (2003)

Gradient-Based Learning Updates Improve XCS Performance in Multistep Problems

Martin V. Butz, David E. Goldberg, and Pier Luca Lanzi

Illinois Genetic Algorithms Laboratory (IlliGAL)
University of Illinois at Urbana-Champaign
Urbana, IL, 61801
{butz,deg,lanzi}@illigal.ge.uiuc.edu

Abstract. This paper introduces a gradient-based reward prediction update mechanism to the XCS classifier system as applied in neural-network type learning and function approximation mechanisms. A strong relation of XCS to tabular reinforcement learning and more importantly to neural-based reinforcement learning techniques is drawn. The resulting gradient-based XCS system learns more stable and reliable in previously investigated hard multistep problems. While the investigations are limited to the binary XCS classifier system, the applied gradient-based update mechanism appears also suitable for the real-valued XCS and other learning classifier systems.

1 Introduction

Despite the recent encouraging applications of the accuracy-based XCS classifier system in data-mining problems [1,2,3,4], successful applications in multistep problems have been restricted to small problems [5,6]. It was shown that without further additions, XCS is not able to solve environments robustly that allow only few generalizations or that require a larger number of steps until reinforcement is encountered [7,8].

Although *learning classifier systems* (LCSs) were developed within the evolutionary computation community rather independently from reinforcement learning research, temporal difference learning methods in LCSs can be compared to *reinforcement learning* (RL). For example, Q-learning [9] is tightly linked to the RL mechanism in the ZCS system [10] and the XCS system [5,11]. Besides the RL relation, LCSs may be viewed as evolutionary-based *function approximation* methods. The XCS system, for example, has been shown to be applicable as a pure function approximator [12].

Over the last years, it has become clear that tabular-based RL scales-up poorly. Thus, function approximation methods have been applied in the RL literature [13,14]. These mostly neural-based function approximators can be highly unstable if direct gradient methods are used to implement Q-learning [14]. Accordingly, residual gradient methods have been developed to improve robustness [15].

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 751–762, 2004.

The aim of this paper is to explore the possibility of applying gradient-based update methods in LCSs and in particular in the XCS system. We show how LCSs are related to neural function approximation methods and use similar gradient-based methods to improve performance. We show that XCS with gradient update methods reaches higher robustness and stability.

The paper is structured as follows. First, we provide the necessary background knowledge on RL including Q-learning and the gradient approaches. Next, we introduce XCS and reveal the differences in its RL approach. We consequently extend XCS with a gradient-based update mechanism. The subsequent performance analysis shows that XCS with gradient descent learns typically hard multistep problems much more reliable. In conclusion, we discuss the similarities of XCS, and LCSs in general, with neural-based RL methods and suggest further comparisons and enhancements.

2 Reinforcement Learning

Reinforcement learning problems are problems in which an agent interacts with an unknown environment. The environment provides state information and a numerical reward as feedback to the agent. The agent's task is to maximize the cumulative reward on the long run. LCSs essentially face a RL problem.

Most research in RL focuses on problems that can be modeled with a finite Markov decision process (MDP). An MDP is formally defined by a finite set S of states; a finite set A of actions; a transition function T ($T : S \times A \to \Pi(S)$) that assigns to each state-action pair a probability distribution ($\Pi(S)$) over states S, and a reward function R ($R : S \times A \to \mathbb{R}$).

At a certain time t, the agent senses its environment perceiving state s_t; based on this state information the agent selects an action a_t. After the execution of action a_t, the agent receives a scalar reward r_{t+1} and a new state s_{t+1}. The agent's goal is to *maximize* the amount of reward it receives from the environment *in the long run.* This is usually expressed as the *discounted expected payoff* which at time t is defined as follows:

$$E\left[\sum_{k=0}^{\infty} \gamma^k r_{t+1+k}\right], \tag{1}$$

where γ is the *discount factor* ($0 \leq \gamma \leq 1$) that specifies the importance of future reward. The larger γ, the more important are more distant future rewards.

In RL, the agent learns how to maximize the incoming reward by developing an action-value function $Q(\cdot,\cdot)$ (or a state value function $V(\cdot)$) that maps state-action pairs (or states) into the corresponding expected payoff value (Equation 1).

2.1 Q-learning

The Q-learning algorithm [9] iteratively approximates the optimal action-value function Q^*, which maps all state-action pairs to the associated expected payoff. Usually, Q^* is approximated by a tabular state-action representation often

referred to as *tabular Q-learning.* At time step t, when the agent senses the environment to be in state s_t, and receives reward r_t for performing a_{t-1} in state s_{t-1}, the entry $Q(s_{t-1}, a_{t-1})$, is updated according to the formula:

$$Q(s_{t-1}, a_{t-1}) \leftarrow Q(s_{t-1}, a_{t-1}) + \beta(r_t + \gamma \max_{a \in A} Q(s_t, a) - Q(s_{t-1}, a_{t-1}))$$

where β is the *learning rate* ($0 \leq \beta \leq 1$). Given a RL problem modeled as an MDP, under adequate hypotheses, Q-learning converges with probability one to the optimal action-value function Q^*.

2.2 Q-learning with Gradient Descent

Tabular Q-learning is simple and easy to implement but it is infeasible for larger problems because the size of the Q-table (which is $|S| \times |A|$) grows exponentially in the problem dimensions. To cope with this complexity, approximation methods need to be used. Particularly, a generalized representation of the optimal action-value function Q^* needs to be learned from a limited number of experiences. In RL, generalization is usually implemented by function approximation techniques. Often, gradient descent techniques are used to build a good approximation of function Q^* from online experience.

When applying gradient descent to approximate Q^* online, we are actually trying to minimize the error between the desired payoff value associated with the current state-action pair (estimated by $r + \gamma \max_{a \in A} Q(s_t, a)$) and the current corresponding payoff estimate $Q(s_{t-1}, a_{t-1})$. If the function approximator is parameterized by a weight matrix W, the change Δw for each weight w is

$$\Delta w = \beta(r + \gamma \max_{a \in A} Q(s_t, a) - Q(s_{t-1}, a_{t-1})) \frac{\partial Q(s_{t-1}, a_{t-1})}{\partial w},$$

where β is the learning rate and γ is the discount factor [14,15,13]. It can be seen, that the weight update depends both (i) on the difference between the desired value and the current value associated with the current state-action pair and (ii) on the gradient component represented by the partial derivate of the current payoff value with respect to the weight. The gradient component essentially adjusts the weight update with respect to its relative contribution to the Q-value estimate. Function approximation techniques that update their weights according to the equation above are called *direct algorithms.*

While tabular RL methods can be guaranteed to converge, function approximation methods based on direct algorithms have been shown to be fast but often unstable [14,15]. To improve the convergence of function approximation techniques in RL applications, another class of techniques, namely *residual gradient algorithms,* have been developed [15]. Residual gradient algorithms are slower but more stable than direct algorithms and, most importantly, they can be guaranteed to converge under adequate assumptions. Residual algorithms extend direct gradient descent approaches by adjusting the gradient of the current state with an estimate of the effect of the weight change on the successor state.

The weight update Δw for Q-learning implemented with a *residual* approach becomes the following:

$$\Delta w = \beta(r + \gamma \max_{a \in A} Q(s_t, a) - Q(s_{t-1}, a_{t-1})) \left[\frac{\partial Q(s_{t-1}, a_{t-1})}{\partial w} - \phi\gamma \frac{\partial}{\partial w} \left(\max_{a \in A} Q(s_t, a) \right) \right], \quad (2)$$

where the partial derivative $\frac{\partial}{\partial w}(\max_{a \in A} Q(s_t, a))$ estimates the effect that the current modifications of the weight have on the value of the next state. Note that since this adjustment involves the next state, the discount factor γ must also be taken into account. Parameter *phi* weighs the degree of influence of the next state on the update in the current state.

With this RL knowledge in mind, we now turn to the XCS classifier system investigating how XCS approximates the optimal Q-function Q^* and how (residual) gradient-based updates can be incorporated into XCS and into LCSs, in general.

3 XCS in Brief

XCS [5] is essentially a RL method in which generalization is obtained through the evolution of a *population* of condition-action-prediction rules *(classifiers).* With respect to the usual RL settings in XCS, (i) the population of classifiers approximates the optimal action-value function Q^* and (ii) the classifiers and their parameters roughly correspond to the weight matrix W that is used in function approximation approaches to generalize over the space of possible solutions. This section gives a short introduction to XCS. A detailed algorithmic description can be found in [16].

In XCS, classifiers consist of a condition, an action, and four main parameters: (i) the prediction p estimates the average payoff that the system expects when the classifier is used; (ii) the prediction error ε estimates the average absolute error of the prediction p; (iii) the fitness F estimates the average relative accuracy of the payoff prediction given by p; and (iv) the numerosity *num* indicates how many copies of classifiers with the same condition and the same action are present in the population.

At each iteration, XCS builds a *match set* [M] containing the classifiers in the population [P] whose condition matches the current sensory inputs; if [M] contains less than θ_{nma} actions, *covering* takes place and creates a new classifier that matches the current inputs and has an unrepresented action. For each possible action a_i in [M], XCS computes the *system prediction* $P(a_i)$ which estimates the payoff that XCS expects if action a_i is performed. The *system prediction* is computed as the fitness weighted average of the predictions of classifiers in [M], $cl \in [M]$, which advocate action a_i (i.e., $cl.a = a_i$):

$$P(a_i) = \frac{\sum_{cl_k \in [M]|_{a_i}} p_k \times F_k}{\sum_{cl_k \in [M]|_{a_i}} F_k}, \quad (3)$$

where $[M]|_{a_i}$ represents the subset of classifiers in $[M]$ with action a_i, p_k refers to the prediction of classifier cl_k, and F_k refers to the fitness of classifier cl_k. Next, XCS selects an action using, e.g., an epsilon-greedy action selection mechanism [13] based on the system prediction values $P(a_i)$. The classifiers in [M] which advocate the selected action form the current *action set* [A]. The selected action is performed in the environment, and a scalar reward r is returned to XCS together with a new input configuration.

After the reward r is received and the next match set $[M]$ is formed with respect to the resulting sensory input, the prediction value P is computed as follows.

$$P = r + \gamma \max_{a \in A} P(a). \tag{4}$$

Next, the parameters of the classifiers in [A] are updated in the following order [16]: prediction, prediction error, and finally fitness. Prediction p and prediction error ε are updated with learning rate β $(0 \leq \beta \leq 1)$,

$$p \leftarrow p + \beta(P - p). \tag{5}$$

$$\varepsilon \leftarrow \varepsilon + \beta(|P - p| - \varepsilon) \tag{6}$$

Based on the current prediction error, fitness is updated. The update consists of three steps successively determining *raw accuracy* κ, *relative accuracy* κ', and new fitness F:

$$\kappa = \begin{cases} 1 & \text{if } \varepsilon \leq \varepsilon_0 \\ \alpha(\varepsilon/\varepsilon_0)^{-\nu} & \text{otherwise.} \end{cases} \tag{7}$$

$$\kappa' = \frac{(\kappa \times num)}{\sum_{cl \in [A]}(cl.\kappa \times cl.num)}, \tag{8}$$

$$F \leftarrow F + \beta(\kappa' - F) \tag{9}$$

Parameter ε_0 $(\varepsilon_0 > 0)$ specifies the threshold that determines to what extent prediction errors are accepted; α $(0 < \alpha < 1)$ causes a strong distinction between accurate and inaccurate classifiers; ν $(\nu > 0)$ and ε_0 determine the steepness of the slope used to calculate classifier accuracy; $cl.\kappa$ is the raw accuracy of classifier *cl; cl.num is* the numerosity of classifier *cl.*

On a regular basis depending on the parameter θ_{GA}, a genetic algorithm (GA) is applied to classifiers in [A]. The GA selects two classifiers with probability *proportional to their fitness,* copies them, and performs crossover with probability χ; each allele is mutated with probability μ . The resulting offspring are inserted into the population and two classifiers are deleted from the whole population to keep the population size constant.

4 XCS with Gradient Descent

In general, XCS uses Q-learning techniques but can also be compared to a function approximation mechanism. In this section, we analyze the similarities between, tabular Q-learning, Q-learning with gradient descent, and XCS. We show

how to fuse the two capabilities adding gradient descent to XCS's parameter estimation mechanism.

4.1 Q-value Estimations and Update Mechanisms

In Section 2.1 we saw that tabular Q-learning iteratively approximates the Q-table entries using the difference between estimated and experienced reward signal to adjust the estimate:

$$Q(s_{t-1}, a_{t-1}) \leftarrow Q(s_{t-1}, a_{t-1}) + \beta(r + \gamma \max_{a \in A} Q(s_t, a) - Q(s_{t-1}, a_{t-1})). \quad (10)$$

As seen in Section 2.2, in a function approximation approach the Q-table is approximated by a weight matrix. Using the direct *(gradient descent)* approach, each weight w in the matrix W is modified by the quantity Δw:

$$\Delta w = \beta(r + \gamma \max_{a \in A} Q(s_t, a) - Q(s_{t-1}, a_{t-1})) \frac{\partial Q(s_{t-1}, a_{t-1})}{\partial w}, \quad (11)$$

where the gradient component $\frac{\partial Q(s_{t-1}, a_{t-1})}{\partial w}$ is used to guide the weight update.

As seen above, XCS exploits a modification of Q-learning, updating at each time step t, given current input s_t and current reward r, the prediction estimates of each classifier in the action set $[A]_{t-1}$ of the previous time step by

$$p \leftarrow p + \beta(r + \gamma \max_{a \in [A]} P(a) - p). \quad (12)$$

We can note that while tabular Q-learning only updates one value at each learning iteration (updating $Q(s_{t-1}, a_{t-1})$), XCS updates all classifiers in the action set $[A]_{t-1}$. In fact, each position in the Q-table is represented by the corresponding prediction array value (Equation 3). Comparing the weight update for gradient descent (Equation 11) and the update for classifier predictions (Equation 12) we note that in the latter no term plays the role of the gradient. Classifier prediction update for XCS was directly inspired by *tabular* Q-learning [5]. Until now, gradient approaches have not been considered in XCS.

4.2 Adding Gradient Descent to XCS

To improve the learning capabilities of XCS we add gradient descent to the equation for the classifier prediction update in XCS. As noted above, the value of a specific state-action pair is represented by the system prediction $P(\cdot)$, which is computed as a fitness weighted average of classifier predictions (Equation 3). In general, learning classifier systems consider the rules that are active and combine their predictions (their *strength*) to obtain an overall estimate of the reward that should be expected. In this perspective, the classifier predictions play the role of the weights in function approximation approaches. The gradient component for a particular classifier cl_k in the to-be-updated action set $[A]_{t-1}$ can be estimated

by computing the partial derivate of $Q(s_{t-1}, a_{t-1})$ with respect to the prediction p_k of classifier cl_k:

$$\frac{\partial Q(s_{t-1}, a_{t-1})}{\partial w} = \frac{\partial}{\partial p_k}\left[\frac{\sum_{cl_j \in [A]_{t-1}} p_j F_j}{\sum_{cl_j \in [A]_{t-1}} F_j}\right] =$$

$$= \frac{1}{\sum_{cl_j \in [A]_{t-1}} F_j} \frac{\partial}{\partial p_k}\left[\sum_{cl_j \in [A]_{t-1}} p_j F_j\right] = \frac{F_k}{\sum_{cl_j \in [A]_{t-1}} F_j}. \quad (13)$$

Thus, for each classifier the gradient descent component corresponds to its relative contribution (measured by its current relative fitness) to the overall prediction estimate.

To include the gradient component in XCS's classifier prediction update mechanism, prediction p_k of each classifier $cl_k \in [A]_{t-1}$ is now updated using

$$p_k \leftarrow p_k + \beta(r + \gamma \max_{a \in A} P(a) - p_k)\frac{F_k}{\sum_{cl_j \in [A]_{t-1}} F_j}. \quad (14)$$

The other parameters are updated as usual (see Section 3). In the remainder of the paper we refer to the version of XCS with the prediction updated based on gradient descent as XCSG.

Due to the contribution-weighted gradient-based update, the estimate of the payoff surface, that is, the approximation of the optimal action-value function Q^*, becomes more reliable. As a side effect, the evolutionary component of XCS can work more effectively since the classifier parameter estimates are more accurate. The next section validates this supposition.

5 Experimental Validation

To evaluate XCSG, we compare its performance to XCS in several typically used maze environments. The experimental setup distinguishes between learning problems and test problems as has been usually done in the literature [5]. In learning problems, the system selects actions randomly from those represented in the match set and applies all learning mechanisms. In test problems, the system always selects the action with highest prediction and applies parameter updates and covering only. Learning problems and test problems alternate. In the investigated multistep problems, performance is computed as the average number of steps needed to reach the goal position averaged over the last 50 test problems. If the goal is still not reached after the execution of 1500 steps in one problem, the next problem begins. All results reported in this paper are averaged over 20 experiments.

5.1 The `Woods1` Environment

First we apply XCSG to `Woods1` (shown in Figure 1a) and compare XCSG's performance with that of XCS. Since `Woods1` is very simple we do not expect

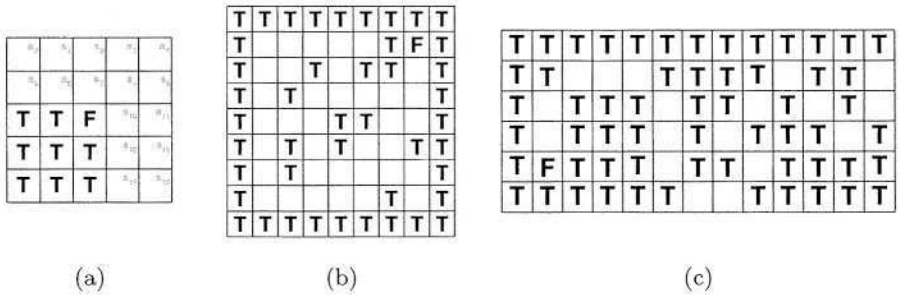


Fig. 1. Environments (a) `Woods1`; (b) `Maze6`; (c) `Woods14`. `Woods1` is a toroid. Obstacles (trees) are denoted by *T*. The goal (food) is denoted by *F*. Perceived are the eight neighboring positions starting north and coding clockwise. Actions are possible to the eight neighboring positions. An action that leads to a position with an obstacle has no effect.

much difference in the performance of the two systems[1]. Figure 2 reports the performance of XCS (dashed line) and XCSG (solid line); as expected there is almost no difference between the two versions, the problem is very simple, and both algorithms reach optimality. A closer look at the very beginning of the experiments shows that XCS learns actually somewhat faster than XCSG. Since the gradient approach effectively decreases the overall update rate of the reward prediction, learning is slightly delayed.

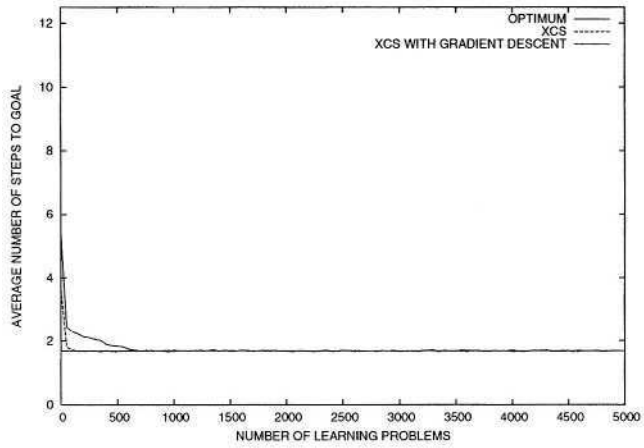


Fig. 2. The performance of XCS (dashed line) and XCSG (solid line) in `Woods1`.

[1] In `Woods1`, parameters were set as follows: $N = 1600$, $P_{\#} = 0.6$, β=0.2, γ=0.7, χ=0.8, μ=0.04, θ_{nma}=8, $p_{\text{explr}} = 1.0$, θ_{GA}=25, ε_0=10, θ_{del}=20. Subsumption is not applied.

5.2 The Maze6 Environment

The Maze6 environment (shown in Figure 1b) was shown to be a hard problem for XCS [7]. Figure 3 reports the performance of XCS (dashed line) and XCSG (dotted line) in Maze6 [2]; note that XCS performance is quite far from the optimum while XCSG reaches the optimum rapidly and stably. The analysis of single runs shows that XCS cannot reach optimal performance in many runs. In contrast, XCSG always reaches full optimality. This suggests that the improvement in XCS performance provided by gradient descent becomes more and more relevant as the problem complexity increases.

To assure that the gradient approach is effective not only due to the reduced update rate of the reward prediction estimate, we also compare the performance of XCSG with that of a modified version of XCS in which the update of classifier prediction is based on a learning rate β_P smaller than the actual learning rate β used for the update of the classifier prediction error and classifier fitness. Figure 3 reports the performance of XCS with $\beta_P = 0.01$ in Maze6 (solid line). Albeit a smaller learning rate β_P does improve XCS performance, the overall improvement is far from that of XCSG. Due to the distinct update of the reward prediction measure in the gradient approach, the reward prediction of overgeneral (and thus low-fitness) classifiers fluctuates less preventing prediction overestimations and error underestimations.

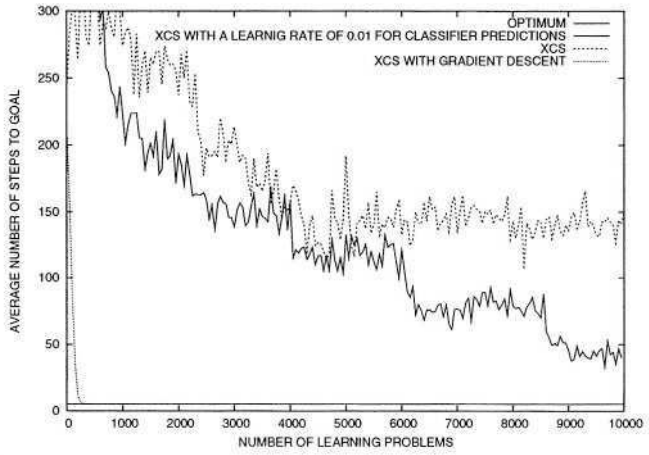


Fig. 3. The performance of XCS with $\beta_P = 0.01$ (solid line), XCS (upper dashed line), and XCSG (lower dashed line) in Maze6.

[2] Parameter setting in the Maze6 environment: N = 3000, $P_{\#}$ = 0.3, β=0.2, γ=0.7, χ=0.8, μ=0.01, θ_{nma}=8, p_{explr} = 1.0, θ_{GA}=100, ε_0=1, θ_{del}=20. Subsumption is not applied.

5.3 The Woods14 Environment

Woods14 is a particular hard problem for XCS because a long chain of classifiers needs to be evolved and maintained to reach the goal optimally [8]. Due to the large number of steps to the food, the prediction error ε_0 must be set small enough to allow XCS to distinguish among small payoff values.[3] To speed up the experiments we reduced the maximum number of steps per problem to 500 steps.

Figure 4 reports the performances of XCS (dashed line) and XCSG (solid line) in Woods14 [4]. XCS does not reach even near optimal performance in any of the 20 runs. In contrast, XCSG reaches the optimum rapidly and stably. The analysis of single runs shows that in 18 of the 20 runs XCSG was able to reach optimal performance; in the remaining two runs, XCSG was able to learn the optimal policy for Woods14 for all the positions except the last one at the end of the corridor, that is, the position farthest from the goal.

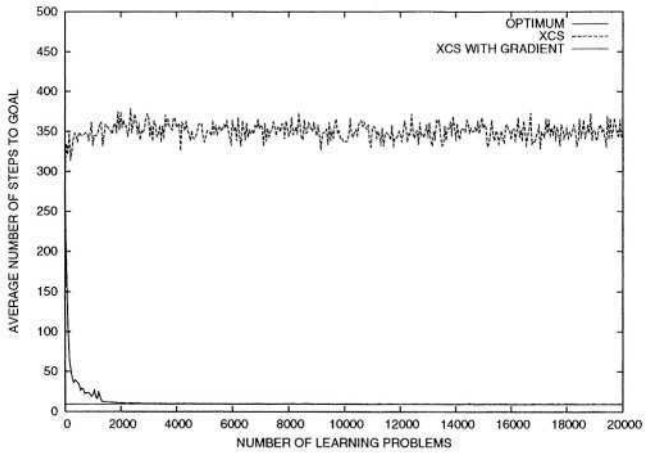


Fig. 4. The performance of XCSG (solid line) and XCS (dashed line) in Woods 14 when $N = 4000$.

6 Summary and Conclusions

This paper showed that the addition of a gradient-based update mechanism in XCS results in more stable and reliable learning. Hard multistep problems, such as Maze6, which allows only few generalizations, and Woods14, which requires a

[3] To be able to reach optimal performance XCS must be able to distinguish between two payoff values: $1000\gamma^{18}$ and $1000\gamma^{19}$; accordingly, ε_0 must be smaller than $1000(1-\gamma)\gamma^{18}/2 = 0.245$.

[4] Parameters were set as follows in Woods14: $N = 4000$, $P_{\#} = 0.3$, β=0.2, γ=0.7, χ=0.8, μ=0.01, θ_{nma}=8, $p_{\text{explr}} = 0.3$, θ_{GA}=400, ε_0=0.05, θ_{del}=20. Subsumption was not applied.

long classifier chain, were both solved reliably. Further investigations with various population sizes N and other GA thresholds θ_{GA} confirmed the robustness of the mechanism [17].

Interestingly, in the investigated problems, stability was reached without any residual update addition as it is necessary in neural-network type Q-value function approximators (see Section 2.2). Residual methods were added in [17] but did not improve XCS's performance in the investigated problems.

XCS, and learning classifier systems in general, appear to be somewhat in between tabular RL mechanisms and neural-based RL mechanisms. Previous comparison to neural-network learning mechanisms have emphasized the co-adaptivity in LCSs and the genetic learning component [18]. Our investigations show that even more importantly the representation significantly differs: XCS (as our exemplar system) estimates reward not only by one rule, as in a tabular learning approach, and also not by all rules, as in a neural-network approach, but by a subset of matching rules. Thus, residual methods may not be necessary in LCSs because only the matching subset of classifiers is used for the estimation of a Q-value. The distinction of successive Q-values is realized by classifier conditions instead of classifier weights. Since the evolutionary component evolves the conditions, the distinction of successive Q-values is realized by the evolutionary component and not by the reinforcement component in LCSs.

The successful application of neural-network type mechanisms suggests further comparisons between XCS, LCSs, and neural-network learning and function approximation mechanisms. Classifier conditions, evolved by an evolutionary mechanism, effectively identify independent problem subspaces. This property appears to be highly similar to the development of kernel methods in neural-networks, in which different kernel types result in different spatial separations. The extension of LCSs to kernel-based condition parts appears to be an interesting future research direction. The real-value extension of XCS may be seen as a first step in this direction [1,12]. Future research will show in which problems the more explicit, rule-based spatial problem separation in conjunction with an evolutionary learning component in LCSs may be advantageous in comparison with neural-network type learning mechanisms.

Acknowledgments. We are grateful to Xavier Llorà, Kei Onishi, Kumara Sastry, and the whole IlliGAL lab for their help and the useful discussions. The work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant F49620-03-1-0129. The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon. Additional funding from the German research foundation (DFG) under grant DFG HO1301/4-3 is acknowledged. Additional support from the Computational Science and Engineering graduate option program (CSE) at the University of Illinois at Urbana-Champaign is acknowledged. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research or the U.S. Government.

References

1. Wilson, S.W.: Get real! XCS with continuous-valued inputs. In Lanzi, P.L., Stolzmann, W., Wilson, S.W., eds.: Learning classifier systems: From foundations to applications. Springer-Verlag, Berlin Heidelberg (2000) 209–219
2. Bernadó, E., Llorà, X., Garrell, J.M.: XCS and GALE: A comparative study of two learning classifier systems and six other learning algorithms on classification tasks. In Lanzi, P.L., Stolzmann, W., Wilson, S.W., eds.: Advances in learning classifier systems: Fourth international workshop, IWLCS 2001. Springer-Verlag, Berlin Heidelberg (2002) 115–132
3. Lanzi, P.L.: Mining interesting knowledge from data with the XCS classifier system. Proceedings of the Third Genetic and Evolutionary Computation Conference (GECCO-2001) (2001) 958–965
4. Dixon, P.W., Corne, D.W., Oates, M.J.: A preliminary investigation of modified XCS as a generic data mining tool. In Lanzi, P.L., Stolzmann, W., Wilson, S.W., eds.: Advances in learning classifier systems: Fourth international workshop, IWLCS 2001. Springer-Verlag, Berlin Heidelberg (2002) 133–150
5. Wilson, S.W.: Classifier fitness based on accuracy. Evolutionary Computation **3** (1995) 149–175
6. Wilson, S.W.: Generalization in the XCS classifier system. Genetic Programming 1998: Proceedings of the Third Annual Conference (1998) 665–674
7. Lanzi, P.L.: An analysis of generalization in the XCS classifier system. Evolutionary Computation **7** (1999) 125–149
8. Barry, A.M.: The stability of long action chains in XCS. Journal of Soft Computing **6** (2002) 183–199
9. Watkins, C.J.C.H.: Learning from Delayed Rewards. PhD thesis, King's College, Cambridge, UK (1989)
10. Wilson, S.W.: ZCS: A zeroth level classifier system. Evolutionary Computation **2** (1994) 1–18
11. Lanzi, P.L.: Learning classifier systems from a reinforcement learning perspective. Soft Computing: A Fusion of Foundations, Methodologies and Applications **6** (2002)
12. Wilson, S.W.: Function approximation with a classifier system. Proceedings of the Third Genetic and Evolutionary Computation Conference (GECCO-2001) (2001) 974–981
13. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT Press, Cambridge, MA (1998)
14. Baird, L.C.: Residual algorithms: Reinforcement learning with function approximation. Machine Learning: Proceedings of the Twelfth International Conference (1995)
15. Baird, L.C.: Reinforcement Learning Through Gradient Descent. PhD thesis, School of Computer Science. Carnegie Mellon University, Pittsburgh, PA 15213 (1999)
16. Butz, M.V., Wilson, S.W.: An algorithmic description of XCS. Soft Computing **6** (2002) 144–153
17. Butz, M.V., Goldberg, D.E., Lanzi, P.L.: Gradient Descent Methods in Learning Classifier Systems. IlliGAL report 2003028, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign (2003)
18. Smith, R.E., Cribbs, H.B.: Is a learning classifier system a type of neural network? Evolutionary Computation **2** (1994) 19–36

System Level Hardware–Software Design Exploration with XCS

Fabrizio Ferrandi, Pier Luca Lanzi, and Donatella Sciuto

Dipartimento di Elettronica e Informazione
Politecnico di Milano
Piazza Leonardo da Vinci 32
I-20133 Milano, Italy
{ferrandi,lanzi,sciuto}@elet.polimi.it

Abstract. The current trend in Embedded Systems (ES) design is moving towards the integration of increasingly complex applications on a single chip. An Embedded System has to satisfy both performance constraints and cost limits; it is composed of both dedicated elements, i.e. hardware (HW) components, and programmable units, i.e. software (SW) components, Hardware (HW) and software (SW) components have to interact with each other for accomplishing a specific task. One of the aims of codesign is to support the exploration of the most significant architectural alternatives in terms of decomposition between hardware (HW) and software (SW) components. In this paper, we propose a novel approach to support the exploration of feasible hardware-software (HW-SW) configurations. The approach exploits the learning classifier system XCS both to identify existing relationships among the system components and to support HW-SW partitioning decisions. We validate the approach by applying it to the design of a Digital Sound Spatializer.

1 Introduction

Embedded Systems (ES) design has undergone several deep transformations during the past few years. The most relevant concern is the raising of the abstraction level of the system modeling, made possible by recent evolutions in the description languages. Another trend in the evolution of the design formalisms is the use, as basis for the system design specification, of typical software languages such as C and C++. The introduction of new description languages for the embedded systems applications such as `SystemC`, poses new problems, mainly due to their ability to express system models at levels of abstractions not possible before. The choice of an architecture, is one of the important steps in design. An architecture is defined by a collection of components which can be either programmable (SW components), re-configurable or customized (HW components), Hardware (HW) and software (SW) components have to interact with each other for accomplishing a specific task. The aims of hardware-software (HW-SW) codesign is to support the exploration of the most significant architectural alternatives in terms of decomposition between hardware (HW) and software (SW) components.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 763–773, 2004.

We present a new methodology based on the learning classifier system XCS [11] to support the HW-SW codesign of embedded systems. The method we propose extract interesting design patterns from raw HW-SW codesign data. These patterns are then exploited by designers to explore the space of the possible partitionings. The method is based on a formal approach to the modeling of the HW-SW codesign problem in which the embedded system is functionally decomposed at the highest possible (functional) level of abstraction. The formal model is used to build a simulator which can be used to obtain an accurate estimate of partitioning costs. The simulator is used to obtain data about the cost function associated to the design of the embedded system. Finally, the learning classifier system XCS is trained with the data obtained by simulator to extract interesting design patterns expressed as condition-action rules which associate cost-patterns to design decisions. While the focus of this paper is mainly the application of XCS to the system level HW-SW design exploration, more details concerning the formal model used to represent the HW-SW design problem can be found in [9]; in addition [3] provides an analysis regarding the use of neural networks to approximate the cost function so to speed up the mining of HW-SW codesign data. The paper is organized as follows. In Section 2 we provide a brief overview of the overall approach we propose. In Section 3 we present a case study involving the design of a Digital Sound Spatializer. In Section 4 we overview the results produced by XCS from the design data. Section 6 ends the paper providing some consideration regarding the results and the future research directions.

2 Proposed Approach

We now shortly overview the codesign approach, we introduced in [9], that we use to generate the data used for the analysis in this paper. We refer the interested reader to the original paper [9] for a detailed description of the overall codesign approach. The Hardware-Software Codesign process is organized into three main steps that lead to a final partitioning optimized with respect to the given metrics [9]. First, the software performance of the different functional components is estimated from a first description of the application which, nowadays, is usually provided in terms of *C*-like programming languages. Then, the global and local system performance are estimated in terms of hardware description, communication interfaces, and mixed hardware-software architecture. In the final step, the above information is used to guide the algorithm which will provide the optimal partition.

2.1 Software Performance Estimation

This first step aims at giving both a characterization of the timing behavior of the system when specified in some programming language (nowadays `C` and `C++` are the most widely used) and a mean to dynamically determine, via simulation, the best granularity at which the partitioning process is to be faced. The choice of the best granularity, is of paramount importance, as it defines how to

decompose the system into sub-parts, which will then be the units over which the final partitioning phase will reason for defining the final system architecture. According to the approach we introduced in [9] this problem is tackled by starting at the *process* level of granularity, which is a sort of meeting in the middle, between the whole system and the operation level. To accomplish this, the target micro-processor is selected and the executable code of the application is produced. Then a parsing tool identifies in the code, by creating a Data Flow Graph (DFG), all the data dependencies to be used for classifying the code fragments, based on which to dynamically identify the mathematical relations that describe execution time as a function of the input data. After having found these mathematical relations, for all the processes involved in the system, the designer, according to the timing system constraints, is able to understand if some of these processes are time critical for the application. For such processes, a finer grain granularity is obtained, by decomposing them in smaller sub-parts, which will be computationally manageable. The outputs of this phase are (i) the identification of the granularity for analyzing the system, based on which all the subsequent phases will be carried on, and, (ii) a characterization in terms of execution time of all the system subparts seen as software modules. This characterization is employed in the final simulation of the system seen as a mixed hardware-software architecture.

2.2 Hardware and Communication Performance Estimation

Given the software performance characterization of the system, and the selected granularity, in this second step we estimate both local and global system performance taking into account the possibility of realizing the different components both in hardware and in software. For gaining a fast estimation on hardware timing, the approach we introduced in [9] relies on the information collected by the parsing tool at the beginning of the software estimation phase. By considering the Data Flow Graph (DFG) of each system component identified by the dynamically determined granularity, we apply to each of them an unoptimized version of an unconstrained scheduling algorithm [6], which gives a characterization of hardware performance. The same approach is applied to model the communication among different classes of components (see [9] for details).

2.3 The Simulator

After having characterized all the components and all the possible communication channels a simulator is built. This is used to simulate several instances of the embedded system architecture, which differ one from the other on the hardware or software characterization of the single sub-parts. Each instance also carries the correct model of communication according to its specific mix of hardware and software parts. The simulator is written in `SystemC` and it is the exact translation of the formal model used to describe the target embedded system. It exploits both the local characterizations given by the previous software and hardware estimation phases and the communication model to compute *global*

system performance when instantiating different possible combinations of hardware and software modules. At a very high level of abstraction, the simulator takes as input a binary string representing a partitioning and returns a cost. This mapping is used in [9] to support the search for the final partitioning and in [3] we show how we can successfully approximate the whole map by means of neural networks. In contrast, in the work presented here, we use XCS to analyze the input-output mapping obtained from the simulator so as to identify interesting patterns which characterize the embedded system.

2.4 Exploration of the Partitioning Space with XCS

The simulator can then be used (i) to extract some data regarding the cost of HW-SW configurations and (ii) to mine such data in search of interesting information regarding the system's criticalies for a given class of ES's. For this purpose, we view the ES model, implemented by the simulator, as a black box with inputs $\overline{x}$, representing a certain partitioning and an output y, representing the cost of the (input) partitioning. The model *can be applied* to an example to obtain a cost; but we basically assume that the model is unknown in that we do not know what is the target function f_t which the model implement. For a certain problem a set of examples E is collected from the simulator; each example is a pair $\langle \overline{x}_j, y_j \rangle$, where $\overline{x}_j$ is a possible input configuration (i.e., partitioning), y_j is the cost (performance) that the model predicts for the partitioning $\overline{x}_j$. Starting from the set of examples E, we can apply statistical and data mining techniques to extract interesting relations among input-output configurations. For instance, in [9] we have applied Principal Component Analysis (PCA) and Linear Regression (LR) to extract basic *linear* relations among the different system components; in [3] we focused on the use of neural networks to approximate the cost function from a limited number of examples. Here, we employ a different approach in which the learning classifier system XCS [11] is applied for mining interesting, highly non-linear patterns, from the data obtained from the simulator. For this purpose, XCS is applied as usually done in any other single-step problem. An experiment consists of a number of problems that XCS must solve. For each problem, an input partitioning, represented by a binary string (like those in Table 2) is presented to XCS. Classifier conditions are represented as strings (one for each component) over the ternary alphabet {0,1,#}; the don't care symbol # means that the corresponding position in the classifier condition can either either 0 and 1, i.e., the corresponding component can be implemented either in hardware or software. There is one action for each component identified by the system-level description; for instance, in the example discussed in the previous section, there are nine possible actions, since there are nine possible components. Based on the current input partitioning, XCS suggests an action which identifies a possible modification to the current partitioning through the flip of a bit in the current hardware-software configuration. The action is *performed,* and the current partitioning is modified. As a result, XCS receives a reward computed as the difference between (i) the cost of the current partitioning and (ii) the cost of the new partitioning obtained through the suggested

modification. Thus XCS will receive a positive reward if the suggested action corresponds to a decrease in the cost of the partitioning, a negative reward if the suggested action corresponds to an increase in the cost of the partitioning. The performance is measured as the error between the actual reward received as effect of the proposed action, and the reward that XCS estimated. Therefore XCS learns to predict how the modifications of the input partitioning will influence the target cost function. More precisely, since our cost function is simply the estimated execution time, XCS learns to prediction how the modifications of the input partitioning will influence the overall execution time. Note that, according to this settings, XCS is used as a step-wise function approximator as firstly done by Wilson [12] instead of being used to learn an optimal behavior like in [11].

3 Case Study

The application chosen as a case study is a Digital Sound Spatializer, which is significant as it encompasses all the characteristics of a typical embedded system and it is simple enough to allow a straightforward illustration of the methodology. A sound spatializer is a machine, be it analog, digital or hybrid, which takes as its input a sound which is acquired by one or more sound sources: e.g., from a musical instrument or a singer, which is usually recorded with high–quality noise–reduction microphones. Its output is the input sound as it would be perceived by someone in a room, in which the sound is supposed to be played. The code of the application has been written in C++ for giving its software characterization needed for performance estimation and because C++ made it easy to convert it in `SystemC`, which is the language used for the global simulation. The choice of `SystemC` is due to its suitable characteristics both of supporting channels description at a high level of abstraction and because it is undoubtedly the leading specification language for Embedded System design. The target microprocessor we choose for testing the software behavior is the Xilinx MicroBlaze, which is a 32 bit RISC soft-processor. The choice of this specific processor is due to its great suitability for being integrated with the Xilinx Virtex II Pro FPGA, which is the final target architecture for the deployed version of the application. For simulating the acquisition phase of the audio stream and the output of the reverberated sound we used an audio file and sampled it at 44.1 KHz extracting a 16 bits sample; once the sample has been processed by the application we wrote the resulting sample on an output file. The code describing the application is composed of 9 processes, which describe all the different components of the Digital Audio Spatializer: the Multitap Delay line, which simulates both the direct sound and the discrete echoes, the Multitap Delay mixer which scales the output of the Multitap Delay line, the Sound Source, which is one of the processes modeling the room where the sound is diffused, the Walls (which for the final partitioning are considered as 4 different processes), the Listener, who receives the audio sample modified by the four walls, reassembles and scales it and the Output Mixer process, which receives the processed samples from the delay

line and from the listener and reassembles it applying to it the final distortion according to the given parameters. The analysis performed on this code shows that, as far as the data dependencies are concerned, the only one involved is the simplest, that is the execution time is *linear* in the number of samples, which is easily understandable if one thinks that the emulation of the room effects on the sound practically are applying a delay and scaling it according to a dispersion factor. According to the results of this phase and the performance needs of the application, which has to process one audio sample per sampling interval, we can state that the granularity of processes is suitable for the overall system, so that we can carry on the subsequent analysis keeping it unchanged. After having found the local information on the software behavior of each process, we turned to the more complex issue of adapting the code to fit in a model allowing the channel insertion with the proper delays and the possibility of instantiating all the possible mixed HW-SW configurations taking into account the previous information on the software performance. To compute the communication delays we referred to a microprocessor frequency f_{sw} of 50 MHz and a hardware frequency of the target *FPGA* of 100 MHz. We considered a HW-HW communication via dedicated parallel lines. Considering the hardware frequency of 100 MHz and the fact that a complete data exchange operation takes 1 clock cycle to execute, we get: $t_{hh1} = 10\,ns$. For the HW-SW communication model, we cannot derive a single value, as from instance to instance the number of processes communicating via *DMA* can change; we then inserted the mathematical equation relative to the computation of t_{bus} and t_{dma} in the simulator, so that, once the single instance was given and the corresponding transition labels identified accordingly, we could compute the true delay value. As for the queue parameters other than the N number of customers, we used for their derivation a bus bandwidth of 1 $\frac{Mb}{s}$, obtaining,

$$\mu = \frac{\#bits}{B} = \frac{16b}{10^6 b/s} = 160\,ms.$$

Considering the sample rate of 44.1 KHz, we computed λ (the average interarrival time of the customers), as:

$$\lambda = \frac{1}{44.1KHz} = 227\,ms.$$

Finally, the numerical value of ρ, the traffic intensity, is given by:

$$\rho = \frac{\lambda}{\mu} = \frac{227}{160} = 1.42.$$

The same reasoning applies to the SW-SW communication, as also here the number of processes accessing the shared memory area changes instance by instance, so we inserted in the simulator the mathematical relation for computing t_{lock}, where all the parameters are computed as shown before, with the only difference that, according to the slower frequency of the microprocessor, we used as bandwidth $B = 500\ \frac{Kb}{s}$, so $\mu = 320\,ms$ and $\rho = 0.71$. We show in Table 1 the

Table 1. Local Characterization of the Processes Behavior

Process:	Delay	Del. Mixer	Source	Wall	Listener	Out Mixer
HW ex-time	$2.15 * 10^8$	$5.864 * 10^8$	$2.552 * 10^8$	$1.82 * 10^8$	$7.197 * 10^7$	$6.642 * 10^6$
SW ex-time	$1.0736 * 10^9$	$1.076 * 10^9$	$1.042 * 10^9$	$3.79 * 10^8$	$8.605 * 10^7$	$5.845 * 10^7$

results of the local behavior of the modules both as HW and as SW, where the execution time is expressed in *ns* per *KB* of samples.

After having completed the local characterization, we run a complete simulation of all the possible system instances, which are, according to the 9 different processes we consider, $2^9 = 512$. This simulation took 3 hours and half to complete on an Intel, PIV, 1.7 GHz, 1 GB RAM. This simulation takes into account the proper communication delays by inserting, instance by instance, the correct values in the `WAIT` instructions inserted in the system channels by the simulator. We show a sample output of the simulator in Table 2, where the global execution time is expressed in *ns* per *KB* of samples.

Table 2. Global Characterization of the Application Behavior

System Instance	Global Execution Time
1 1 1 1 0 0 0 1 1	4449011566
1 1 1 1 0 0 1 0 0	4585806280
1 1 1 1 0 0 1 0 1	4634142896
1 1 1 1 0 0 1 1 0	4597492768
1 1 1 1 0 0 1 1 1	4645829384
1 1 1 1 0 1 0 0 0	4585806280
1 1 1 1 0 1 0 0 1	4634142896
1 1 1 1 0 1 0 1 0	4633614640

In Table 2, each entry represents one HW-SW configuration, a one indicates that the process is implemented in software, a zero indicates that the process is implemented in hardware; the order of the processes is: sound source, multitap delay line, delay mixer, wall 1, wall 2, wall 3, wall 4, listener and output mixer.

4 The Results Produced by XCS

We apply the XCS classifier systems to mine interesting patterns from the cost function obtained by applying the codesign approach we described in Section 2 to the Digital Sound Spatializer we illustrated in Section 3. Classifier conditions are strings of 9 symbols (one for each component) over the ternary alphabet {0,1,#}; There are 9 possible classifier actions, numbered from 0 to 8, that

represent the change of the corresponding bit in the current input partitioning. The performance of XCS is measured as the error between the actual reward received as effect of the proposed action, and the reward that XCS estimated.

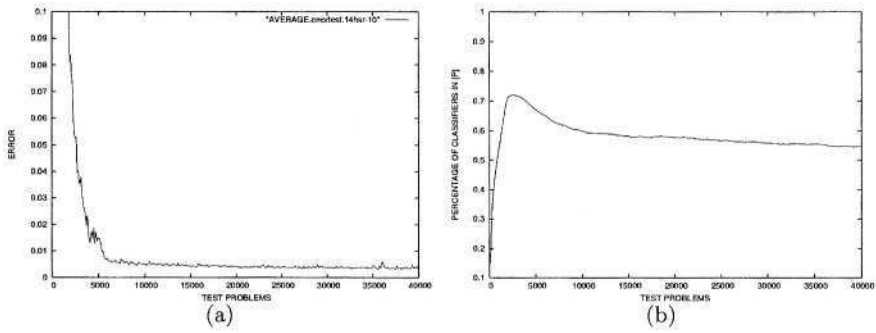


Fig. 1. XCS applied to the cost function for the Digital Sound Spatializer: (a) error on the prediction of the cost; (b) percentage of classifiers in the population. Curves are averages over 10 experiments.

Figure 1a reports the error over XCS prediction for the data derived for the Digital Sound Spatializer; population size N is 5000 classifiers and $\epsilon_0 = 10^{-4}$, while all the other parameters are set as usual (e.g., [11]). As the figure shows, XCS learns to predict quite accurately how the modifications in the current partitioning will affect the cost, i.e., the execution time. Figure 1b reports the percentage of classifiers in the population. Initially, the population rapidly grows while XCS is starting to learn, as the learning proceeds the population size shrinks and the number of classifiers in the population decreases showing that XCS is converging toward a minimal set of classifiers that represent some accurate piece of knowledge extracted from the cost function.

Table 3 reports some classifiers evolved during one of the runs depicted in Figure 1. The first classifier indicates that if component c_2 and component c_3 are implemented in software, then changing component c_2 to hardware (action is 2) will cause a reduction p in the cost (i.e., in the execution time) equal to 0.61, and that this prediction is affected by an absolute error (ϵ) of 1.384×10^{-4}. Note that the second and the third classifiers provide complementary information. The second classifier suggests that if component c_0 and c_1 are implemented in hardware, then changing component c_1 to software will cause *an increase of 0.76* in the cost (we remind the reader that a negative prediction means that the resulting partitioning has an higher cost). Conversely, the third classifier suggests that if component c_0 is implemented in hardware and component c_1 is implemented in software, the changing component c_1 to hardware (so to obtain a partitioning matched by the first classifier), will cause *a decrease of 0.76* in the cost. Note that the classifiers reported in Table 3 represent high level and accurate information about the cost function; all the classifiers in table 3 are

Table 3. Examples of classifiers evolved by XCS from the data obtained for the Digital Sound Spatializer: `Condition` identifies the classifier condition; `Action` identifies the classifier action; p is the classifier prediction which estimates how the partitioning cost will be modified by the corresponding action; ϵ is the prediction error which estimate how much accurate is the prediction p.

Condition	action	p	ϵ
##11##########	2	0.61	1.384e-04
00###########	1	-0.76	5.520e-03
01###########	1	0.76	5.292e-03
11###########	1	0.72	6.658e-05
0###010#######	4	-0.15	4.481e-03
1###100#######	4	0.16	4.335e-03
0###110#######	4	0.16	4.410e-03
######1#100###	8	0.15	4.563e-03
########0#101#	10	0.16	5.407e-03
######0#101###	8	1.52	3.923e-03
...	...	...	...

very general in that they apply to many partitioning (i.e., they have many don't care symbols); in addition they are very accurate since their prediction error ϵ is very small if compared to the prediction value. The evolved classifiers can be used to improve the designer understanding of the existing interactions among different system components, either as an effective support to the search of the best partitioning.

5 Related Work

The first significant work dates back to 1993 [4] and it represents one of the first approaches tackling the complete design of embedded applications. The system is described at the behavioral level by means of an appropriate specification language, whose underlying formal model is a Control Data Flow Graph (CDFG) and the chosen granularity is the operation. The operation delays (needed for performance estimation) are provided separately for hardware and software implementations, based on the type of hardware to be used and on the processor used to run the software. The partitioning algorithm focuses the attention on minimizing communication delays and measuring the partition effects on system performance, trying to devise a partition cost function that captures these properties by means of iterative heuristics. In [10] the problem of HW-SW partitioning is tackled by defining closeness metrics, that is a measure of the likelihood that two pieces of the specification should be implemented on the same system component. The objects are addressed at the procedural level of granularity, and the metrics also have been developed with this granularity in mind. The final algorithm used for finding the final partitioned architecture is based on N-way clustering. There are works (e.g., [2]) focused on solving the

partitioning problem according to iterative improvement heuristics, basically exploiting the Simulated Annealing and Tabu Search algorithms. In these works partitioning is performed at the loop level of granularity, with the aim of minimizing communication costs and enhancing parallelism. The partitioning algorithm takes into account simulation statistics, information from static analysis of the source specification and cost estimations, focusing on two main statistics: computational load and communication intensity. In [8], a hierarchical evolutionary approach to HW-SW partitioning is presented. The main characteristic of this work is to apply a hierarchical structure and dynamically determine the granularity of tasks and hardware modules to adaptively optimize the solution while keeping the search space as small as possible. Another work managing a flexible granularity in the partitioning phase is [5]. Recently, several approaches perform design exploration based on Genetic Algorithms (GAs) [1], [7], which seem particularly promising when dealing with parametrized systems to be tuned for final deployment. The approach is a combination of two phases: globally the authors use a parameter dependency model of the target parametrized specification to pre-prune non-optimal subspaces, while locally GAs are applied to discover Pareto-optimal configurations representing the range of performance trade-offs obtainable with parameters tuning.

6 Summary and Future Research Directions

We have presented a novel approach to face the design exploration of the architectural solutions of an embedded system specification based on Wilson's XCS. Our approach is based on the system-level decomposition of the target embedded system. The high-level description is exploited to develop a simulator to estimate the performance of possible hardware-software partitionings. The simulator is used to produce data, associating HW-SW partitionings to cost, which can be analyzed either using basic statistical techniques, as done in [9], either using learning classifier systems, as proposed here. In particular, in this paper we have applied XCS to the mining of HW-SW data from a Digital Sound Spatializer. Although the application is limited in complexity it is significant as (i) it encompasses all the characteristics of a typical embedded system and, most important, (ii) it is a real-world problem provided by a major company interested in the design of embedded system. The results reported here show that XCS can extract accurate rules which identify interesting design information. The results reported here have also been presented to researchers involved in the design of embedded systems for a main company. The comments we received so far are promising. Generally speaking, the rule-based representation of design patterns provided by learning classifier systems appear to be more intuitive than the information provided by the statistical techniques we discussed in [9]. For instance, from our interactions with field experts, we noted that the concept of *generalization* is more easily conveyed by means of rules than it is by using linear models. At the moment, we are applying the same approach on an-

other real-world application involving the design of an embedded mpeg codec component.

References

1. G. Ascia, V. Catania, and M. Palesi. Parameterized system design based on genetic algorithms. In *Proceedings of the Ninth International Workshop on Hardware/Software Codesign,* April 2001.
2. P. Eles, K. Kuchcinski, Z. Peng, and A. Doboli. System level hardware/software partitioning based on simulated annealing and tabu search. *Journal on Design Automation for Embedded Systems,* 2:5–32, 1997.
3. Fabrizio Ferrandi, Pier Luca Lanzi, and Donatella Sciuto. Mining Interesting Patterns from Hardware-Software Codesign Data with the Learning Classifier System XCS. In *Proceedings of the 2003 Congress on Evolutionary Computation (CEC 2003),* pages 1486–1492, Canberra, Australia, 9-12 December 2003. IEEE.
4. R.K. Gupta and G. De Micheli. Hardware-software cosynthesis for digital systems. *Design & Test of Computers, IEEE,* 10:29–41, 1993.
5. J. Henkel and R. Ernst. An approach to automated hardware/software partitioning using a flexible granularity that is driven by high-level estimation techniques. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* 9:273–289, 2001.
6. G. De Micheli. *Synthesis and Optimization of Digital Circuits.* McGraw Hill, 1994.
7. M. Palesi and T. Givargis. Multi-objective design space exploration using genetic algorithms. In *Proceedings of the Tenth International Workshop on Hardware/Software Codesign,* pages 67–72, May 2002.
8. G. Quan, X. Hu, and G. Greenwood. Preference-driven hierarchical hardware/software partitioning. In *International Conference on Computer Design (ICCD '99),* pages 652–657, 1999.
9. Donatella Sciuto, Fabrizio Ferrandi, Pier Luca Lanzi, and Mara Tanelli. System-level metrics for hardware/software architectural mapping. In *Proceedings of the 2nd IEEE International Workshop on Electronics Design, Test and Applications (DELTA 2004),* Burswood Resort, Perth, Australia, January 2004.
10. F. Vahid and D.D. Gajski. Closeness metrics for system-level functional partitioning. In *Proceedings EURO-DAC'95 Design Automation Conference with EURO-VHDL,* pages 328–333, September 1995.
11. Stewart W. Wilson. Classifier Fitness Based on Accuracy. *Evolutionary Computation,* 3(2):149–175, 1995. http://prediction-dynamics.com/.
12. Stewart W. Wilson. Function approximation with a classifier system. In Lee Spector, Erik D. Goodman, Annie Wu, W.B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001),* pages 974–981, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann.

Parameter Adaptation within Co-adaptive Learning Classifier Systems

Chung-Yuan Huang[1,2] and Chuen-Tsai Sun[1]

[1] Department of Computer and Information Science
National Chiao Tung University
Hsinchu, Taiwan 300
{gis89802, ctsun}@cis.nctu.edu.tw
[2] Department of Computer Science and Information Engineering
Chung Kuo Institute of Technology
Hsinchu, Taiwan 303

Abstract. The authors propose a co-adaptive approach to controlling parameters for coevolution-based learning classifier systems. By taking advantage of the on-line incremental learning capability of such systems, solutions can be produced that completely cover a target problem. The system combines the advantages of both adaptive and self-adaptive parameter-control approaches. Using a coevolution model means that two learning classifier systems can operate in parallel to simultaneously solve target and parameter-setting problems. Furthermore, the approach needs very little time to become efficient in terms of latent learning, since it only requires small amounts of information on performance metrics during early run-time stages. Our experimental results show that the proposed system outperforms comparable models regardless of a problem's stationary/non-stationary status.

1 Introduction

Learning classifier systems (LCSs) [2] were introduced in the mid-1970s by John H. Holland, known as the father of genetic algorithms (GAs). Such systems take advantage of GAs and reinforcement learning (RL) [11] to build adaptive rule-based systems that learn gradually via on-line experiences [3, 6]. Depending on its architecture, a learning classifier system may be regarded as an extended GA application or a reinforcement learning algorithm. The reinforcement component of learning classifier systems is responsible for distributing credit among rules and resolving rule conflicts (i.e., distinguishing between appropriate and inappropriate rules). The genetic algorithm component is responsible for comparing performance in order to identify potentially better rules to replace unsuitable rules. Originally, learning classifier systems were not completely analyzable due to the complex nature of component interrelationships [3]. It wasn't until 1995, when Wilson proposed his eXtended Classifier System (XCS) [12] based on classifier prediction accuracy, that a number of new models and applications were presented, thus renewing interest in learning classifier systems. Wilson retained Holland's original ideas and main architecture, but made some fundamental changes that gave XCS at least four

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 774–784, 2004.

advantages [3, 12, 13]: a) easier analyzability; b) the ability to deal with complex problems (e.g., optimization issues) that had previously been considered unsolvable; c) adding a robust generalization mechanism capable of generating compact, complete, maximized, and accurate solutions [6]; and d) the capability to use various representations to specify classifiers [9, 10].

As with evolutionary computations (ECs), parameter settings determine whether learning classifier systems can generate optimal or near-optimal solutions and whether it can do so efficiently. All of the currently available approaches [1, 4, 5] to solving the parameter-setting problem associated with learning classifier systems have important drawbacks requiring improvement and modification [14-18]. Our proposed co-adaptive approach, which is based on the coevolution concept and Dyna architecture [11], takes advantage of the incremental on-line learning capability of learning classifier systems to produce solutions that completely cover a target problem.

The system simultaneously adapts parameters according to current learning performance and state. As shown in Fig. 1, the framework consists of two learning classifier systems. Main-LCS is responsible for solving the external target problem and Meta-XCS is responsible for adapting internal parameters. We used the Dyna architecture to acquire the parameter-control capability of Meta-XCS in a short time period. Dyna uses an internal world model to save real experiences that are obtained during learning and uses them for an intensive latent learning process that shortens training time and speeds up the construction of a complete set of solutions. In [8], Lanzi showed that a combination of Dyna and XCS (Dyna-XCS) was capable of greatly enhancing learning performance. For the present study we used Dyna-XCS to a) solve the slow-learning problem of the adaptive parameter control approach (which requires a long training period) and b) significantly enhance parameter-control stability.

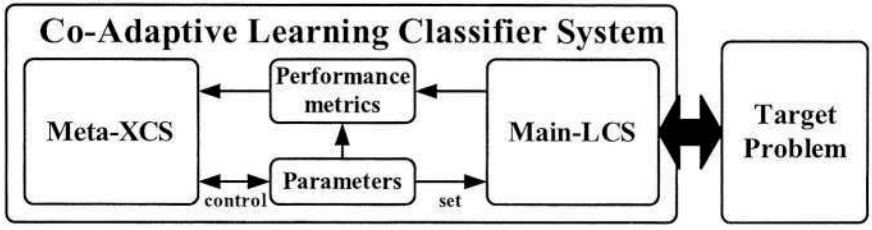


Fig. 1. Co-adaptive learning classifier system framework

To solve the parameter-control problem common to learning classifier systems, we established a framework in which Main-LCS and Meta-XCS operate in parallel. Solutions co-evolve as the systems cooperatively adjust parameters to a given target problem. There are two advantages to using coevolution to solve the parameter setting problem: many benefits of the self-adaptive parameter-control approach are maintained without expanding the target problem's original search space, and the premature convergence problem that often accompanies this approach is avoided.

2 XCS Overview

As with traditional learning classifier systems, XCS is a problem-independent and adaptive machine learning model. As shown in Fig. 2, XCS has four components: a finite population of classifiers, a performance component, a reinforcement component, and a rule discovery component. Stored classifiers control the system through a horizontal competition mechanism and perform tasks via vertical cooperation. The performance component governs interactions with the target problem. The input interface (called a detector) is used to transmit the current state of the target problem to the performance component and to determine dominant classifiers according to an exploration/exploitation criterion. Through the output interface (called an effector), any action advocated by dominant classifiers is performed and receives feedback. The reinforcement component (also known as the credit assignment component) uses an algorithm similar to Q-learning [11] to update the reinforcement parameters of classifiers advocated the action. Finally, the rule discovery component uses a genetic algorithm to search for better or more general classifiers and to discard existing incorrect or more specific classifiers.

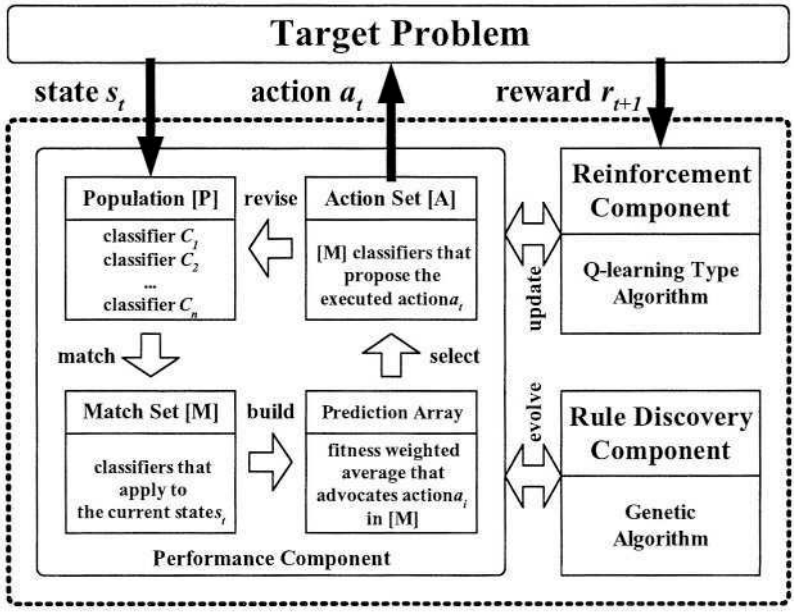


Fig. 2. XCS architecture

When running XCS, we use performance metrics to observe system performance and to state classifier populations. Kovacs [6] divided these performance metrics into two categories: performance measures and population state measures. Three well-known on-line metrics for measuring performance in research environments and real-world XCS applications are performance ρ, system error, and population size. Performance ρ and system error are used to measure XCS learning capability

according to results from target problem interactions. As one would expect, the population size performance metric (defined as the number of macro-classifiers in a classifier population) belongs to the category of population state measures. It is responsible for measuring XCS learning quality.

3 Co-adaptive Learning Classifier Systems

Because of its ability to deal with complex problems, we decided to use XCS to optimize other learning classifier system parameters, especially its ability to represent various parameter-control strategies [3, 9, 10]. Specifically, we used XCS to capture relations and changing parameter patterns between parameter-control strategies and to observe their effects on a given target problem. Its principal features include a) the combined advantages of adaptive and self-adaptive parameter-control approaches that allow for the use of a coevolution model to simultaneously solve a target problem and parameter-setting problem, and b) reduced time requirements for becoming efficient via a latent learning process that uses small pieces of information about performance metrics in the early stages of a run.

3.1 The Model

The architecture of the co-adaptive learning classifier system is shown in Fig. 3. Its four principal components are the Main-LCS, Meta-XCS, performance-metrics, and parameter components. As with ordinary learning classifier systems, the Main-LCS component is responsible for interacting with and solving the target problem. Meta-XCS integrates Dyna architecture with XCSμ [7, 8], which is especially useful in stochastic environments where the results of actions are affected by uncertainty. The Meta-XCS component learns parameter control and adjustment strategies in a short time period. The performance-metrics component is responsible for collecting, recording, and evaluating Main-LCS performance and regularly transmitting performance metrics information to the Meta-XCS component. The last component stores all parameters that need to be adjusted by the Meta-XCS component and assigns updated parameters to the Main-LCS component.

3.2 Meta-XCS Component

After a certain number of Main-LCS runs, the Meta-XCS component receives a message (s_t) transmitted by the parameter and performance-metrics components. In addition to the current parameter settings affecting the Main-LCS component, s_t also contains measures of the Main-LCS component's performance and population states. Based on the information in s_t, XCSμ in the Meta-XCS component executes parameter-control action a_t and instructs the parameter component to update the corresponding parameter. Next, the Meta-XCS component receives a s_{t+1} message and r_{t+1} feedback in the form of a reward or penalty from the performance-metrics component. The Meta-XCS component uses r_{t+1} for reinforcing learning and for storing the $<s_t, a_t, s_{t+1}, r_{t+1}>$ information within the Dyna architecture. During intervals

between parameter-control actions, the Meta-XCS component uses these records for latent learning. The cycle continues until the target problem is solved by the Main-LCS component or a user-requested criterion is met.

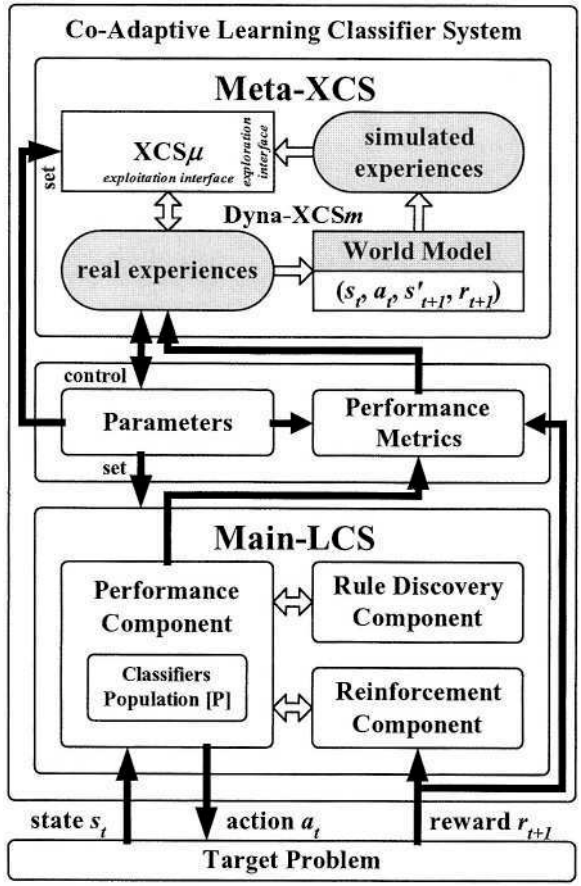


Fig. 3. Detail of co-adaptive learning classifier system architecture

3.3 Meta-XCS Dyna

Dyna uses the XCSμ exploration interface to perform latent learning, and the parameter-control operation is executed through the XCSμ exploitation interface. Theoretically, the latent learning and practical parameter control operations of Meta-XCS can be processed simultaneously, but in practice, a higher priority is assigned to the parameter-control operation in the Meta-XCS component in order to decrease potential conflicts and to meet the hardware restrictions of sequential processing. Therefore, latent learning is delayed until parameter-control operations are fully executed. However, we believe that the arrangement takes advantage of system idling time to improve parameter control and learning performance.

4 Experiments

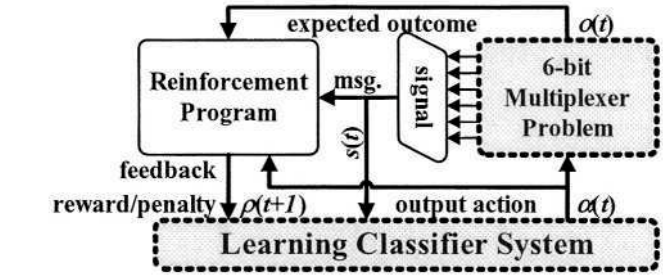
In learning classifier systems, the mutation operator plays an important role in learning performance and target problem solution quality. During the early stages of a run, the mutation operator provides novelty by moving classifiers within the search space. A faster mutation rate helps speed up the rule discovery component. As an essential background operator during the later stages, mutation ensures some probability of finding a better solution. A slower mutation rate helps fine-tune existing classifiers without disturbing runs or decreasing learning performance. However, it is difficult to predetermine the optimal mutation rate for a given target problem or to dynamically adjust the mutation rate during every stage of a run.

4.1 Test Environment: 6-bit Multiplexer Problems

Given the restrictions just described, we experimented with a co-adaptive parameter-control approach in the form of the 6-bit multiplexer problem (6-MP)—a version of the well-known benchmark single-step problem for machine learning in general and learning classifier systems in particular [12]. As shown in Fig. 4, the input message signal transmitted to learning classifier systems consists of a string of six binary digits in which the first (version A) or last (version B) two bits (called *address bits*) represent a binary index and the remaining bits represent data bits. The expected outcome is the value of the indexed data bit. For example, the expected outcome of "11*1110*" in version A is 0, since the first two bits (11) represent index 3—the fourth bit following the address. The expected outcome of "*0100*01" in version B is 1, since the second bit preceding the address is indexed.

6-MP is considered challenging because of its non-linear characteristic, yet it yields many useful generalizations that help in comparing learning performance in various models. During each cycle, the 6-MP produces signals by randomly setting all six bits. Expected outcomes are computed as single bits from the generated signals, which are transmitted as input messages to the learning classifier system on request and returned as output actions that are compared with expected outcomes. A positive feedback score of 1,000 means that a reward was returned to the learning classifier system for reinforcement; a feedback score of 0 means that a penalty was returned. During the run, the 6-MP continues to produce 6-bit messages with similar probabilities as the classifier system tries to learn the correct mapping relations between signals and expected outcomes—thus developing an optimal solution.

With the exception of the mutation rate, the default parameter for our experiments was $N = 800$, $\beta = 0.2$, $\alpha = 0.1$, $\varepsilon_0 = 10$, $\nu = 5$, $\theta_{GA} = 25$, $\chi = 0.8$, $\theta_{del} = 20$, $\delta = 0.1$, $\theta_{sub} = 20$, $P_{\#} = 0.33$, $p_1 = 10$, $\varepsilon_1 = 0$, $F_1 = 0.01$, $p_{expir} = 0.5$, *doGAsubsumption* = true, *doActionSetSubsumption* = true. All results discussed in this report represent an average of ten runs. Learning classifier system performance metrics were recorded for each trial and computed as average moving window numbers in the last 50 trials.



$$o_a(t) = f_{6\text{-}mp}(b_0,b_1,b_2,b_3,b_4,b_5) = \overline{b_0}\,\overline{b_1}\,b_2 + \overline{b_0}\,b_1\,b_3 + b_0\,\overline{b_1}\,b_4 + b_0\,b_1\,b_5 \quad \textit{(version A)}$$

$$o_b(t) = f_{6\text{-}mp}(b_0,b_1,b_2,b_3,b_4,b_5) = \overline{b_4}\,\overline{b_5}\,b_0 + \overline{b_4}\,b_5\,b_1 + b_4\,\overline{b_5}\,b_2 + b_4\,b_5\,b_3 \quad \textit{(version B)}$$

Fig. 4. The 6-bit Multiplexer Problem (6-MP)

4.2 Experiment 1: Stationary 6-bit Multiplexer Problem

We determined average variation in performance metrics at fixed mutation rates of 0.01, 0.05, and 0.09 (Fig. 5), and used our experimental results to observe parameter-setting problems that often occur in XCS. At a high mutation rate (0.09) the population size metric was often poor while performance ρ and system error metrics were good among the three models. At a low mutation rate (0.01) the population size metric was often good but the performance ρ and system error metrics very poor. When applying the co-adaptive learning classifier system to solve the 6-MP, issues tied to learning performance and population state were avoided. As shown In Fig. 5, the performance ρ and system error metrics for the co-adaptive learning classifier system were comparable to or better than those from the XCS at a fixed mutation rate of 0.09. Regarding the population size metric, the co-adaptive learning classifier system metric was similar to or outperformed the XCS metric at a fixed mutation rate of 0.01. After 10,000 trials, approximately 21 classifiers remained for forming an optimal solution.

As shown in Fig. 6, during the early stages of a run the co-adaptive learning classifier system increases its own mutation rate in order to produce classifiers that can solve the 6-MP, and in the later stages decreases the mutation rate in order to fine-tune existing classifiers and to arrive at an optimal solution. This variation in mutation rate continually oscillates between faster and slower. From our observations, it appears as though the co-adaptive learning classifier system tests the influence of a lower mutation rate during the early stages of a run and tests the influence of a higher mutation rate in the later stages. Mutation rates are abandoned if they negatively impact classifier populations or learning performance; in such cases, the system returns to the original rate.

4.3 Experiment 2: Non-stationary 6-bit Multiplexer Problem

Our second experiment was similar to the first except that the second made use of 6-MP versions A and B (Figs. 7 and 8). For this experiment we ran 20,000 trials—10,000 that were similar to the first experiment and 10,000 in which the input signal bit sequence was abruptly changed from version A to B. In the version B run, the two address lines were moved from the initial ($b_0 b_1$) to final input signal bit ($b_4 b_5$) position. Whenever the bit sequence underwent a sudden change during the second 10,000 trials, the co-adaptive learning classifier system had to re-generalize the existing classifiers and rebuild an appropriate solution. The goal was to determine whether or not the co-adaptive learning classifier system could quickly reestablish a proper mutation rate following an abrupt change in the problem environment, recover the original learning performance and population size state, and still rebuild an optimal solution.

Details of co-adaptive learning classifier system performance are presented in Fig. 7. Regardless of the performance metric, the system outperformed XCS at the fixed mutation rates of 0.01, 0.05, or 0.09. At a fixed mutation rate of 0.09, the performance ρ and system error metrics for the co-adaptive learning classifier system outperformed those from the XCS. At a fixed mutation rate of 0.01, the population size metric for the co-adaptive learning classifier system was similar to that from the XCS. An optimal solution aimed at the 6-MP version B was rebuilt after 20,000 trials.

Fig. 8 has two mutation rate peaks, the first before 2,500 trials and the second between 10,000 and 12,400 trials. Each peak reflects the time required by the co-adaptive learning classifier system to learn from the beginning. According to these experimental results, the co-adaptive learning classifier system is capable of handling non-stationary problems at a high performance level.

5 Conclusions

Previous studies of evolutionary computation and learning classifier systems describe the search for robust or optimal parameter sets for target problem solutions as a time-intensive trial-and-error task requiring large amounts of computation resources. Different parameter values are essential for inducing an optimal balance between exploration and exploitation at different run stages. In response to the common problem of setting parameters for practical applications, we extended the original learning classifier system with a parameter-control approach in order to enhance performance and learning stability.

Our proposal for a co-adaptive approach to learning classifier system parameters is based on a coevolution process and a Dyna architecture. The approach takes advantage of the on-line learning capabilities of learning classifier systems; solutions produced in this manner cover entire target problems. Results from our experiments show that the co-adaptive approach was successful in terms of setting parameters according to target problem properties. In both stationary and non-stationary problem experiments, the system outperformed the models it was tested against.

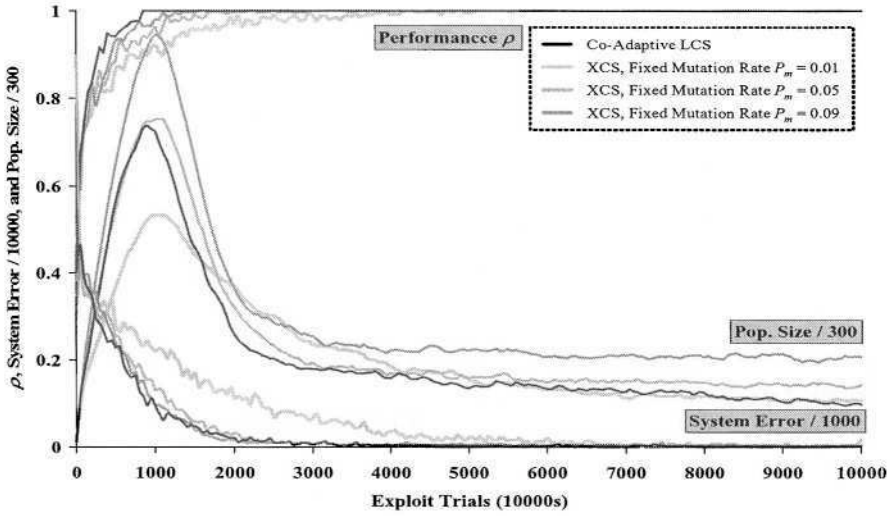


Fig. 5. Performance metrics (performance ρ, system error, and population size) of the co-adaptive learning classifier system and comparative models with fixed mutation rates of 0.01, 0.05, and 0.09 in the stationary 6-MP (version B)

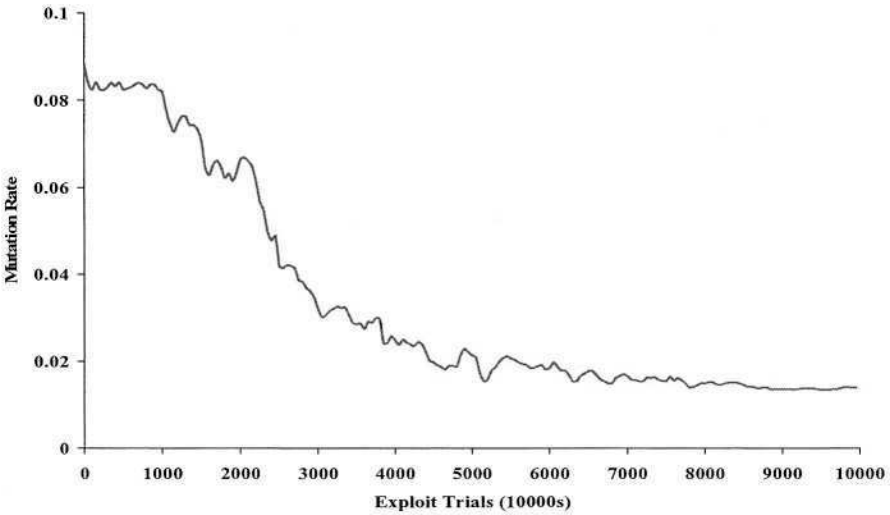


Fig. 6. Mutation rate adaptation for the co-adaptive learning classifier system in the stationary 6-MP (version A)

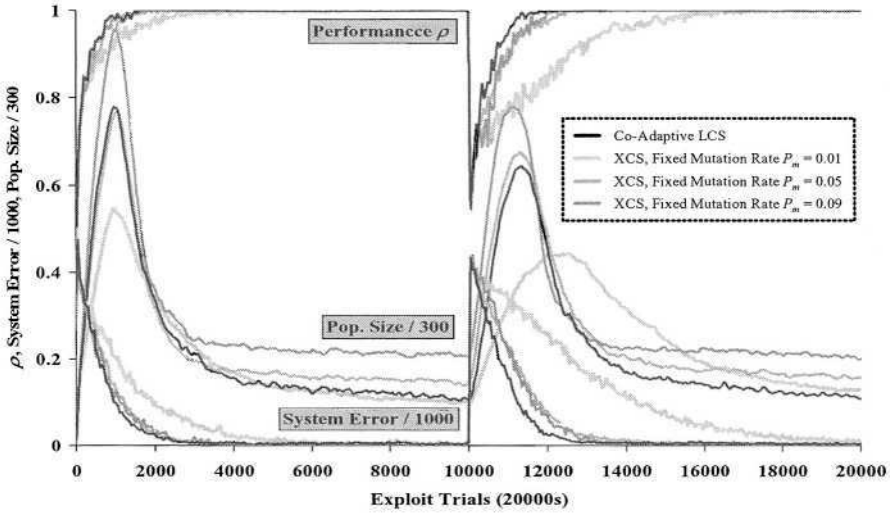


Fig. 7. Performance metrics (performance ρ, system error, and population size) of the co-adaptive learning classifier system and comparative models with fixed mutation rates of 0.01, 0.05, and 0.09 in the non-stationary 6-MP (versions A and B)

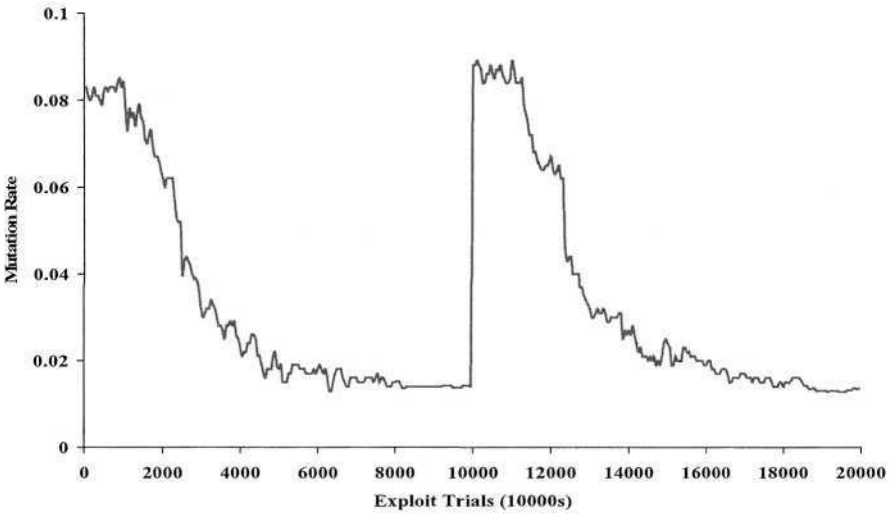


Fig. 8. Mutation rate adaptation for the co-adaptive learning classifier system in the non-stationary 6-MP (versions A and B)

References

1. Eiben, A.E., Hinterding, R., Michalewicz, Z.: Parameter Control in Evolutionary Algorithms. IEEE Transactions on Evolutionary Computation, Vol. 3, No. 2 (1999) 124-141
2. Holland, J.H.: Adaptation in Natural and Artificial Systems. MIT Press (1992); University of Michigan Press, Ann Arbor (1975)
3. Holmes, J.H., Lanzi, P.L., Stolzmann, W., Wilson, S.W.: Learning Classifier Systems: New Models, Successful Applications. Information Processing Letters, Vol. 82 (2002) 23-30
4. Hurst, J., Bull, L.: A Self-Adaptive Classifier System. In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) Advances in Learning Classifier Systems. Paris: Springer-Verlag (2001) 70-79
5. Hurst, J., Bull, L.: A Self-Adaptive XCS. In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) Advances in Learning Classifier Systems. Berlin: Springer-Verlag (2002) 50-73
6. Kovacs, T.: What Should a Classifier System Learn and How Should We Measure It? Journal of Soft Computing, Vol. 6, Nos. 3-4 (2002) 171-182
7. Lanzi, P.L., Colombetti, M.: An Extension to the XCS Classifier System for Stochastic Environments. In: Banzhaf, W., Daida, J., Eiben, A.E., Garzon, M.H., Honavar, V., Jakiela, M., Smith, R.E. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99). Morgan Kaufmann (1999) 353-360
8. Lanzi, P.L.: An Analysis of Generalization in the XCS Classifier System. Evolutionary Computation, Vol. 7, No. 2 (1999) 125-149
9. Lanzi, P.L.: Extending the Representation of Classifier Conditions Part I: From Binary to Messy Coding. In: Banzhaf, W., Daida, J., Eiben, A.E., Garzon, M.H., Honavar, V., Jakiela, M., Smith, R.E. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99). Morgan Kaufmann (1999) 337-344
10. Lanzi, P.L.: Extending the Representation of Classifier Conditions Part II: From Messy Coding to S-Expressions. In: Banzhaf, W., Daida, J., Eiben, A.E., Garzon, M.H., Honavar, V., Jakiela, M., Smith, R.E. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99). Morgan Kaufmann (1999) 345-352
11. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press (1998)
12. Wilson, S.W.: Classifier Fitness Based on Accuracy. Evolutionary Computation, Vol. 3, No. 2 (1995) 149-175
13. Wilson, S.W.: Generalization in the XCS Classifier System. In: Koza, J.R., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo, M., Fogel, D.B., Garzon, M.H., Goldberg, D.E., Iba, H., Riolo, R.L. (eds.): Genetic Programming 1998: Proceedings of the Third Annual Conference San Francisco. Morgan Kaufmann (1998) 665-674
14. Glickman, M., Sycara, K.: Reasons for Premature Convergence of Self-Adaptating Mutation Rates. Proceedings of the 2000 Congress on Evolutionary Computation CEC00 California, USA. IEEE Press (2000) 62-69
15. Liang, K.H., Yao, X., Newton, C.S.: Adapting Self-Adaptive parameters in Evolutionary Algorithms. Applied Intelligence, Vol. 15 (2001)
16. Rudolph, G.: Self-Adaptive Mutations May Lead to Premature Convergence. IEEE Transactions on Evolutionary Computation, Vol. 5, No. 4 (2001) 410-414
17. Swain, A.K., Morris, A.S.: Performance improvement of self-adaptive evolutionary methods with a dynamic lower bound. Information Processing Letters, Vol. 82, No. 1 (2002) 55-63
18. Herrera, F., Lozano, M.: Adaptive Genetic Operators Based on Coevolution with Fuzzy Behaviors. IEEE Transactions on Evolutionary Computations, Vol. 5, No. 2, (2001) 149-165

High Classification Accuracy Does Not Imply Effective Genetic Search

Tim Kovacs[1] and Manfred Kerber[2]

[1] University of Bristol, Bristol BS8 1UB, U.K.
kovacs@cs.bris.ac.uk
http://www.cs.bris.ac.uk/~kovacs
[2] University of Birmingham, Birmingham B15 2TT, U.K.
M.Kerber@cs.bham.ac.uk
http://www.cs.bham.ac.uk/~mmk

Abstract. Learning classifier systems, their parameterisation, and their rule discovery systems have often been evaluated by measuring classification accuracy on small Boolean functions. We demonstrate that by restricting the rule set to the initial random population high classification accuracy can still be achieved, and that relatively small functions require few rules. We argue this demonstrates that high classification accuracy on small functions is not evidence of effective rule discovery. However, we argue that small functions can nonetheless be used to evaluate rule discovery when a certain more powerful type of metric is used.

1 Introduction

Much research on Learning Classifier Systems (LCS) has made use of small artificial data sets to evaluate alternative systems, mechanisms, and parameter settings. Of these data sets, the venerable 6 multiplexer (a 6-bit Boolean function) is the most widely used test in the LCS literature [36,33,37,6,13,5,35,25, 11,14,15,10,39,16,18,19,40,9,20,3,8,21]. It has also been used with other machine learning systems including neural networks [4,2,17,38,5,34], decision trees [30,29], and the GPAC algorithm [28]. See [25] for a review of some of the earlier work using the multiplexer. A number of studies have looked at larger multiplexer functions, e.g. [39,40,22], up to the 70-bit multiplexer [7].

Although some authors have referred to the 6 multiplexer as a difficult task, section 3 demonstrates that the XCS classifier system with a fixed set of random rules, of the same number as standardly used by the XCS for this task, reliably achieves 100% classification accuracy. That is, we discontinue genetic search in XCS (except for covering) after the initial random rule population has been generated. We refer to this algorithm as XCS-NGA (XCS-No Genetic Algorithm). This result clearly demonstrates that good classification accuracy on this task is not evidence of the efficacy of genetic search. (We note that rule parameters such as fitness and prediction are updated as usual by the credit assignment system, and that we are thus dealing with fixed randomly defined regions of the input space rather than random classification of inputs.)

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 785–796, 2004.

The same holds for other small data sets. We demonstrate 98% classification accuracy with XCS-NGA on the 11-bit multiplexer, although this requires four times as many rules as normally used by XCS. We suggest any degree of accuracy can be achieved on any task, given sufficient random rules. We do not suggest this is a practical approach as the number of rules required scales poorly. However, we do suggest that this is a useful measure of the difficulty of a given task, and one which can potentially demonstrate the invalidity of conclusions drawn from experiments with smaller tasks, for example those with the 6 multiplexer.

Despite this criticism of the use of small tasks, we argue that many conclusions drawn from them are valid. For example, comparisons based on small tasks can demonstrate performance differences between alternative mechanisms and parameterisations. We demonstrate this in section 3.3 by comparing the performance of XCS with and without initial populations on the 6 multiplexer.

We further demonstrate in section 4 that an alternative measure of adaptation can distinguish between XCS and XCS-NGA on the 6 multiplexer, and argue that the use of this metric increases the utility of small tests.

2 Method

2.1 XCS

For our experiments we will use XCS [39], which is currently the most widely used classifier system, and which has shown good results on data mining tasks (e.g. [31,12]). XCS introduced a number of innovations, foremost among them its accuracy-based fitness under which rule fitness is related to its classification accuracy and not the magnitude of the reward it receives as in earlier systems. For lack of space we do not include the details of the XCS updates, but suffice it to say that XCS evaluates the prediction and fitness of each rule. Prediction is, for concept learning tasks such as those we study here, an estimate of the proportion of inputs matched by the rule which belong to the positive class. Prediction is used in conflict resolution, when matching rules perform a weighted vote on the classification of a data point. Accuracy is a measure of the consistency of prediction. Rules with prediction near the maximum or minimum have high fitness. Higher fitness rules are allocated more reproductive opportunities by the genetic algorithm in XCS, and fitness is also factored into the classification vote.

For our experiments we use if-then rules whose conditions are terms in Disjunctive Normal Form. Specifically, we use the ternary representation widely used with classifier systems, in which rule conditions are drawn from $\{0, 1, \#\}$ and rule actions (classifications) are drawn from {0,1}. Inputs to the system are also drawn from {0,1}. A rule's condition c matches an environmental input m if for each character m_i the character in the corresponding position c_i is identical or the wildcard (#). For example, the condition 00# matches two inputs: 000 and 001. The wildcard is the means by which rules generalise over environmental states; the more #s a rule contains the more general it is. *Overgeneral* rules are those which misclassify some of the inputs they match. Since actions do not contain wildcards the system cannot generalise over them.

If no rule matches the current input, XCS's *covering* mechanism is triggered. This mechanism takes the current input and with probability $P_{\#}$ for each bit flips it to a #, and uses this as the condition for a new rule with a random classification. XCS may or may not use an initial population of random rules whose conditions are generated with $P_{\#}$ and equiprobable 0s and 1s. The covering mechanism is used regardless of whether an initial population is used, but, when $P_{\#}$ is not very close to 0, covering is triggered only sparingly and typically only at the outset of the experiment, even in the absence of an initial population.

2.2 XCS-NGA

Our procedure for learning with random rules, XCS-NGA, uses XCS modified so that genetic search does not operate on the initial rule population. In all other respects, XCS-NGA functions as XCS. The adaptive power of this approach lies in the XCS updates which estimate the prediction and fitness of rules, and weight classification votes on these two values. In section 3.1 we give some intuition as to why this approach can be effective.

We could attempt to improve XCS-NGA by restricting the generality of rules found to be overgeneral, or simply deleting them. However, our aim is not to propose a practical learning technique but rather to provide a baseline against which to evaluate other methods.

We note that in experimental results presented later we will quote a certain number of random rules having been used. In some experiments, some additional rules will have been generated by covering. In these cases, the same number of initial random rules will have been removed from the population in order to make room for the rules generated by covering. In most experiments covering does not occur, and when it does (typically when the rule set is small) it is not triggered many times.

2.3 The Multiplexer Tests

The 6 multiplexer is one of a family of Boolean multiplexer functions defined for strings of length $L = k + 2^k$ where k is an integer > 0. The series begins $L = 3, 6, 11, 20, 37, 70, 135, 264, 521 \ldots$. The first k bits are used to encode an address into the remaining 2^k bits, and the value of the function is the value of the addressed bit. In the 6 multiplexer ($k = 2, L = 6$), the input to the system consists of a string of six binary digits, of which the first $k = 2$ bits (the address) represent an index into the remaining $2^k = 4$ bits (the data). For example, the value of 101101 is 0 as the first two bits 10 represent the index 2 (in base ten) which is the third bit following the address. Similarly, the value of 001000 is 1 as the 0th bit after the address is indexed.

To use the 6 multiplexer as a test, on each time step we generate a random binary string of 6 digits which we present as input to the LCS. The LCS responds with either a 0 or 1, and receives a high reward (1000) if its output is that of the multiplexer function on the same string, and a low reward (0) otherwise.

2.4 Statistics

Classification Accuracy. We make use of Wilson's explore/exploit framework [39], in which training and testing interleave, so the learner is evaluated as it is learning, rather than after it has been trained. Specifically, on each time step we alternate between explore and exploit modes. In the former we select an action at random from among those advocated by the set of matching rules. In the latter we select the action most strongly advocated by the matching rules. We record statistics only on those time steps in which we exploit (exploit trials).

Wilson defines a measure of performance which he refers to simply as "performance" [39]. Performance is defined as a moving average of the proportion of the last n trials in which the system has responded with the correct action, where n is customarily 50. That is, on each time step, we determine the proportion of the last n time steps on which the LCS has taken the correct action. The performance curve is scaled so that when the system has acted correctly on all of the last 50 time steps it reaches the top of the figure, and when it has acted incorrectly on all these time steps it reaches the bottom of the figure.

Macroclassifiers. In addition to performance, on each exploit trial we monitor the number of *macro*classifiers in the population. These are rules with a numerosity parameter indicating the number of identical virtual rules represented by the macroclassifier. The macroclassifier curves gives us an indication of the diversity in the rule population and the extent to which it has found and converged on useful general rules and hence a compact representation of the solution. When an initial population is used the macroclassifier curve starts a little below the specified population size limit since a few duplicate rules are likely to have been generated. This curve can at most reach the population size limit, which would occur when each rule in the population has a unique condition/action pair. In the figures shown later, the number of macroclassifiers is divided by 1000 in order to display it simultaneously with other curves.

2.5 Parameter Settings

For the 6 multiplexer the standard XCS parameter settings from [39] were used: subsumption threshold $\theta_{sub} = 20$, Genetic Algorithm (GA) threshold $\theta_{GA} = 25$, covering threshold $\theta_{mna} = 1$, low-fitness deletion threshold $\delta = 0.1$, population size limit $N = 400$, learning rate $\beta = 0.2$, accuracy falloff rate $\alpha = 0.1$, accuracy criterion $\varepsilon_o = 0.01$, crossover rate $\chi = 0.8$, mutation rate $\mu = 0.04$. Hash probability $P_{\#} = 0.3$ rather than the standard 0.33 as a study of different values (not shown) was performed with hash probabilities at regular intervals, and the results shown are a subset of this study. GA subsumption was used but not action set subsumption. The original accuracy calculation was used [39]. Rules were deleted as in [20] with a delay of $\theta_{del} = 25$, and mutated bits had equiprobable outcomes. Initial random populations were used except as noted.

Parameter settings for the 11 multiplexer differed only in having a population size of 800 and hash probability of 0.4, to compensate for the larger search space.

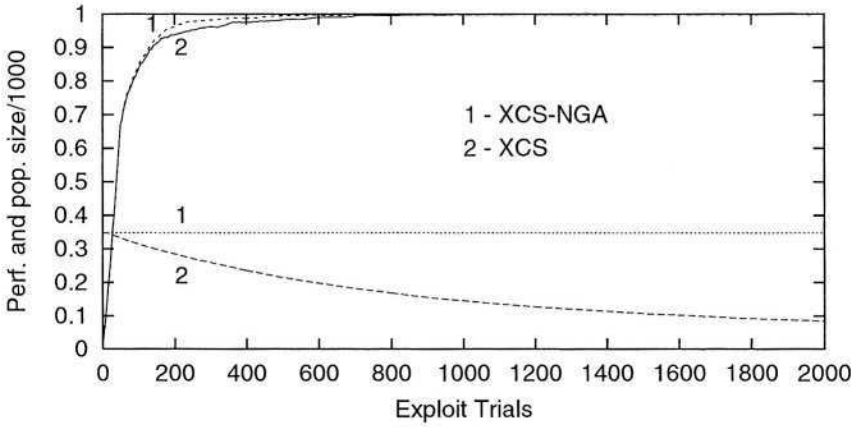


Fig. 1. XCS and XCS-NGA compared on the 6 multiplexer. The upper two curves show performance and the lower two show the population size in macroclassifiers.

3 The Difficulty of Small Boolean Functions

In this section we evaluate XCS and XCS-NGA empirically on the 6 and 11-bit multiplexer functions and discuss implications of our findings. Fig. 1 compares XCS and XCS-NGA on the 6 multiplexer. Curves are averages of 100 runs. The upper two curves show performance and the lower two show the population size in macroclassifiers (divided by 1000). Although 400 initial rules were generated for each system, both population size curves start somewhat below this as the curves show macroclassifiers, and some duplicate rules were generated. Both systems reach 100% performance, and, perhaps surprisingly, XCS-NGA does so first. We note that random guessing of class has an expected performance of 50% on multiplexer tasks, and, given the even class distribution, guessing the majority class of previously seen inputs will perform somewhat worse than random [32].

Fig. 2 repeats the comparison using the 11 multiplexer. Curves are an average of 50 runs. XCS-NGA was evaluated with 800 and 3200 rules. In both cases XCS-NGA initially outperformed XCS. XCS-NGA with 800 random rules ultimately achieve approximately 86% classification accuracy and with 3200 rules reached approximately 98%, while XCS eventually reaches 100%. (A larger number of random rules should do even better, but in a sense 98% is high enough: using a method with disjoint training and testing data sets, overfitting will occur on many data sets by the time 98% performance is reached.)

3.1 How XCS-NGA Achieves High Accuracy

Suppose we have a space of data points to be categorised. XCS uses a generate-and-test approach to classification, which entails two problems: i) rule discovery

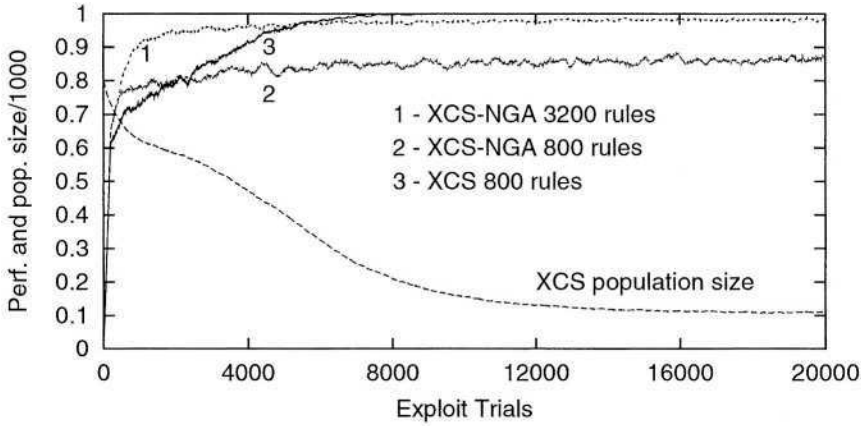


Fig. 2. XCS and XCS-NGA compared on the 11 multiplexer.

and ii) credit assignment. Specifically, XCS addresses problem i) using a GA to generate fitter rules (regions in the data space), each with an associated class label. Problem ii) is that of evaluating rule fitness such that more general rules and rules with higher classification accuracy are fitter. Essentially, rules must be found which capture many positive data points and few negative ones (or vice versa). XCS classifies data points by a vote among the rules which match it, with each vote weighted both by the rule's fitness. In this way, a point matched by a low-accuracy rule and a high-accuracy rule is given the classification of the high-accuracy rule.

In XCS, the rules (region shapes and sizes) are adapted by the genetic algorithm. XCS-NGA lacks a GA and its region shapes and sizes do not change; only the classification made through voting may change. XCS-NGA relies on there being enough rules to adequately solve problem i) (rule discovery) by chance. Of the randomly generated rules, those with low classification accuracy are assigned low weights and have less influence in the classification vote than higher accuracy rules. Roughly speaking, XCS-NGA's approach is to generate many random rules and ignore those which happen to have low accuracy. The number of random rules needed for high classification accuracy on small multiplexers is low because there are relatively few data points and clustering them into regions of the same class is easy (using our chosen language).

The difficulty of the rule discovery problem depends on the Kolmogorov complexity[1] of the data set. There is considerable variability in the Kolmogorov complexity of functions of the same length and representation. For example, with the language we have used the 6-bit parity functions are much more complex than the 6 multiplexer, which in turn is considerably more complex than 6-bit constant functions. Elsewhere, we have demonstrated a strong correlation between the size

[1] In simple terms, the shortest possible representation in a given formalism [24].

of the minimal representation of these functions and their difficulty for XCS [23]. One consequence is that even successful solution by XCS of a large multiplexer, such as the 70-bit multiplexer [7], does not mean that XCS can solve all 70-bit functions with comparable effort; quite the opposite. We hypothesise that the difficulty of a function for XCS-NGA will also correlate with the minimal number of rules needed to represent the function in a particular language.

XCS-NGA is related to a number of other machine learning algorithms. For example, CMAC function approximators [1] adapt the weight of each region in each of multiple partitions of the input space. Partitions may be regular or generated at random, and XCS-NGA differs essentially only in the details of how regions are formed. XCS-NGA is also very similar to the Weighted-Majority algorithm [27], which enumerates all possible concepts and weights them according to their consistency with the training data. They differ in that XCS-NGA generates only a random subset of possible concepts.

3.2 Why XCS-NGA Initially Outperforms XCS

It is not clear why XCS-NGA initially outperform XCS, but it may be that XCS is deleting overgeneral rules which have some value; overgenerals can advocate the correct action for the great majority of inputs they match. In both XCS and XCS-NGA, overgeneral rules have low accuracy and hence low weight in action selection, but nonetheless may have some effect. In XCS, however, low accuracy results in low fitness and greater likelihood of deletion under the genetic algorithm, and once deleted rules have no effect. Further study of this phenomenon is warranted, and perhaps improved performance in XCS can be obtained by allowing it to retain overgeneral rules when no accurate rule matches an input, or by delaying the application of the GA to the initial population until it has been better evaluated.

3.3 Implications of High Accuracy of XCS-NGA

Although XCS outperforms XCS-NGA on the 11 multiplexer, it seems likely that XCS-NGA with a large enough set of random rules would also reach 100% performance on this function, or indeed any function.

Although we have shown that good performance on the 6 multiplexer with 400 rules does not demonstrate effective genetic search in a classifier system, we do not claim that the 6 multiplexer is without uses. For example, in section 2.1 we noted that XCS may either use an initial population of rules or rely entirely on covering to generate rules. Fig. 3 compares XCS with and without an initial population of 400 rules on the 6 multiplexer, and clearly shows the performance advantage which occurs with an initial population. Consequently, we argue that only those studies which claim effective genetic search based on results with small functions are demonstrated invalid by our results with XCS-NGA.

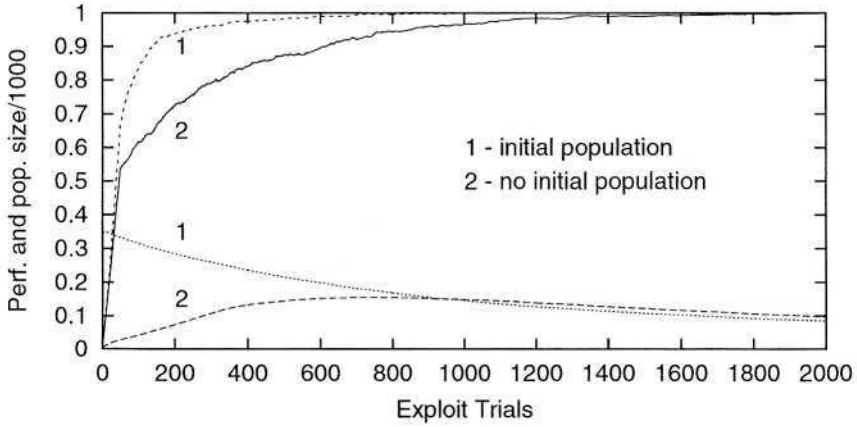


Fig. 3. XCS with and without an initial population on the 6 multiplexer.

4 A More Powerful Metric for Evaluating Genetic Search

High classification accuracy is usually regarded as the primary goal of a concept learning system. (Another goal might be human readability.) However, from the point of view of a researcher engaged in engineering better concept learning systems, classification accuracy is not a goal in itself, but just an indication of the relative merit of alternative mechanisms and parameterisations. In this context, classifier systems researchers often interpret good classification accuracy as an indication of effective genetic search for good classification rules. However, our demonstration in section 3 of the high classification accuracy achievable with XCS-NGA on the 6 and 11 multiplexers indicates that this interpretation is not justified when the number of rules is high relative to the size of the problem. This suggests that in order to evaluate the efficacy of genetic search good classification performance is required with a small number of rules on a large problem. Unfortunately, this approach requires evaluation of "large" and "small" in the context of a particular learning system, and running experiments on large problems is computationally expensive. In this section we demonstrate use of a metric which provides an alternative to large problems. Experiments with this metric clearly distinguish the efficacy of genetic search as opposed to random rules on the 6 multiplexer.

Using the rule language of section 2.1 the most compact description of the 6 multiplexer is a set of 8 rules, each as general as it can be without being overgeneral. Because XCS assigns fitness based on rule accuracy it actually finds each rule and its complement – the rule with the same condition but complementary action. This forms a set of 16 rules referred to as the *optimal solution* due to the optimal generality of each rule, and the minimality of the set.

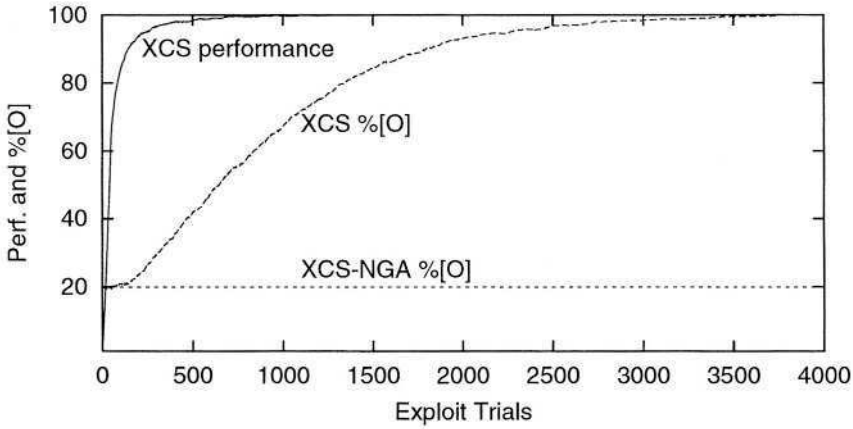


Fig. 4. Proportion of the optimal solution on the 6 multiplexer.

The proportion of the optimal solution in the rule population on a given time step, denoted %[O], has been used as a measure of the progress of genetic search. In [20] it was shown to have greater discriminatory power than the performance metric of section 2.4, and we will show that it can discriminate between XCS and XCS-NGA on the 6 multiplexer. Fig. 4 shows the performance of XCS and the %[O] of both XCS and XCS-NGA on this task. Curves are averages of 100 runs. While XCS-NGA contains a fixed proportion of approximately 20% of the optimal population, the proportion of this set grows in XCS until it reaches 100% (indicating all 16 rules occur in the population). Similarly, Fig. 5 compares %[O] for XCS and XCS-NGA on the 11 multiplexer, averaged over 50 runs. The size of the optimal solution for this function (including complementary rules) is 32. In this case XCS-NGA contains a much smaller proportion of the optimal set than in the 6 multiplexer because on the 11 multiplexer the optimal rules form a smaller proportion of the set of all rules.

Clearly this metric is better able to discern the progress of genetic search than the performance metric. We argue that using this metric extends the utility of small tests. However, we note that, as discussed in [22], %[O] has disadvantages including the need to compute the optimal solution in advance, the computational expense of evaluating it, and the complication that some functions have multiple optimal solutions (alternative minimal solution sets). These features make it difficult or impossible to apply %[O] to some tasks. In such cases measuring the population size in macroclassifiers or plotting mean rule generality can somewhat compensate for the lack of %[O], as decline of the former and rise of the latter both imply effective genetic search. Finally, replacing the GA with an iterative random rule generator would provide a baseline against which to compare genetic search.

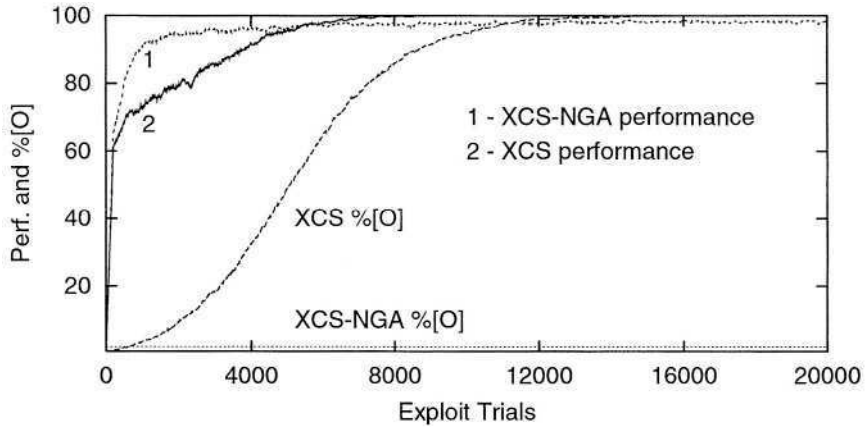


Fig. 5. Proportion of the optimal solution on the 11 multiplexer.

5 Conclusion

With XCS-NGA we have demonstrated that adapting the prediction and fitness of a fixed set of random rules reliably achieves 100% classification accuracy on the 6 multiplexer even when we only use as many rules as are standardly used to solve this task with XCS. This illustrates the danger of interpreting good classification accuracy on small tasks as indicative of successful genetic search. We have demonstrated the very high classification accuracy of XCS-NGA on the 11 multiplexer, and suggest that arbitrarily high accuracy can be achieved on any concept learning task given enough random rules. Given these results, we suggest that XCS-NGA can provide a useful baseline for classification accuracy.

We have demonstrated that, in contrast to performance, an alternative metric based on the proportion of the optimal solution present in the rule population clearly distinguishes the performance of fixed random rules and genetic search.

As a final point, we note that the use of small tasks is limited neither to classifier systems nor to artificial data sets. Although most data sets in the very widely used UCI repository of machine learning data sets [26] have larger attribute spaces than the 11-multiplexer studied here, many have fewer data points. Consequently a reasonable number of random rules seems likely to perform well on those data sets just as they do on those tested here.

References

1. J. S. Albus. A new approach to manipulator control: the cerebellar model articulation controller (CMAC). *Journal of dynamic systems, measurement and control,* 97(3), 1975.
2. C. W. Anderson. *Learning and Problem solving with multilayer connectionist systems.* PhD thesis, University of Massachussetts, Amherst, MA, USA, 1986.

3. Alwyn Barry. *XCS Performance and Population Structure within Multiple-Step Environments.* PhD thesis, Queens University Belfast, 2000.
4. A. G. Barto, P. Anandan, and C. W. Anderson. Cooperativity in networks of pattern recognizing stochastic learning automata. In *Proceedings of the Fourth Yale Workshop on Applications of Adaptive Systems Theory,* pages 85–90, 1985.
5. Pierre Bonelli, Alexandre Parodi, Sandip Sen, and Stewart Wilson. NEWBOOLE: A Fast GBML System. In *International Conference on Machine Learning,* pages 153–159, San Mateo, California, 1990. Morgan Kaufmann.
6. Lashon B. Booker. Triggered rule discovery in classifier systems. In J. David Schaffer, editor, *Proc. 3rd International Conference on Genetic Algorithms (ICGA-89),* pages 265–274, George Mason University, June 1989. Morgan Kaufmann.
7. Martin Butz, Tim Kovacs, Pier Luca Lanzi, and Stewart W. Wilson. Toward a theory of generalization and learning in xcs. *To appear in the IEEE Transactions on Evolutionary Computation,* 2004.
8. Martin V. Butz, David E. Goldberg, and Wolfgang Stolzmann. Investigating Generalization in the Anticipatory Classifier System. In *Proceedings of Parallel Problem Solving from Nature (PPSN VI),* 2000. Also technical report 2000014 of the Illinois Genetic Algorithms Laboratory.
9. Henry Brown Cribbs III and Robert E. Smith. What Can I do with a Learning Classifier System? In C. Karr and L. M. Freeman, editors, *Industrial Applications of Genetic Algorithms,* pages 299–320. CRC Press, 1998.
10. Bart de Boer. Classifier Systems: a useful approach to machine learning? Master's thesis, Leiden University, 1994.
11. Kenneth A. De Jong and Willliam M. Spears. Learning Concept Classification Rules Using Genetic Algorithms. In *Proceedings of the International Joint Conference on Artificial Intelligence,* pages 651–656, Sidney, Australia, 1991.
12. Phillip William Dixon, David W. Corne, and Martin John Oates. A preliminary investigation of modified xcs as a generic data mining tool. In Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors, *Advances in Learning Classifier Systems,* volume 2321 of *LNAI,* pages 133–150. Springer-Verlag, Berlin, 2002.
13. David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, MA, 1989.
14. David Greene and Stephen Smith. COGIN: Symbolic induction using genetic algorithms. In *Proceedings 10th National Conference on Artificial Intelligence,* pages 111–116. Morgan Kaufmann, 1992.
15. David Greene and Stephen Smith. Using Coverage as a Model Building Constraint in Learning Classifier Systems. *Evolutionary Computation,* 2(1):67–91, 1994.
16. John H. Holmes. *Evolution-Assisted Discovery of Sentinel Features in Epidemiologic Surveillance.* PhD thesis, Drexel University, 1996.
17. R. A. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural Networks,* 1:295–307, 1988.
18. Tim Kovacs. Evolving Optimal Populations with XCS Classifier Systems. Master's thesis, University of Birmingham, 1996.
19. Tim Kovacs. XCS Classifier System Reliably Evolves Accurate, Complete, and Minimal Representations for Boolean Functions. In Roy, Chawdhry, and Pant, editors, *Soft Computing in Engineering Design and Manufacturing,* pages 59–68. Springer–Verlag, London, 1997.
20. Tim Kovacs. Deletion schemes for classifier systems. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference,* pages 329–336. Morgan Kaufmann, 1999.

21. Tim Kovacs. What should a classifier system learn? In *Proc. of the 2001 Congress on Evolutionary Computation (CEC01),* pages 775–782. IEEE Press, 2001.
22. Tim Kovacs. *Strength or Accuracy: Credit Assignment in Learning Classifier Systems.* Springer, 2004.
23. Tim Kovacs and Manfred Kerber. What makes a problem hard for XCS? In Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors, *Advances in Learning Classifier Systems,* LNAI 1996, pages 80–99. Springer–Verlag, 2001.
24. Ming Li and Paul Vitányi. An Introduction to Kolmogorov Complexity and Its Applications. Springer, 1997.
25. Gunar E. Liepins and Lori A. Wang. Classifier System Learning of Boolean Concepts. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms (ICGA-91),* pages 318–323, San Mateo, CA, 1991. Morgan Kaufmann Publishers.
26. Christopher Merz and P. M. Murphy. UCI Repository of Machine Learning Databases. http://www.ics.uci.edu/˜mlearn/MLRepository.html Irvine, CA: University of California, Department of Information and Computer Science, 1997.
27. Tom M. Mitchell. *Machine Learning.* McGraw–Hill, 1997.
28. E. M. Oblow. Implementing Valiant's Learnability Theory using Random Sets. Technical Report ORNL/TM-11512R, Oak Ridge National Laboratory, 1990.
29. G. Pagallo and D. Haussler. Boolean feature discovery in empirical learning. *Machine Learning,* 5(1):71–100, 1990.
30. J. R. Quinlan. An empirical comparison of genetic and decision-tree classifiers. In *Proc. of the Fifth Int. Machine Learning Conference,* pages 135–141, 1988.
31. Shaun Saxon and Alwyn Barry. XCS and the Monk's Problems. In Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors, *Learning Classifier Systems. From Foundations to Applications,* volume 1813 of *LNAI,* pages 223–242, Berlin, 2000. Springer-Verlag.
32. Cullen Schaffer. A conservation law for generalization performance. In Haym Hirsh and Willian W. Cohen, editors, *Machine Learning: Proc. of the Eleventh International Conference,* pages 259–265, San Francisco, CA, 1994. Morgan Kaufmann.
33. Sandip Sen. Classifier system learning of multiplexer function. The University of Alabama, Tuscaloosa, Alabama. Class Project, 1988.
34. Robert E. Smith and H. Brown Cribbs. Is a Learning Classifier System a Type of Neural Network? *Evolutionary Computation,* 2(1):19–36, 1994.
35. L. A. Wang. Classifier System Learning of the Boolean Multiplexer Function. Master's thesis, Computer Science Dept., University of Tennessee, Knoxville, 1990.
36. Stewart W. Wilson. Classifier Systems and the Animat Problem. *Machine Learning,* 2:199–228, 1987.
37. Stewart W. Wilson. Bid competition and specificity reconsidered. *Complex Systems,* 2:705–723, 1989.
38. Stewart W. Wilson. Perceptron Redux: Emergence of Structure. *Physica D,* pages 249–256, 1990.
39. Stewart W. Wilson. Classifier Fitness Based on Accuracy. *Evolutionary Computation,* 3(2):149–175, 1995.
40. Stewart W. Wilson. Generalization in the XCS classifier system. In John Koza et al., editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference,* pages 665–674. Morgan Kaufmann, 1998.

Mixed Decision Trees: Minimizing Knowledge Representation Bias in LCS

Xavier Llorà[1] and Stewart W. Wilson[2]

[1] Illinois Genetic Algorithms Laboratory (IlliGAL),
National Center for Supercomputing Applications,
University of Illinois at Urbana-Champaign, Urbana, IL 61801.
xllora@illigal.ge.uiuc.edu
[2] Prediction Dynamics, Concord, MA,
Department of General Engineering,
University of Illinois at Urbana-Champaign, Urbana, IL 61801.
wilson@prediction-dynamics.com

Abstract. Learning classifier systems tend to inherit—*a priori*—a given knowledge representation language for expressing the concepts to learn. Hence, even before getting started, this choice biases what can be learned, becoming critical for some real-world applications like data mining. However, such bias may be minimized by hybridizing different knowledge representations via evolutionary mixing. This paper presents a first attempt to produce an evolutionary framework that evolves mixed decision trees of heterogeneous knowledge representations.

1 Introduction

The genetic algorithms (GAs) community, as well as researchers coming from the genetic programming (GP) field and learning classifier systems (LCS) practitioners, have been using evolutionary algorithms for building (or inducing depending on the authors' emphasis) decision trees. Initially, researchers used GAs and GP for building decision trees by means of optimizing the classification error of a forest of decision trees [1,2,3]. Later on, researchers took this effort even further building frameworks that challenge the traditional methods for decision tree induction [4,5,6,7,8]. Other researchers introduced GAs for improving the efficiency of traditional tree builders [9].

Nevertheless, the majority, with a few exceptions that emphasize the relevance of the knowledge representation language [6], ignored the relevance of the knowledge representation in biasing the concepts that can be learned. Most of this research focused on evolving decision trees based on axis-parallel—let's call them orthogonal—classification boundaries. A few of them worked with oblique decision trees trying to enrich the set of concepts that can be learned, using GAs to beat the computational complexity of hyperplane split computation [9]—an NP-Hard problem. However, all these approaches evolve homogeneous decision trees, in which all the individuals of the population are encoded using the same knowledge representation language.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 797–809, 2004.

Evolutionary computation is a powerful tool, not only for addressing NP problems like the one mentioned above, but also for adapting the knowledge representation to the specific problem to be solved. Given this potential, we decided to investigate an evolutionary *mixing* of different kinds of trees–or different knowledge representations. This approach, besides using evolutionary algorithms to tackle NP problems, should also minimize the bias introduced by the knowledge representation, producing what we call *mixed decision trees.*

Section 2 briefly introduces an evolutionary LCS framework used for evolving decision trees. This framework already evolves different kinds of homogeneous decision trees, as shown in section 3. In this paper we took this framework one step further allowing it to produce mixed trees. Section 4 reviews the experimentation conducted using mixed trees, as well as summarizes the results obtained. Finally, section 5 presents some conclusions.

2 GALE

This section briefly reviews an algorithm of the so-called *Pittsburgh* approach to LCS. GALE *(Genetic and Artificial Life Environment)* is a fine grain parallel model for knowledge discovery that integrates elements of the *Pittsburgh* approach and cellular automata. The rest of this section presents an overview of the key elements of GALE—a detailed description can be found in [10].

2.1 Topology and Knowledge Representation

GALE uses a 2D grid (board $\mathcal{T}$) formed by $m \times n$ cells for spreading the evolving population spatially. Each cell ($\mathcal{T}_{ij}$) of the grid contains either one ($\zeta(\mathcal{T}_{ij}) = 1$) or zero individuals ($\zeta(\mathcal{T}_{ij}) = 0$); thus, for instance, a 32×32 grid can contain up to 1024 individuals, each one placed on a different cell. Each individual ($\mathcal{T}_{ij}^{I}$) is a complete solution to the classification problem; in fact, each individual codifies the knowledge that describes the mined data set. Genetic operators are restricted to the immediate neighborhood ($\mathcal{T}_{ij}^{\nu}$) of the cell in the grid. The size of the neighborhood is r. Given a cell $\mathcal{T}_{ij}$ and $r = 1$, the neighborhood $\mathcal{T}_{ij}^{\nu}$ of $\mathcal{T}_{ij}$ is defined by the 8 adjacent cells to $\mathcal{T}_{ij}$ (being $\zeta\left(\mathcal{T}_{ij}^{\nu}\right)$ the number of occupied cells in $\mathcal{T}_{ij}^{\nu}$). Thus, r is the number of hops that defines the neighborhood, and p_{ζ} is the probability that a cell $\mathcal{T}_{ij}$ contains an individual after initialization.

The evolutionary model of GALE coevolves different knowledge representations. GALE can perform homogeneous runs where only one type of knowledge representation is used [11,10], or heterogeneous runs where different knowledge representations compete in the same board [12]. Coevolving individuals expressed using different knowledge representations in the same board $\mathcal{T}$, helps GALE minimizing the bias introduced by the knowledge representation [13]. However, this approach is based on restricted mating policies, not allowing the mixing of different knowledge representation.

GALE mainly deals with three different knowledge representations [10]: (1) sets of fully-defined instances—or prototypes—[14,15], (2) orthogonal (or axis-parallel) decision trees [16], (3) oblique decision trees [17,18], and (4) rules sets.

GALE can perform homogeneous [14,11] and heterogeneous runs [12]. However, even in the heterogeneous runs, the individual maintain their initial knowledge representation. That means that no mixing of different knowledge representations takes place in the original GALE approach.

2.2 Mapping

GALE works as a supervised learning model. Hence, one key point is how the training instances Σ are spread over the board $\mathcal{T}$. In other words, each individual $\mathcal{T}_{ij}^{I}$ is evaluated using a set of examples mapped to the cell $\mu(\mathcal{T}_{ij}, \Sigma)$. The fitness function used is $fit(I) = \left(\frac{I^c}{l}\right)^2$ [19], being I^c the number of correctly classified instances and l the number of instances of the $\mu(\mathcal{T}_{ij}, \Sigma)$ data set.

The easiest mapping approach is to allocate all the training instances in each cell of the board ($\mu_u(\mathcal{T}_{ij}, \Sigma) = \Sigma$), or uniform mapping. Hence, all the cells in the board $\mathcal{T}$ contain the same training set. Therefore, the same environmental conditions are obtained in each cell. However, a mapping can perform non-uniform instance allocation. The pyramidal mapping $\mu_p(\mathcal{T}_{ij}, \Sigma)$ spreads the examples using a pyramid shape. Central cells contain all the available instances in Σ. The rest of the cells contains only a subset of Σ. Hence, the pyramidal mapping does not guarantee the same environment for each cell $\mathcal{T}_{ij}$. Instead, it guarantees that just few changes are introduced in adjacent cells of the neighborhood $\mathcal{T}_{ij}^{\nu}$. This means that few instances are removed when moving toward the cells in the outer part of the board $\mathcal{T}$. Under $\mu(\mathcal{T}_{ij}, \Sigma)$ assumptions, the classification complexity should grow when moving toward the inner cells of the board $\mathcal{T}$. Please refer to [10] for further details.

```
GALE(T, Σ)
   FOR-EACH T_ij ∈ T
   DO IN PARALLEL
      t ← 0
      initialize T_ij
      evaluate the accuracy of individual in T_ij using μ(T_ij, Σ)
      REPEAT
            t ← t+1
            merge individual in T_ij among T_ij^ν
            split individual in T_ij among T_ij^ν
            evaluate the accuracy of individual in T_ij using μ(T_ij, Σ)
            survival of T_ij among T_ij^ν
      UNTIL Ω(T_ij, t)
   DONE
RETURN T
```

Fig. 1. Pseudo-code of the algorithm running in each of the cells that compose GALE.

2.3 Control Flow

The evolutionary model proposed by GALE is based on the local interactions among cells. Each cell $\mathcal{T}_{ij}$ can only interact with the cells in the neighborhood $\mathcal{T}_{ij}^{\nu}$. Figure 1 presents the evolutionary process performed in each cell of GALE.

After the initialization phase, briefly explained in section 2.1, *merge* in GALE crosses the individual in the cell $\mathcal{T}_{ij}^{I}$ with one individual $I_{ij}^{\mathcal{M}}$ randomly chosen among its neighborhood $\mathcal{T}_{ij}^{\nu}$. The *merge* process occurs with a given probability $p_{\mathcal{M}}$. *Merge* generates only one individual $\mathcal{D}$ that replaces the individual in the cell $\mathcal{T}_{ij}$. The operator used for the genetic material recombination depends on the knowledge representation used for encoding $\mathcal{T}_{ij}^{I}$. Figure 2 shows this process schematically.

Then, *split* is applied with a given probability $p_s(\mathcal{T}_{ij}^{I}) = k_{sp} \cdot fit(\mathcal{T}_{ij}^{I})$, being $k_{sp} \in [0, 1]$ the maximum splitting rate. *Split* clones and mutates the individual in the cell. The new individual is placed in the empty cell $\mathcal{T}_{kl}$ of the neighborhood $\mathcal{T}_{kl} \in \mathcal{T}_{ij}^{\nu}$ with higher number of occupied cells in its neighborhood—$\max(\zeta(\mathcal{T}_{kl}^{\nu}))$. If all the cells of the neighborhood $\mathcal{T}_{ij}$ are occupied $\left(\zeta\left(\mathcal{T}_{ij}^{\nu}\right) = 8\right)$, the new individual is placed in the cell of the neighborhood $\mathcal{T}_{ij}^{\nu}$ that contains the worst individual—lowest fitness. Figure 3 illustrates this process graphically.

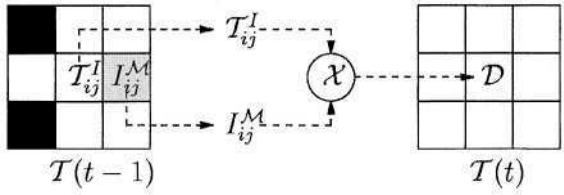
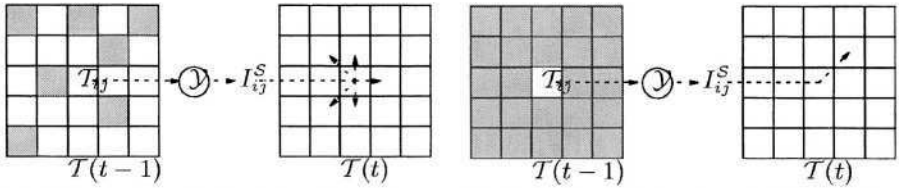


Fig. 2. Schematic representation of the merging process implemented by GALE.

The last step in the evolutionary cycle—*survival*—decides if the $\mathcal{T}_{ij}^{I}$ individual is kept for the next cycle or not. This process uses the neighborhood information. If a cell has up to one neighbor $(\zeta(\mathcal{T}_{ij}^{\nu}) \leq 1)$, then the probability of survival of the individual is $p_{sr}^{\zeta(\mathcal{T}_{ij}^{\nu})\leq 1}(\mathcal{T}_{ij}) = fit(\mathcal{T}_{ij}^{I})$. If a cell has seven or eight neighbors $\zeta(\mathcal{T}_{ij}^{\nu}) \geq 7$ then $p_{sr}^{\zeta(\mathcal{T}_{ij}^{\nu})\geq 7}(\mathcal{T}_{ij}) = 0$, where the individual is replaced by the best neighbor in $\mathcal{T}_{ij}^{\nu}$. On the other neighborhood configurations $(1 < \zeta(\mathcal{T}_{ij}^{\nu}) < 7)$, an individual survives if and only if $fit(\mathcal{T}_{ij}^{I}) \geq \overline{\mu}_{nei}^{\nu} + k_{sr} \times \sigma_{nei}^{\nu}$; $\overline{\mu}_{nei}^{\nu}$ is the average fitness value of the occupied neighbor cells $\mathcal{T}_{ij}^{\nu}$, and σ_{nei}^{ν} their standard deviation. k_{sr} is a parameter that controls the survival pressure over the current cell. For further details about GALE model, please see [6,14,11,10,12].

3 Heterogeneous Knowledge Representations

This section describes a first approach to break the constraint that individuals only use one kind knowledge representation. In order to test the feasibility of



(a) The new individual is placed in the empty cell $\mathcal{T}_{kl}$ of the neighborhood $\mathcal{T}_{kl} \in \mathcal{T}_{ij}^{\nu}$ with higher number of occupied cells in its neighborhood—$\max(\zeta(\mathcal{T}_{kl}^{\nu}))$.

(b) If all the cells in the neighborhood $\mathcal{T}_{ij}^{\nu}$ are occupied, then the new individual $I_{ij}^{\mathcal{S}}$ replaces the worst neighbor in $\mathcal{T}_{ij}^{\nu}$.

Fig. 3. The location of the new splitted individual is decided based on the current cell $\mathcal{T}_{ij}$ neighborhood $\mathcal{T}_{ij}^{\nu}$.

allowing the evolutionary mixing of knowledge representations, we focused on tree based representations for GALE. Our goal was to allow individuals to represent a solution using different types of knowledge representation for evolutionary tuning. The rest of this section briefly reviews the available tree representations in GALE, as well as the modifications introduced to allow the evolution of mixed trees.

3.1 Previous Homogeneous Tree Representations

GALE originally evolved three different tree knowledge representations [11,10, 12]. The simplest type of decision tree evolved by GALE is the one termed *orthogonal* (or axis-parallel) [16,20]. The internal nodes are a simple test over one attribute of the classification problem. This test is usually presented as:

$$a_i \leq \alpha \tag{1}$$

where a_i is a problem attribute and α is a numeric constant. The leaves of the tree are labeled with the class of the set of instances represented by the path between the root of the tree and the leaf itself. The classification boundaries defined by equation 1 are parallel to the axis of the instance space. This kind of tree has been effectively built, in the machine learning community, using heuristic algorithms based, for instance, on the *information gain* concept [16,20].

In order to overcome the limitations of parallel-axis boundaries, some authors proposed a more elaborate test for the internal nodes of the decision trees. *Oblique* decision trees [17,18] define the internal nodes using the following equation:

$$\sum_{i=1}^{d} \omega_i a_i + \omega_{d+1} > 0 \tag{2}$$

This test is based on an orientable hyperplane that can be adjusted via the vector of coefficients defined by $\omega = \langle \omega_1, \omega_2 \ldots \omega_{d+1} \rangle$. Thus, non parallel-axis boundaries can be easily defined choosing the right values for the ω vector. Unfortunately, finding w^* (the optimal data set split) is *NP-Hard* [21]. Therefore, the algorithms used for building this kind of tree use several heuristics to obtain suboptimal trees [18].

The last kind of decision tree evolved by GALE is a *multivariate* decision trees. This kind of tree defines non-linear boundaries on the instance space. In order to achieve this goal, each internal node contains a prototype δ [22,23](e.g. an artificially defined instance) and an activation threshold θ. A node is *active* if the following equation is satisfied

$$\sqrt{\sum_{i=1}^{d} (\delta_i - e_i)^2} \leq \theta \tag{3}$$

where e_i is the value of a_i in the instance e to classify. This selective activation defines hyperspheric boundaries across the instance space. Next, if the node is *active,* the instances e is given to the *nearest child* using the *nearest neighbor* algorithm [24]. Thus, children define non parallel-axis hyperplane splits inside the parent hyperspheres. Detailed descriptions of these decision trees can be found in [10].

3.2 Heterogeneous Tree Representations

The work presented in this paper explored the combination of different knowledge representations in a single individual. We focused on mixing two different kinds of decision trees (orthogonal and obliques.) This combination can be easily achieved due to the structural and functional similarity of the test nodes. For achieving such a goal, GALE needs to be slightly modified in two different places: (1) the initialization phase, (2) the *merge* phase.

The initialization phase, for a given cell $\mathcal{T}_{ij}$, now produces an orthogonal and oblique homogeneous tree. This choice is done at random, ideally filling the population with half orthogonal and half oblique homogeneous trees. The *merge* phase cuts and exchanges subtrees of two parent trees, regardless of whether they are orthogonal or oblique. Thus, *merge* mixes the different tree representations. Later on, these trees are evaluated using the original *survival* phase of GALE. It is important to mention here that the classification performed by such heterogeneous trees maintains the same hierarchical process. At any given node, the classification process relies on its type, performing orthogonal- (equation 1) or oblique-based (equation 2) classification accordingly.

4 Experiments

We conducted two different kinds of experiments for testing the mixed trees presented in the previous section. The first set of experiments used artificially-generated data sets. The goal was to test the implementation of orthogonal,

oblique, and mixed trees. These experiments also let us perform a first comparison between GALE and traditional orthogonal and oblique decision tree inducers [17,25,26]. The second set of experiments focused on studying the behavior of GALE and other tree learners mainly on data sets provided by the UCI repository [27].

4.1 Classifier Schemes

Besides GALE, tree other non-evolutionary decision tree inducers were tested:

- **C4.5 revision 8** [16,20]
- **CART-LC** [17]
- **OC1** [26]

A detailed description of these algorithms is beyond the scope of this paper. However, it is important to mention here that C4.5 is a well-know inducer of orthogonal decision trees. CART-LC is an oblique decision tree inducer, as is OC1. We also tested OC1 in its orthogonal inducer facet.

4.2 Artificially-Generated Data Sets

We prepared different data sets for this first test. All the artificially-generated data sets are defined on a bidimensional space for binary classification problems. Figure 4 displays each of these data sets. The data set instances were obtained sampling the given classification space using an uniformly distributed rectangular grid. Three axis-parallel classification data sets (`ORT-1`, `ORT-2`, and `ORT-3`), three oblique classification data sets (`OBL-1`, `OBL-2`, and `OBL-3`), and one mixed data set (`MIXED`) were generated. `ORT-1`, `ORT-2`, `OBL-1`, and `OBL-2` contain 400 instances each, whereas `ORT-3`, `OBL-3`, and `MIXED` the number of artificial instances raises till 1600.

The ultimate goal of the data sets presented in figure 4 was to test the performance of GALE on problems designed by axis-parallel, oblique, and mixed classification boundaries. Such tests used the three different tree knowledge representation presented in section 3.2. In order to compare the quality of the results produced by GALE, the outputted tree was compared to the one produce by OC1. For `ORT-1`, `ORT-2`, and `ORT-3` GALE evolved homogeneous orthogonal trees, and OC1 was set to induce orthogonal decision trees. In a similar way, for `OBL-1`, `OBL-2`, and `OBL-3` GALE and OC1 produced oblique trees. Finally for the `MIXED` data set, GALE produce heterogeneous trees, where as OC1 was tested in both its orthogonal and oblique tree facets.

Both algorithms were run using the parameters described in GALE [11] and OC1 [26] original papers. Since the goal of this test was to learn more about the behavior of these algorithms in the artificially-generated problems, we used the whole data sets for training, and inspected the outputted trees. Both algorithms discovered the overall classification structure correctly approximating the classification boundaries presented in figure 4. For each kind of data set,

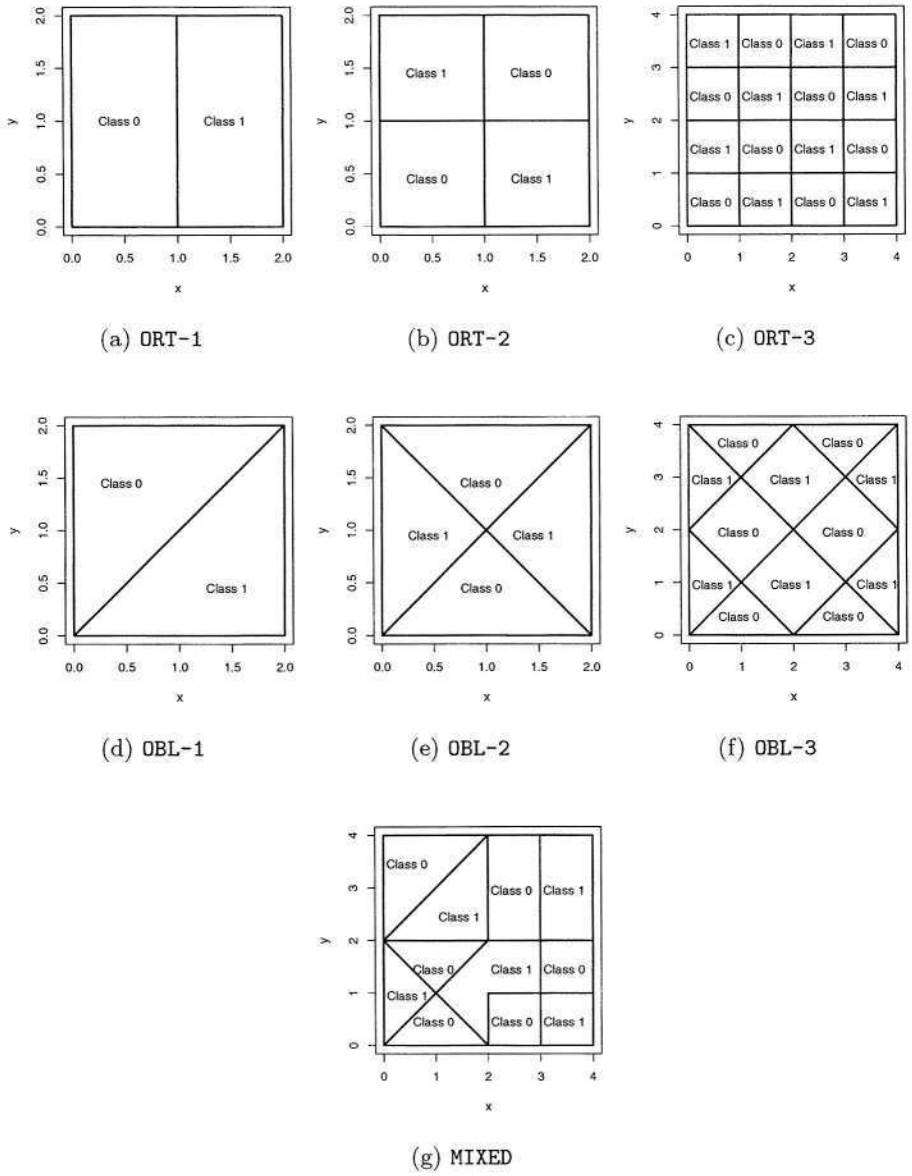


Fig. 4. Artificially generated data sets for initial testing.

the appropriate knowledge representation was chosen, as explained above. An special case is OC1 and the `MIXED` data set, where we ran it for the two available knowledge representations. Table 1 summarizes the results achieved.

Table 1. GALE and OC1 results (classification accuracy and number of leaves) on artificially-generated data sets. For each data set, the appropriate knowledge representation was used. OC1 induced both orthogonal and oblique trees for the MIXED data set.

	ORT-1	ORT-2	ORT-3	OBL-1	OBL-2	OBL-3	MIXED
GALE	100.00(2)	100.00(4)	100.00(16)	100.00(2)	100.00(6)	98.84(20)	98.88(25)
OC1	100.00(2)	100.00(6)	100.00(20)	100.00(2)	99.00(19)	97.75(46)	98.38(73)/97.69(29)

The classification mistakes made by OC1 and GALE were done at intersection points of classification boundaries, or at the extremes of the hyperplanes. OC1 presented a strong tendency to produce overfitted trees—even after pruning them. Such tendency is clearly stated in the orthogonal data sets, where GALE was the only one able to evolve a perfect solution (minimal orthogonal tree that matches the classification boundaries.) This behavior was also observed on the oblique data sets runs. Finally, in the `MIXED` data set GALE also took advantage of evolving mixed trees, whereas when OC1 induced orthogonal trees it displayed a *staircase effect* on the oblique decision boundaries. The *staircase effect* [28] emerges in parallel-axis learners if there are non-axis-parallel classification boundaries in the problem. To define such boundaries the learner is forced to produce large subtrees.

4.3 UCI Repository Data Sets

The second kind of experiment used data sets from the UCI repository [27]. The data sets used were: (1) *Wisconsin breast cancer* (`bre`), (2) *glass identification* (`gls`), (3) *heart disease from statlog project* (`h-s`), (4) *ionosphere* (`ion`), (5) *iris* (`irs`), (6) *Pima-indian diabetes* (`dia`), (7) *sonar* (`son`), and (8) *vehicle from statlog project* (`veh`). A detailed explanation of these data sets is beyond the scope of this paper. For further details, please refer to [27]. Another non-UCI data set was also used in these experiments. The *tao* (`tao`) data set, although being artificially-generated, presents non-linear classification boundaries. This problem was firstly introduced by Llorà & Garrell [14]. The criteria for selecting these data sets was to explore problems with a wide range of dimensions, as well as different numbers of classes.

In order to compare the performance of the different algorithms in terms of classification accuracy, the following methodology was used. Classification accuracy was estimated using stratified ten-fold cross-validation runs. To estimate the difference in performance between the proposed framework for mixed tree evolution and the previous algorithms presented in sections 2 and 4.1, a paired t-test was used [29]. Table 3 summarizes the results achieved using this methodology.

The experiments revealed several interesting insights about the algorithms and the selected problems. If we just take a plain look at the results presented in table 3, the results of GALE evolving mixed trees confirm the viability of the in-

Table 2. Experimental results: percentage of correct classifications and standard deviation from stratified ten-fold cross-validation runs. Results are also marked with a ∘ if they show a significant improvement (1% significant level on paired two-sided t-test) over the corresponding GALE-mix results, and with a • if they show a significant degradation.

DS	C4.5r8	OC1-ort	GALE-ort	CART-LC	OC1-obl	GALE-obl	GALE-mix
brs	95.42±1.69	94.79±1.22	94.42±1.88	95.86±2.37	95.46±3.42	91.70±3.24	95.10±2.10
gls	65.89±10.47	65.19±10.89	65.42±11.89	65.45±14.52	59.81±9.55	49.07±9.20•	65.19±7.27
h-s	76.30±5.85	77.41±8.27	82.22±7.11	76.30±6.34	78.15±8.98	71.11±7.35	79.64±9.11
ion	89.74±5.23	89.18±7.69	94.02±3.27	84.57±3.61•	90.01±4.11	90.31±3.57	91.52±5.63
irs	95.33±3.26	93.33±8.32	96.00±3.46	94.00±6.63	95.33±6.33	98.67±2.98°	95.33±3.05
pmi	73.05±5.32	74.57±4.67	75.78±4.01	72.86±4.25	72.00±5.52	69.40±3.24•	73.60±5.88
son	71.15±8.54	72.57±13.04	74.52±7.42	68.14±5.94	64.79±17.24	68.27±10.03	71.57±11.32
tao	95.07±2.11°	95.06±1.57°	97.03±2.52°	96.23±1.48°	89.78±2.29•	91.74±2.65	91.31±1.53
veh	73.64±5.42°	71.23±5.32°	68.32±6.01°	68.33±6.48°	71.97±4.09°	58.87±5.37•	63.84±3.04
Average	80.04	81.56	81.24	80.19	80.51	75.82	80.79

Table 3. Experimental results: percentage of correct classifications and standard deviation from stratified ten-fold cross-validation runs. Results are also marked with a ∘ if they show a significant improvement (1% significant level on paired two-sided t-test) over the corresponding GALE-mix results, and with a • if they show a significant degradation.

DS	C4.5r8	OC1-ort	GALE-ort	CART-LC	OC1-obl	GALE-obl	GALE-mix
brs	95.42±1.69	94.79±1.22	94.42±1.88	95.86±2.37	95.46±3.42	91.70±3.24	95.10±2.10
gls	65.89±10.47	65.19±10.89	65.42±11.89	65.45±14.52	59.81±9.55	49.07±9.20•	65.19±7.27
h-s	76.30±5.85	77.41±8.27	82.22±7.11	76.30±6.34	78.15±8.98	71.11±7.35	79.64±9.11
ion	89.74±5.23	89.18±7.69	94.02±3.27	84.57±3.61•	90.01±4.11	90.31±3.57	91.52±5.63
irs	95.33±3.26	93.33±8.32	96.00±3.46	94.00±6.63	95.33±6.33	98.67±2.98°	95.33±3.05
pmi	73.05±5.32	74.57±4.67	75.78±4.01	72.86±4.25	72.00±5.52	69.40±3.24•	73.60±5.88
son	71.15±8.54	72.57±13.04	74.52±7.42	68.14±5.94	64.79±17.24	68.27±10.03	71.57±11.32
tao	95.07±2.11°	95.06±1.57°	97.03±2.52°	96.23±1.48°	89.78±2.29•	91.74±2.65	91.31±1.53
veh	73.64±5.42°	71.23±5.32°	68.32±6.01°	68.33±6.48°	71.97±4.09°	58.87±5.37•	63.84±3.04
Average	80.04	81.56	81.24	80.19	80.51	75.82	80.79

tuition presented in section 4.2. Results showed that for some problems parallel-axis (`gls`) or oblique (`irs`) boundaries were preferable. In other problems, there was no clear preference for any of the available knowledge representations (`brs`.)

However, the results became more interesting when we took a look at the evolved mixed trees. GALE required a little longer to evolve competent mixed trees—proved in informal experiments. Hence, the accuracy achieved by GALE evolving mixed trees was slightly lower However, we maintained a common configuration among all the data set in favor of a fair comparison. Nevertheless, we are already addressing this issue as part of our further work. Mixed trees, as result of the evolutionary guidance, adapted the knowledge representation to the needs of each of the different data sets explored. A clear tendency to obtain the right solution, in terms of knowledge representation, always emerged in GALE-mix runs. This result is encouraging, since it confirms our intuition that the bias introduced by the knowledge representation can be minimized by mixing different kinds of trees, under the guidance of evolution. This tendency also encourages us to pursue a better understanding of GALE-mix behavior using the Illinois decomposition methodology [30,31].

5 Conclusions

The knowledge representation language has been traditionally chosen beforehand, tailoring the evolutionary algorithm around it. Such a decision may constraint—*a priori*—the concepts that can be learned and the possible usage of such frameworks in real-world problems. In order to minimize this bias, we proposed the evolutionary mixing of different knowledge representations. The work presented in this paper modified an existing LCS framework, allowing the mixing of different knowledge representations in the individuals of the population.

The experiments showed promising initial results. Besides showing that mixing different knowledge representations under the guidance of evolution is possible, results also showed that mixing helped minimize the bias introduced a priori. The experiments showed how the population adapted the knowledge representations used in its individuals to fit the problem to be solved, with little extra evolutionary machinery. This adaptive behavior encourages us to conduct further research on combining other kinds of representations, as well as introducing specialized mechanisms for achieving this purpose. Further research will also include a theoretical insight into GALE using the Illinois decomposition methodology.

Acknowledgments. This work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant F49620-03-1-0129, and by the Technology Research, Education, and Commercialization Center (TRECC), at University of Illinois at Urbana-Champaign, administered by the National Center for Supercomputing Applications (NCSA) and funded by the Office of Naval Research under grant N00014-01-1-0175. The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research, the Technology Research, Education, and Commercialization Center, the Office of Naval Research, or the U.S. Government.

References

1. Janikow, C.Z.: A genetic algorithm for optimizing fuzzy decision trees. Proceedings of the Sixth International Conference on Genetic Algorithms (1995) 421–428
2. Kennedy, H., Chinniah, C., Bradbeer, P.V.G., Morss, L.: The construction and evaluation of decision trees: a comparison of evolutionary and concept learning methods. In: Selected Papers from AISB Workshop on Evolutionary Computing, Springer-Verlag (1997) 147–162
3. Ryan, M.D., Rayward-Smith, V.J.: The evolution of decision trees. Genetic Programming 98 (1998) 350–358

4. Folino, G., Pizzuti, C. Spezzano, G.: Genetic programming and simulated annealing: A hybrid method to evolve decision trees. Genetic Programming: Third European Conference (2000) 294–303
5. Tanigawa, T., Zhao, Q.: A study on efficient generation of decision trees using genetic programming. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2000) (2000) 1047–1052
6. Llorà, X., Garrell, J.M.: Evolution of Decision Trees. Forth Catalan Conference on Artificial Intelligence (CCIA'2001) (2001) 115–122
7. Papagelis, A., Kalles, D.: GA Tree: genetically evolved decision trees. 12th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'00) (2000) 203–206
8. Fu, Z., Golden, B., Lele, S., Wasil, E.: A Genetic Algorithm-based Approach for Building Accurate Decision Trees. INFORMS Journal on Computing **15** (2003) 3–22
9. Cantu-Paz, E., Kamath, C.: Using evolutionary algorithms to induce oblique decision trees. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000) (2000) 1053–1060
10. Llorà, X.: Genetic Based Machine Learning using Fine-grained Parallelism for Data Mining. PhD thesis, Enginyeria i Arquitectura La Salle. Ramon Llull University, Barcelona (February, 2002)
11. Llorà, X., Garrell, J.M.: Knowledge-Independent Data Mining with Fine-Grained Parallel Evolutionary Algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2001), Morgan Kaufmann Publishers (2001) 461–468
12. Llorà, X., Garrell, J.M.: Coevolving different knowledge representations with fine-grained parallel learning classifier systems. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002), New York, Morgan Kaufmann Publishers (2002) 934–941
13. Brodley, C.E.: Addressing the seletive superiority problem: Automatic algorithm/model class selection. In: Proceedings of the 10th International Conference on Machine Learning, Morgan Kaufmann (1993) 17–24
14. Llorà, X., Garrell, J.M.: Evolving Partially-Defined instances with Evolutionary Algorithms. In: Proceedings of the 18th International Conference on Machine Learning (ICML'2001), Morgan Kauffmann (2001) 337–344
15. Llorà, X., Garrell, J.M.: Prototype induction and attribute selection via evolutionary algorithms. Intelligent Data Analysis **7** (2003) 193–208
16. Quinlan, R.: Induction of decission trees. Machine Learning, Vol. 1, No. 1 (1986) 81–106
17. Breiman, L., Friedman, J., Olshen, R., Stone, C.: Classification and Regression Trees. Wadsworth International Group (1984)
18. Van de Merckt, T.: Decision trees in numerical attribute spaces. In: Proceedings of the 13th Intenational Joint Conference on Artificial Intelligece, Morgan Kaufmann (1993) 1016–1021
19. Jong, K.A.D., Spears, W.M.: Learning Concept Classification Rules using Genetic Algorithms. In: Proceedings of the Twelfth International Conference on Artificial Intelligence IJCAI-91. Volume 2., Morgan Kaufmann (1991) 651–656
20. Quinlan, J.R.: C4.5: programs for machine learning. Morgan Kaufmann (1993)
21. Heath, D., Kasif, S., Salzberg, S.: Learning oblique decision trees. In: Proceedings of the 13th Intenational Joint Conference on Artificial Intelligece, Morgan Kaufmann (1993) 1002–1007

22. Domingos, P.: Rule Induction and Instance-based Learning: A Unified Approach. In: Proceedings of the 14th International Joint Conference on Artificial Intelligence, Morgan Kaufmann (1995) 1226–1232
23. Wettschereck, D.: A hybrid Nearest-Neighbor and Nearest-Hyperrectangle Algorithm. In: Proceedings of the 7th European Conference on Machine Learning, LNAI. Volume 784. (1994) 323–335
24. Aha, D.W., Kibler, D., Albert, M.K.: Instance-based Learning Algorithms. Machine Learning **6** (1991) 37–66
25. Quinlan, J.R.: Induction of decision trees. Machine Learning **1** (1986) 81–106
26. Murthy, S.K., Kasif, S., Salzberg, S.: A system for induction of oblique decision trees. Journal of Artificial Intelligence Research **2** (1994) 1–32
27. Merz, C.J., Murphy, P.M.: UCI Repository for Machine Learning Data-Bases [http://www.ics.uci.edu/~mlearn/MLRepository. html]. Irvine, CA: University of California, Department of Information and Computer Science (1998)
28. Abbott, D.W.: Combining models to improve classifier accuracy and robustness. In: Proceedings of Second International Conference on Information Fusion. (1999) 289–295
29. Dietterich, T.G.: Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms. Neural Computation **10** (1998) 1895–1924
30. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading, Mass. (1989)
31. Goldberg, D.E.: The Design of Innovation: Lessons from and for Competent Genetic Algorithms. Kluwer Academic Publishers, Norwell, MA (2002)

Improving MACS Thanks to a Comparison with 2TBNs

Olivier Sigaud, Thierry Gourdin, and Pierre-Henri Wuillemin

LIP6
8, rue du Capitaine Scott
75015 PARIS

Abstract. Factored Markov Decision Processes is the theoretical framework underlying multi-step Learning Classifier Systems research. This framework is mostly used in the context of Two-stage Bayes Networks, a subset of Bayes Networks. In this paper, we compare the Learning Classifier Systems approach and the Bayes Networks approach to factored Markov Decision Problems. More specifically, we focus on a comparison between MACS, an Anticipatory Learning Classifier System, and Structured Policy Iteration, a general planning algorithm used in the context of Two-stage Bayes Networks. From that comparison, we define a new algorithm resulting from the adaptation of Structured Policy Iteration to the context of MACS. We conclude by calling for a closer communication between both research communities.

1 Introduction

As [Lan02] did show very clearly, Learning Classifier Systems (LCSs) are a family of Reinforcement Learning (RL) systems endowed with a generalization property. The usual formal representation of RL problems is a Markov Decision Process (MDP), which is defined by a finite state space S, a finite set of actions A, a transition function T and a reward function R.

In the LCS framework, the states consist of several independent attributes whose value define a perceptual situation. Indeed, instead of the (state, action, payoff) triples in the Q-Table of standard RL algorithms, a LCS contains rules called "classifiers" organized into a [`condition`] part (or `C` part), an [`action`] part (or `A` part) and a payoff. The `C` part specifies the value of a set of attributes. It is satisfied only if all values specified match the corresponding values in the current state.

In the LCS framework, finding a compact representation is seen as a generalization problem. The generalization capability of LCSs comes from their use of *don't care* symbols # in the `C` part of the classifiers. Indeed, a # *matches* any particular value of the considered attribute. Therefore, changing an attribute into a # makes the corresponding `C` part more general (it matches more situations). As a result, the representation is more compact.

The same framework is also used in a sub-part of the Bayes Networks (BNs) community, under the name "factored MDP". Indeed, a factored MDP is a MDP

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 810–823, 2004.

where each state is a collection of random variables. This representation takes advantage on local probabilistic conditional dependences among some variables to build a more compact model of the underlying MDP by factoring the dependent variables. More precisely, according to [BDH99], if the situation is composed of n variables such that any of them only depends of the value of l other variables at the previous time step, then the size of the factored representation is in $O(n2^l)$ instead of $O(2^n)$.

Thus the intuition giving rise to the use of factored MDPs in BNs seem to be exactly the same as the intuition underlying LCS research, particularly in the multi-step context. In both cases, the goal is to solve MDPs with a compact represent ation.

Given this identity, we present a comparison between both perspectives, showing that 2TBN, a formalism devoted to solving factored MDPs in BNs, is very similar to the one of MACS, an Anticipatory Learning Classifier System (ALCS). Then, as a result of the comparison, we show how the Structured Policy Iteration (SPI) algorithm designed in the 2TBN case can be adapted to MACS and we discuss the improvements that result from this adaptation.

The paper is organized as follows. In section 2, we briefly present the formalism used in MACS. In section 3, we present how 2TBNs tackle factored MDP problems and we illustrate their formalism through an example used as a basis for a comparison. In section 4, we compare the 2TBN formalism with the one used in MACS, and in section 5, we show how to adapt SPI to MACS. We discuss in section 6 the insights that result from this adaptation both by the 2TBN side and by the ALCS side. Finally, we conclude by calling for more exchanges between both research communities.

2 Brief Overview of MACS

2.1 Anticipatory Learning Classifier Systems

As indicated in the introduction, among RL systems, LCSs are a family of systems designed to solve problems where the state consists of several attributes. They take advantage of factored representations with respect to plain RL techniques tabular *Q-learning* thanks to their generalization capability. As a consequence, they can solve problems with fewer classifiers than a system dealing with tabular representations.

Most standard LCSs call upon a combination of a RL algorithm such as *Q-learning* with a Genetic Algorithm (GA) [Gol89]. The RL algorithm estimates the quality of actions in the situations specified by the different classifiers, while the GA evolves the population of classifiers. Thus each classifier holds both an estimated quality and a fitness. In *strength-based* LCSs such as ZCS [Wil94], the fitness is equal to the estimated quality. As a result, only the classifiers specifying actions with a high payoff are kept. In *accuracy-based* LCSs such as XCS [Wil95], the fitness is equal to the capacity of the classifier to accurately predict the payoff it will receive. This results in a more efficient coverage of the (state, action) space, and XCS is now the most widely used LCS. For a general overview of recent LCS research, see [LR00].

Instead of directly learning a model of the quality function as standard LCSs do, Anticipatory Learning Classifier Systems (ALCSs) such as ACS [Sto98, BGS00], ACS2 [But02], YACS [GS01,GSS02] and MACS [GMS03,GS03] learn a model of the transition function *T*. This model is then used to speed up the RL process.

In ALCSs, the classifiers are organized into [condition] [action] [effect] parts, denoted C−A−E. The E part represents the effects of action A in situations matched by condition C. It records the perceived changes in the environment. Different formalisms giving rise to generalization in this approach are described hereafter.

2.2 From YACS to MACS

In ACS, ACS2 and YACS, a C part may contain *don't care* symbols "#" or specific values (like 0 or 1), as in XCS. An E part may contain either specific values or *don't change* symbols "= ", meaning that the attribute remains unchanged when the action is performed. A specific value in the E part means that the value of the corresponding attribute *changes* to the value specified in that E part.

Unfortunately, such a formalism cannot represent regularities across different attributes from one time step to another, though such regularities are common.

Table 1. During the integration process, MACS scans the E parts and selects classifiers whose A part matches the action and whose C part matches the situation. The integration process builds all the possible anticipated situations with respect to the possible values of every attribute. Here, MACS anticipates that using [01020100] as a current situation should lead either to [11012000] or to [11011000]. If all the classifiers were accurate, this process would generate only one possible anticipation.

[01020100]	[Action]	*Anticipated situation*
[0#######]	[East]	[???????0]
[#1######]	[East]	[1???????]
[##02####]	[East]	[?????2???]
[###2####]	[East]	[????1???]
[####0###]	[East]	[?????0??]
[#####1##]	[East]	[?1??????]
[######0#]	[East]	[??0?????]
[#######0]	[East]	[???1????]
[########]	[East]	[??????0?]
Resulting anticipations	→	[11012000]
	or	[11011000]

In order to discover more regularities, our second ALCS, MACS [GMS03, GS03], uses *don't know* symbols "?" instead of *don't change* symbols in the E part. Thanks to that formalism, the overall system gains the opportunity to discover regularities across different attributes in the C and the E parts. As a result of this modification, we can split the model of transitions into modules predicting

the value of different attributes. The consequence is that the system needs an additional mechanism which *integrates* the partial anticipations provided by the modules and builds a whole anticipated situation, without any *don't know* symbol in its description, as shown in Table 1.

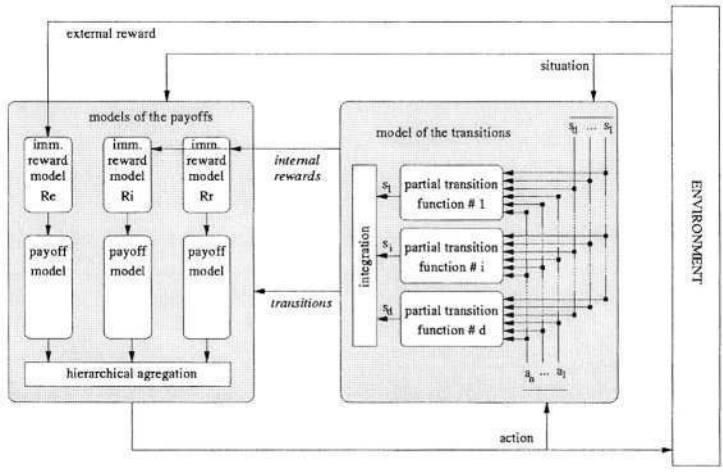


Fig. 1. MACS global architecture.

2.3 The Architecture of MACS

The MACS architecture is illustrated in figure 1. As shown in [GS03], it is similar to a DYNA architecture in all respects, but it is additionally endowed with a generalization capability. It consists of a model of transitions, which models the dynamics of the agent-environment interactions, and a model of the payoff, which models the payoffs that the agent can get from its actions. The main processes in MACS are devoted to learning the model of transitions thanks to dedicated generalization and specialization heuristics, and learning the model of the payoff, according to different criteria that endow MACS with an active exploration capability. These processes have already been presented in detail in [GMS03]. We must just mention that MACS uses the model of transitions to speed up the learning of the model of the payoff thanks to a Value Iteration algorithm that is somewhat inefficient: the model of the payoff consists in storing two values for each encountered state without generalization, and Value Iteration requires a costly lookahead operation involving the anticipations integration process described in section 2.2.

Experimental results presented in [GMS03] demonstrated that the new formalism used by MACS actually affords more powerful generalization capacities than the former ones, without any cost in terms of learning speed.

3 DBNs and 2TBNs

As we have said in the introduction, a factored MDP is a MDP where all states are defined as collections of random variables X_i. The set of states can be denoted $S = (X_1, ..., X_n)$. Each variable X_i takes its value x_i in a domain: $\forall i, x_i \in Dom(X_i)$. Thus a given state s is defined by a vector of values of the random variables: we have $s = (x_1, ..., x_n)$ which defines the states exactly as in LCSs. A Bayesian Network (BN) is a tool to deal with probabilistic dependencies among these random variables.

The BN framework [Pea88] includes a graphical formalism devoted to the representation of conditional relations between variables. Graphically, variables are represented as nodes in a Directed Acyclic Graph. A link between two nodes represent a probabilistic dependency between the corresponding variable such that the joint probability of all variables can be decomposed into a product of conditional probabilities of each variable, given its parents in the graph. Hence BNs provide an easy way to specify conditional independences between variables. Moreover, they provide a compact parameterization of the model.

Dynamic Bayesian Networks (DBNs) [DK89] are BNs of stochastic processes, representing (temporal) sequential data. They extend BNs by focusing on modeling changes on stochastic variables ($X_{t+1} = f(X_t)$) over time with a BN. Hence the sequence is infinite but the time slice is finite and (usually) static. DBNs generalize Hidden Markov Models (HMMs), Linear Dynamical Systems (LDSs) and Kalman filters, and represent (hidden and observed) states in terms of probabilistic variables. Note that the assumption of time slice invariability (relations do not change over time) may be relaxed for the parameters as well as for the structure. Usually, a DBN is represented by 2 time slices, under the name 2TBN (2 Time (slices) Bayesian Network). The former slice represents the initial state of the model; the second one represents the generic state. The relations between the two slices represent the stochastic process. In the following we will consider restrictions of 2TBNs where there are no relations between variables in the second time slice.

With respect to the standard BNs, this particular class is very restricted, but it is nevertheless rich enough to adequately represent factored MDPs and give rise to tractable algorithms, whereas more powerful representations are generally untractable.

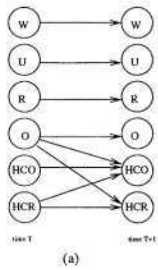
In order to show how 2TBNs are used to model a factored MDP, we will use the example given in [BDG00]. In this example, a robot must go to a café to buy some coffee and deliver it to its owner in her office. It may rain on the way, in which case the robot may get wet, unless it has an umbrella. There are six boolean propositions describing the state of the system:

- O: the robot is at the office ($\overline{O}$ means that it is at the café);
- R: it is raining; W: the robot is wet;
- U: the robot has its umbrella;
- HCR: the robot has coffee; HCO: the owner has coffee.

The robot can take four actions:

– *Go:* move to the opposite location; *GetU*: get the umbrella.
– *BuyC*: buy coffee; *DelC*: deliver coffee to the user;

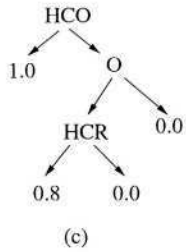
A 2TBN can be used to model the effect of each action on the state of the problem. Such 2TBNs are called "action networks" [DG94]. In figure 2.a, we show the action network corresponding to the action "DelC".



(a)

O^t	HCR^t	HCO^t	HCO^{t+1}
t	t	t	1.0
f	t	t	1.0
t	f	t	1.0
f	f	t	1.0
t	t	f	0.8
f	t	f	0.0
t	f	f	0.0
f	f	f	0.0

(b)



(c)

Fig. 2. (a) The action network for *DelC*, represented as a 2TBN; (b) The CPT for *HCO/DelC*; (c) The corresponding tree representation

From this network we can see for instance that, when the robot delivers coffee, the fact that he is wet afterwards does only depend on whether he was wet beforehand, since delivering coffee does not happen outdoors. On the contrary, the fact that the owner finally gets some coffee depends on three propositions: is the robot at the office? did the robot have coffee? did the owner have coffee?

There is one such network per action. But these networks do only indicate whether there is a probabilistic dependency or not, they do not provide enough information to determine the actual dynamics of the environment. The missing information is represented in Conditional Probabilistic Tables (CPTs). An example of CPT is shown in figure 2.b. This table gives the probability that *HCO* will hold depending on *O*, *HCR* and *HCO* if the agent takes action *DelC*. Finally, the same information can be represented in a tree, as shown in figure 2.c. One such table or tree must be given for each variable and for each action in order to get a complete specification of the dynamics of the factored MDP.

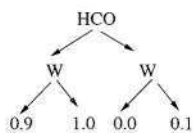


Fig. 3. The reward tree for the café example

Finally, one can also represent the reward function of the problem with a decision, tree, as shown in figure 3. Let us now compare this formalism with the one used in MACS before comparing the processes used to deal with these representations.

4 Comparison of Formalisms

First of all, at the level of the organization of data, the representations in 2TBNs and MACS are slightly different.

In MACS, each module in the model of transitions tries to predict the value of one particular attribute for all possible situations and all possible actions, while in the standard BN representation, there is one 2TBN for each possible action and then, for any such action, there is one CPT for each predicted attribute.

Thus in the 2TBN representation, the distinction between actions is prior to the distinction between attributes, while it is the contrary in MACS. But this is just a different organization for the same data.

At a lower level, the content of these data is similar. Indeed, for a given action, CPTs convey the same information as a subset of possible modules dedicated to predicting the same attribute for the same action in MACS, as shown in table 2.

Table 2. The CPT for $HCO/DelC$ represented with the formalism of MACS

W	U	R	O	HCR	HCO	Action	W	U	R	O	HCR	HCO	missing proba.
#	#	#	#	#	1	DelC	?	?	?	?	?	1	**1.0**
#	#	#	1	1	0	DelC	?	?	?	?	?	1	**0.8**
#	#	#	#	0	0	DelC	?	?	?	?	?	0	**1.0**
#	#	#	0	#	0	DelC	?	?	?	?	?	0	**1.0**

Thus expressing a CPT as a set of MACS classifiers is straightforward: for each leaf in the tree, add a new classifier with a # in front of all irrelevant variables in the `C` part, the considered action in the `A` part, and an `E` part whose only specialized attribute is the one predicted by the CPT.

When this is done, one obvious difference remains: the CPT conveys a probability that the attribute will take the predicted value, while MACS conveys a deterministic information. Indeed, MACS was designed to solve deterministic MDPs and should be extended in that direction if we want to reduce the distance between both formalisms.

Another important difference comes from the way the payoff is modeled in both approaches. In MACS we map the values to all situations encountered, giving rise to a flat (situation, value) table. On the contrary, [BDG95] indicates that the reward function can be represented as a tree whose nodes are variables, as shown in figure 3. This gives rise both to a more compact representation and to more efficient algorithms, as we will discuss now.

5 Incorporating a 2TBN Planning Algorithm in MACS

The main difference between mechanisms in MACS and in 2TBNs comes from the fact that MACS is designed to learn by itself a representation from its interaction with the MDP thanks to a RL mechanism, while in most 2TBNs cases the representation is considered as given by the user.

Thus MACS must solve both a learning problem and a planning problem while most 2TBNs just solve the planning problem. Note however that some authors are interested in learning the structure of DBNs [FMR98,HGC95,Gha97], but in general they do so separately from tackling the planning problem, drawing inspiration from algorithms developed for Hidden Markov Models [RJ86], and they focus on more complex structures than 2TBNs.

As a consequence of this difference, in order to go further in the comparison, we must focus on the way the planning problem is solved both in MACS and in 2TBNs.

5.1 Structured Policy Iteration

The algorithm we will present in order to build an optimal policy in 2TBNs is Structured Policy Iteration (SPI) [BDG95,BDG00]. SPI is inspired from the tabular Modified Policy Iteration (MPI) algorithm [How71,PS78] improved in order to work exclusively on tree structures, which results in favorable cases in a significant reduction of the computational cost of the algorithm with respect to tabular MPI. For a more detailed presentation of SPI than the one given below, read [BDG00].

The central operation common to all dynamic programming and reinforcement learning algorithms is the propagation of values or qualities among the states of the MDP [1], according to the Bellman equation:

$$\forall s_i \in S, Q_a^V(s_i) = R(s_i, a) + \beta \sum_{s_j \in S} T(s_i, a, s_j) V(s_j) \quad (1)$$

The key idea in SPI consists in achieving this propagation more efficiently than in tabular representations by grouping together the states that share the same values into tree-based representations. Indeed, all structures in SPI are decision trees whose leaves correspond to groups of states sharing the same values on the variables present in the parent nodes of these leaves.

In this tree-based representation, we note:

- $QTree_a^V(s)$ the quality tree of action a given the value function V;
- $RTree(s, a)$ the tree of rewards obtained from performing action a in state s;
- $VTree(s)$ the value tree of state s.

[1] This process is called "Decision-Theoretic Regression" in [BDG00].

As a consequence, the Bellman equation becomes:

$$\forall s_i \in S, QTree_a^V(s_i) = RTree(s_i, a) + \beta \sum_{s_j \in S} T(s_i, a, s_j) VTree(s_j) \qquad (2)$$

The value propagation algorithm in the tree-based case strictly follows the standard algorithm for the tabular case:

- It starts with an initial value function $VTree_0(s) = RTree(s, a)$.
- Then, in order to update $QTree_a^V(s)$, it looks for the term $T(s_i, a, s_j)$, i.e. for transitions from one state to the next through actions, so as to propagate the values along these transitions. Instead of looking for these transitions in a (s_t, a_t, s_{t+1}) table, the tree-based algorithm uses the *CPT*s of all actions, which directly give a tree-based representation of the same information.

This algorithm can infer the tree-based representation of the quality of all actions from a tree-based representation of the current value function and back, so as to perform a tree-based value iteration, that we will call Structured Value Iteration (SVI) hereafter. The details of this algorithm can be found in [BDG00].

Taking SVI as a basic component, SPI can be decomposed into two stages as for all Policy Iteration algorithms: a Structured Policy Evaluation [2] stage, and a Structured Policy Improvement stage. The former consists in evaluating the long term reward that an agent can expect from a given policy. It is based on the estimation of the value function explained above. The latter consists in improving the current policy according to the new value function calculated by the former. It can be seen as a variant of the former where, instead of labelling the leaves of $VTree(s)$ with the maximum estimated values, the algorithm labels them with the actions that deliver these maximum values.

5.2 CbVI, a Classifier-Based SVI

As we briefly mentioned in section 2.3, in MACS, determining the reachable situations is an expensive process, since it implies the partial anticipations integration process described in section 2.2. This process is used both when the agent controlled by MACS chooses the next action and when the model of the payoff is improved thanks to a Value Iteration algorithm. Thus it would be interesting to improve this part of MACS by drawing inspiration from the way SVI is performed in 2TBNs.

In order to do so, we must distinguish two different sub-problems:

- **The planning problem:** Given a perfect model of transitions under the form of a list of classifiers and a perfect knowledge of the reward function, adapt SVI to work with classifiers. We deal with that sub-problem below.
- **The learning problem:** Adapt SVI or its classifier-based version to the case where the model of transitions is learned simultaneously with looking for an optimal policy. We keep this second sub-problem for the final discussion.

[2] called Structured Successive Approximation (or SSA) in [BDG00].

One way to integrate the insights from the SVI algorithm into MACS would be to derive from the classifier-based representation of the model of transitions a tree-based representation and then apply SVI as such.

In this section we rather present our adaptation of SVI to work directly with a classifier-based representation.

Let us call "token" the specification of a value or a # for one attribute, and let us call "message" a set of k tokens, where k is the number of attributes that define a state. In particular, the C part of classifiers are messages.

Our algorithm, called CbVI, heavily relies on an associative and commutative intersection operator $\cap$ between two messages. The intersection between two tokens T_a and T_b is defined as follows (we note x_a the value of token a):

$$T_a \cap T_b = \begin{cases} x_a & \text{if } x_a = x_b \text{ or } x_b = \# \\ x_b & \text{if } x_a = \# \\ \emptyset & \text{otherwise} \end{cases}$$

Now, if we note $Message = \{T_i(Message)\}_{i \in [1,m]}$ the fact that the message contains the tokens T_i for each attribute i, then we have:

$$Mess1 \cap Mess2 = \begin{cases} \emptyset \text{ if } \exists i \in [1,m] \text{ such that } T_i(Mess1) \cap T_i(Mess2) = \emptyset \\ \{T_i(Mess1) \cap T_i(Mess2)\}_{i \in [1,m]} \text{ otherwise} \end{cases}$$

With this operator, we can give the classifier-based version of SVI.

Let $\{PreL(A, T_i)\}$ be the list of C parts of classifiers in the model of transitions specifying action A and predicting T_i in their E part.

The current value function V_n is a list of p messages $(M_n)_{n \in [1,p]}$ with one reward value for each message. To compute V_{n+1}, we do the following:

1. For each action A
   a) For each message M_n,
      i. For each value x_i of token $T_i(M_n)$ such that $x_i \neq \#$, retrieve $\{PreL(A, x_i)\}$.
      ii. Compute the list $L(A, M_n)$ of all the non-empty intersections between each possible combination of C parts from the different $\{PreL(A, T_i(M_n))\}$, taking one message in each list per $i \in \{1, ..., m\}$.
      iii. If some messages of $L(A, M_n)$ overlap, specialize them so as to get non-overlapping messages.
      iv. For each element M' in $L(A, M_n)$, compute the contribution of M_n to the value of M'. The attributes being considered independent, the probability of reaching M_n from M' is equal to the product of the probabilities given by each classifier implied in the intersection. We note Π_{proba} this product. Thus the contribution of M_n is $contribV(M_n, M') = \Pi_{proba} V(M_n)$.
   b) If some messages M'_j and M'_k among the different $L(A, M_n), n \in [1, p]$ strictly overlap, specialize them so as to get non-overlapping messages M'' [3].

[3] If the messages defining V_n do not overlap, if the problem is markov and deterministic and if the C parts in the model of transitions do not overlap, there will not be any

c) For all M'' in all $L(A, M_n)$, sum the contributions from all states M_n computing their value according to the Bellman equation :

$$V(M'') = R(M'', A) + \beta \sum_{M_n} contribV(M_n, M'') \quad (3)$$

where $R(M'', A)$ is the immediate reward (if any) for performing action A in situations matched by M'' and β the discount factor. We obtain a unique list $L(A)$, corresponding to the quality function of the action A for any situation, according to the previous value function [4].

2. Finally, in order to compute V_{n+1}, find the maximum value for each M''' among the different $L(A)$ given by the different actions A. This implies the same merging process as in step 1c, taking the greatest value when two M''' overlap.

In step 2, we can record both the value of each situation and the action that gave rise to that value. By doing so, we record the policy giving the best action in any situation.

6 Discussion and Future Work

CbVI is very similar to SVI, apart from the fact that the basic operator in CbVI consists of an intersection between messages while SVI relies on *merging trees, appending trees* and *tree simplifications.* Though we did not measure that yet, we suspect that our version is much simpler to code, though computationally more expensive than the original SVI. Furthermore, CbVI overcomes two main drawbacks of MACS:

- it makes it possible to build a compact representation of the value function, where MACS previously had to store a value for each encountered state separately;
- as a side effect, it makes it possible to store a compact representation of the policy with very little extra cost, where MACS had to perform an expensive lookahead operation at each time step to select the best action.

Nevertheless, before incorporating CbVI into MACS, one must consider carefully the fact that MACS is a latent learning system building its own model of transitions while we made in the previous section the assumption that we had a perfect model of transitions and a perfect knowledge of the immediate reward function at hand.

To face the case where this assumption does not hold, there are four solutions:

overlapping between the $L(A, M_n)$ lists, so this step can be simplified. But in a context where we learn the model of transitions and where new sources of rewards can be discovered, it is necessary to use this more secure version.

[4] In the deterministic case to which MACS was restricted so far, Π_{proba} is always 1.0 for one particular M_n and 0 for any other, thus this sum does not need to be computed.

- First build a perfect model of transitions and of immediate reward through random or active exploration before starting to perform value iteration loops. This is not satisfactory in the context of large problems.
- Reinitialize the value function to the immediate reward classifier-based representation each time the model of transitions or of immediate reward changes.
- Try to repair locally the current value function in case of model changes.
- Consider that CbVI will be robust and do nothing special. This is more or less what previous versions of MACS were doing, and it seems to work, so it must be investigated.

The third solution seems to be the most appealing. On a more theoretical line of research, we believe that a formal functional equivalence between the SVI and CbVI can be proven, which would result in the possibility to import all the theoretical results from [BDG00] in our framework. This must be investigated soon.

7 Conclusions

As designers of MACS, we have drawn some interesting ideas about a way to make our system computationally more efficient by adapting the SVI algorithm to a classifier-based representation. But we feel that the reader can draw some more general lessons from the work presented in that paper.

Indeed, we believe that a closer communication between the LCS community and the BN community can result in a significant mutual enrichment, since both approaches come with a different perspective on the same framework, namely factored MDPs, and have developed different technical and theoretical tools to deal with that framework.

We feel that coming from one perspective to the other can raise interesting questions. For instance, since DBNs are just 2TBNs where some probabilistic dependencies between variables within the same time step (called "synchronic arcs") might hold, could such dependencies be integrated in the LCS framework without deeply reconsidering the classifier-based representation?

From a more global perspective, LCS researchers seem more focused on the learning problem than BN researchers, who are more interested in the planning/inference problem. It seems also that BN research is often more mathematically formalized than LCS research and that this formalization could be imported into the LCS framework.

Conversely, a greater knowledge of the LCS framework in the BN community would probably result in more interest in the 2TBN case, often disregarded as too simple by BN researchers. And, above all, the RL approach used in LCSs should probably be imported into the 2TBN context, since learning the structure of a problem is becoming an important research topic in the BN community.

References

[BDG95] C. Boutilier, R. Dearden, and M. Goldszmidt. Exploiting structure in policy construction. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95),* pages 1104–1111, Montreal, 1995.

[BDG00] C. Boutilier, R. Dearden, and M. Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence,* 121(1) :49–107, 2000.

[BDH99] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research,* 11:1–94, 1999.

[BGS00] M. V. Butz, D. E. Goldberg, and W. Stolzmann. Introducing a genetic generalization pressure to the Anticipatory Classifier System part I: Theoretical approach. In *Proceedings of the 2000 Genetic and Evolutionary Computation Conference (GECCO 2000),* pages 34–41, 2000.

[But02] M. V. Butz. An Algorithmic Description of ACS2. In P.-L. Lanzi, W. Stolzmann, and S. W. Wilson, editors, *Advances in Learning Classifier Systems,* volume 2321 of *LNAI,* pages 211–229. Springer-Verlag, Berlin, 2002.

[DG94] A. Darwiche and M. Goldszmit. Action networks: A framework for reasoning about action and change under uncertainty. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence (UAI-94),* pages 136–144, Seattle, WA, 1994.

[DK89] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence,* 5(3):142–150, 1989.

[FMR98] N. Friedman, K. Murphy, and S. Russell. Learning the structure of dynamic probabilistic networks. In *Proceedings of UAI 98,* 1998.

[Gha97] Z. Ghahramani. Learning dynamic bayesian networks. In C. L. Giles and M. Gori, editors, *Adaptive Processing of Temporal Information.* LNAI, Springer-Verlag, Berlin, 1997.

[GMS03] P. Gérard, J.-A. Meyer, and O. Sigaud. Combining latent learning with dynamic programming. *European Journal of Operation Research,* to appear, 2003.

[Gol89] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison Wesley, Reading MA, 1989.

[GS01] P. Gérard and O. Sigaud. YACS : Combining Anticipation and Dynamic Programming in Classifier Systems. In P.-L. Lanzi, W. Stolzmann, and S. W. Wilson, editors, *Advances in Learning Classifier Systems,* volume 1996 of *LNAI,* pages 52–69. Springer-Verlag, Berlin, 2001.

[GS03] P. Gérard and O. Sigaud. Designing efficient exploration with macs: Modules and function approximation. In *Proceedings of the Genetic and Evolutionary Computation Conference 2003 (GECCO03),* pages 1882–1893, Chicago, IL, july 2003. Springer-Verlag.

[GSS02] P. Gérard, W. Stolzmann, and O. Sigaud. YACS: a new Learning Classifier System with Anticipation. *Journal of Soft Computing : Special Issue on Learning Classifier Systems,* 6(3-4):216–228, 2002.

[HGC95] D. Heckerman, D. Geiger, and D. M. Chickering. Learning bayesian networks: The combination of knowledge and statistical data. *Machine Learning,* 20:197–243, 1995.

[How71] R. A. Howard. *Dynamic Probabilistic Systems.* Wiley, 1971.

[Lan02] P.-L. Lanzi. Learning Classifier Systems from a Reinforcement Learning Perspective. *Journal of Soft Computing,* 6(3-4): 162–170, 2002.

[LR00] P.-L. Lanzi and R. L. Riolo. A roadmap to the last decade of Learning Classifier Systems research (from 1989 to 1999). In P.-L. Lanzi, W. Stolzmann, and S. W. Wilson, editors, *Learning Classifier Systems: from Foundations to Applications,* pages 33–62. Springer-Verlag, Heidelberg, 2000.

[Pea88] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufman, San Mateo, CA, 1988.

[PS78] M. L. Puterman and M. C. Shin. Modified policy iteration algorithms for discounted Markov Decision Problems. *Management Science,* 24:1127–1137, 1978.

[RJ86] L. R. Rabiner and B. H. Juang. An introduction to hidden markov models. *IEEE ASSP Magazine,* Jan.:4–16, 1986.

[Sto98] W. Stolzmann. Anticipatory Classifier Systems. In *Genetic Programming,* pages 658–664. Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1998.

[Wil94] S. W. Wilson. ZCS, a zeroth level Classifier System. *Evolutionary Computation,* 2(1):1–18, 1994.

[Wil95] S. W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation,* 3(2):149–175, 1995.

Classifier Systems for Continuous Payoff Environments

Stewart W. Wilson

Prediction Dynamics, Concord MA 01742 USA
Department of General Engineering
The University of Illinois at Urbana-Champaign IL 61801 USA
wilson@prediction-dynamics.com

Abstract. Recognizing that many payoff functions are continuous and depend on the input state x, the classifier system architecture XCS is extended so that a classifier's prediction is a linear function of x. On a continuous nonlinear problem, the extended system, XCS-LP, exhibits high performance and low error, as well as dramatically smaller evolved populations compared with XCS. Linear predictions are seen as a new direction in the quest for powerful generalization in classifier systems.

1 Introduction

This paper extends learning classifier system (LCS) architecture (using XCS [8,3] as a basis) to environments in which the payoff is a continuous function of the input, or state, x. In most previous LCS work, the environmental payoff function $P(x, a)$ (with a the system's action) has been discontinuous. Continuous payoff functions bring new challenges but also new opportunities for classifier generalization and LCS application and permit seeing classifier systems from a broadened perspective.

The next section gives an explanation of continuous payoff environments and distinguishes them from discontinuous ones. Section 3 follows with a concrete example that will be used in experiments. Modifications of XCS for continuous payoff are given in Section 4 (see [12] for a review of XCS and [3] for an algorithmic description). Section 5 describes the experiments, followed by Section 6 with discussion.

2 Environments with Continuous Payoff Functions

2.1 Discontinuous Environments

Payoff functions for environments typically used in LCS experiments are nearly always discontinuous. Consider learning a Boolean function in which a correct action (i.e., the correct value of the function) is rewarded with 1000 and an incorrect action with 0. Very often in such functions changing a single bit of x changes the function value from, say, 1 to 0. Suppose the action a does not change. Then

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 824–835, 2004.

the reward received will change from 1000 to 0 (or vice-versa), implying that the payoff function $P(x, a)$ is locally discontinuous with respect to x. In fact, payoff functions for Boolean environments are full of such discontinuities. (They are also widely discontinuous with respect to a, but attention in this paper will focus on x.)

The discontinuity is not restricted to environments where the input is binary. Consider a data-inference problem in which exemplar attributes are real-valued and, as is typical, the correct discrete-valued decision (e.g., "malignant", "benign", "indeterminate") may change if the value of a single attribute crosses a threshold. If $P(x, a)$ assigns discrete reward values to the decisions, then $P(x, a)$ will again be discontinuous with respect to x.

These kinds of discontinuous payoff functions are well handled by the classifier syntax in a conventional LCS such *as* XCS. A classifier of XCS consists of a condition $t(x)$ (a truth function of x), an action a, and a scalar prediction p of the payoff to be expected if the system takes action a when the condition is satisfied by the current x. Collectively, the system's classifiers can evolve successfully to represent the environmental payoff function because discontinuities in $P(x, a)$ are simply handled by two classifiers, one for each side of the payoff "step". Conversely, where $P(x, a)$ is *not* discontinuous—i.e., the payoff is *the same* for several states (and a given action a)—XCS may evolve single classifiers in which $t(x)$ generalizes over those states (provided the syntax of $t(x)$ can express the generalization).

2.2 Continuous Environments

Robotic, control, and other "real world" environments such as financial time-series prediction are often characterized by payoff functions that are continuous with respect to the input x and sometimes also with respect to the action a. Simplifying somewhat, a function $P(x, a)$ is *continuous* at $(x, a) = (x_0, a_0)$ if $\lim_{x \to x_0, a \to a_0} P(x, a)$ exists and $\lim_{x \to x_0, a \to a_0} P(x, a) = P(x_0, a_0)$ [1]. If $P(x, a)$ is continuous at all (x, a) of interest, then we will call it a continuous function. The intuition is that in a continuous function, small changes in the input result in small changes in the value of the function.

Sometimes $P(x, a)$ will only be continuous with respect to x, for example if a system is capable of just a finite set of discrete actions. Or, in principle, the function might only be continuous with respect to a. In the following, we shall consider the case where $P(x, a)$ has the form $P(x, a_j)$, where a_j is one of a finite set of discrete actions, and $P(x, a_j)$ is continuous with respect to x at every x of interest.

As a general framework, consider a robot system for which the environmental state is represented by a vector x of real-valued sensor readings and the robot can take any of a finite set of discrete motor actions such as "turn left", "take one step forward", etc. Given an input state x and an action a_j, the resulting state y may depend in a continuous way on x. That is, a slightly different x would, given a_j, result in a slightly different y. If in turn the payoff p to the system depends continuously on y, then in effect the robot would be acting in an environment

where $P(x, a_j)$ is continuous with respect to x. A classifier system designed to optimize the robot's movements in this environment would need to learn to predict $P(x, a_j)$. As will be seen in the following example environment and experiments with it, the traditional LCS architecture with its scalar predictions is inefficient and lacks transparency in learning continuous payoff functions, but these problems are greatly alleviated with a modified architecture. (For neural LCS approach to continuous payoff, see [2].)

3 Example: A 'Frog' Problem

Consider a frog-like system that learns the best-sized jump to catch a fly. The "frog" receives sensory input that is related to the fly's distance, jumps a certain distance in the fly's direction, and gets payoff that is based on the remaining distance. We shall assume that the frog has a finite set of discrete actions—in this case, jumps of certain lengths. For any fly distance (in the range allowed) the frog should learn to choose the action that lands it closest to the fly.

Let d be the frog's distance from the fly, with $0.0 \leq d \leq 1.0$. What should we take for x, the sensory input? Any quantity that monotonically decreases with d would be reasonable. For the moment we will take the simplest, a linear decrease: $x(d) = 1 - d$. For actions a_j, we will assume $k \geq 2$ equally-spaced jump lengths: $0.0, 1/(k-1), 2/(k-1), ..., (k-2)/(k-1), 1.0$.

The payoff should be any function of x and a that is bigger the smaller the distance d' that remains after jumping. That is, $P(x, a)$ should monotonically increase with smaller d'. One quite natural choice is to let the payoff equal the sensory input *following* the jump, as though the frog is rewarding itself based on what it "sees". Then, with the sensory function above, $P = 1 - d'$.

To write the payoff in terms of x and a, we need to make one assumption. Suppose the frog's jump overshoots, i.e., the frog lands *beyond* the target fly. In this case we shall assume that d' equals the amount of the overshoot (taken as a positive number). Thus $d' = d - a$ for $a \leq d$ and $d' = a - d$ for $a \geq d$. Substituting for d' in $P = 1 - d'$, then using $d = 1 - x$ and rearranging, we get

$$P(x, a) = \begin{cases} x + a & : \quad x + a \leq 1 \\ 2 - (x + a) & : \quad x + a \geq 1 \end{cases} \tag{1}$$

Payoff functions for each of the a_j discrete actions may be found by substituting a_j for a in (1). For $k = 5$ actions, the resulting functions—all continuous—are shown in Figure 1. The functions for actions 0.0 (no jump) and 1.0 are simple straight lines, but the functions for the three intermediate actions are nonlinear, forming "tents" that peak at $a_j = 1 - x$ (i.e. where the action equals the distance to the fly). For any x, the optimal action is the one corresponding to the function that is largest at that x. A classifier system would solve the frog problem by learning the functions and then, given an x, pick the action whose function is greatest there.

Because they make scalar predictions, traditional classifier systems are inefficient learners of continuous payoff functions. The reason is that the scalar

predictions amount to a piecewise-constant approximation. This is generally the least efficient approximation technique for continuous functions, in the sense that large numbers of classifiers are required in order to meet a given error criterion. [10,12] demonstrated a classifier-system-like technique that learned continuous functions using piecewise-linear approximations, thereby exploiting the latter's far greater efficiency. In this paper we employ that technique in a standard multiple-action LCS architecture, where it is used to learn the $P(x, a_j)$.

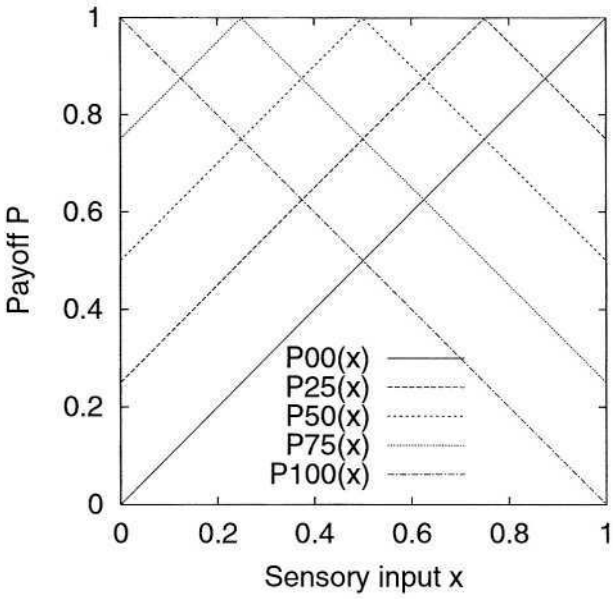


Fig. 1. Frog problem payoff functions $P(x, a_j)$ for $a_j \in \{0.00, 0.25, 0.50, 0.75, 1.00\}$. (Functions denoted by "P00" for $P(x, 0.00)$, etc.)

4 XCS-LP: XCS Modified for Linear Predictions

To address the frog problem and, generally, to make an LCS capable of predicting efficiently in continuous payoff environments, XCS was modified in two respects. The first was to adapt the program for real instead of binary input vectors. The second was to change XCS's classifier architecture so that the fixed scalar prediction was replaced by a linear polynomial that would *calculate* the prediction from x. The resulting program was called XCS-LP ("linear prediction"). XCS-LP differs from XCS *only* as described in the next two subsections.

4.1 Real Inputs

The changes to XCS for real inputs were as follows [9,11]. The classifier condition was changed from a string from {0,1,#} to a concatenation of "interval predicates", $int_i = (l_i, u_i)$, where l_i ("lower") and u_i ("upper") are reals. A classifier matches an input x with attributes x_i if and only if $l_i \leq x_i \leq u_i$ for all x_i. In the experiments reported here, the input value range was [0.0,1.0].

Crossover (two-point) in XCS-LP operates in direct analogy to crossover in XCS. A crossover point can occur between any two alleles, i.e., within an interval predicate or between predicates, and also at the ends of the condition (the action is not involved in crossover). Mutation, however, is different. An allele is mutated by adding an amount $\pm rand(m_0)$, where m_0 is a fixed real, *rand* picks a real number uniform randomly from $(0.0, m_0]$, and the sign is chosen uniform randomly. If a new value of l_i is less than the minimum possible input value, in the present case 0.0, the new value is set to 0.0. If the new value is greater than u_i, it is set equal to u_i. A corresponding rule holds for mutations of u_i.

The condition of a "covering" classifier (a classifier formed when no existing classifier matches an input) has components $l_0, u_0, ..., l_n, u_n$, where each $l_i = x_i - rand_1(r_0)$, but limited by the minimum possible input value, and each $u_i = x_i + rand_1(r_0)$, limited by the maximum possible input value; $rand_1$ picks a random real number from $[0.0, r_0]$, with r_0 a fixed real.

For the subsumption deletion operations, we defined subsumption of one classifier by another to occur if every interval predicate in the first classifier's condition subsumes the corresponding predicate in the second classifier's condition. An interval predicate subsumes another one if its l_i is less than or equal to that of the other and its u_i is greater than or equal to that of the other. For purposes of action-set subsumption, a classifier is more general than another if its *generality* is greater. Generality is defined as the sum of the widths $u_i - l_i + 1$ of the interval predicates, all divided by the maximum possible value of this sum.

4.2 Linear Predictions

As mentioned, a classifier in XCS-LP calculates its prediction using a linear polynomial in x (i.e., in the components of x). [10] introduced a system, XCSF, in which the classifiers similarly used linear polynomials in x to learn, collectively, a piecewise-linear approximation to a given continuous function $f(x)$. However, XCSF had no actions (or just the "null" action) because its purpose was to use its predictions to approximate just one function. In XCS-LP, exactly the same piecewise-linear prediction technique is used, except that now there are k functions to be approximated, the $P(x, a_j)$ for each of the k actions. The following, reproduced substantially from [12], explains the prediction mechanism.

Besides its condition and action a_j, each classifier in XCS-LP has an associated *weight vector* $w = (w_0, w_1, ..., w_n)$, where n equals the number of components in x. To calculate its prediction, the classifier forms $p(x) = w \cdot x'$, where x' is x augmented by a constant x_0, i.e., $x' = (x_0, x_1, ..., x_n)$. Just as in XCS,

the prediction is only produced when the classifier matches the input. As a result, $p(x)$ in effect computes a *hyperplane approximation* to the payoff function $P(x, a_j)$ over the subspace defined by the classifier's condition. Classifiers will have different weight vectors w since in general the subspaces of their conditions differ.

Of course, the classifiers' weight vectors must be adapted. If classifiers are to predict with a given accuracy, the coefficients w_i of their weight vectors must be appropriate. Following [10] we used a modification of the *delta rule* [5]. The delta rule is given by

$$\Delta w_i = \eta(t - o)x_i \tag{2}$$

where w_i and x_i are the ith components of w and x', respectively. In the quantity $(t - o)$, o is the output, in the present case the classifier prediction, and t is the *target,* in this case the current payoff $P(x, a_j)$. Thus $(t-o)$ is the amount by which the prediction should be corrected (the negative of the classifier's instantaneous error). Finally, η is the *correction rate.* The delta rule says to change the weight proportionally to the product of the input value and the correction.

Notice that correcting the w_i in effect changes the output by

$$\Delta o = \Delta w \cdot x' = \eta(t - o)|x'|^2. \tag{3}$$

Because $|x'|^2$ is factored in, it is difficult to choose η so as to get a well-controlled overall rate of correction: η too large results in the weights fluctuating and not converging; if η is too small the convergence is unnecessarily slow. We noticed that in its original use [7], the correction rate was selected so that the entire error was corrected in one step; this was possible, however, because the input vector was binary, so its absolute value was a constant. In our problem, reliable one-step correction would be possible if a *modified delta rule* were employed:

$$\Delta w_i = (\eta/|x'|^2)(t - o)x_i. \tag{4}$$

Now the total correction would be strictly proportional to $(t - o)$ and could be reliably controlled by η. For instance, $\eta = 1.0$ would give the one-step correction of [7]. In the experiments that follow, we used the modified delta rule with $\eta = 0.2$.

Use of a delta rule requires selection of an appropriate value for x_0, the constant that augments the input vector. We found that if x_0 was too small, weight vectors would not learn the right slope, and would tend to point toward the origin—i.e. $w_0 \approx 0$. Note that x_i is a factor in the above equation for Δw_i. If x_0 is small compared with the other x_i, then adjustments of w_0 will tend to be swamped by adjustments of the other w_i, keeping w_0 small. Choosing x_0 to be about the same order of magnitude as the other x_i solved the problem in [10] and is adopted here.

To change XCS from the traditional fixed predictions to linear predictions only required appending the weight vectors to the classifiers, plus providing for calculation of the predictions and application of the modified delta rule to the weight vectors of the action set classifiers (instead of employing and updating

scalar predictions as in XCS). In a classifier created by covering, the weight vector was randomly initialized with weights from [–1.0, 1.0]; GA offspring classifiers inherited the parents' weight vectors. In [10] both policies yielded performance improvements over other initializations and are followed here.

5 Experiments

In this section we report three experiments on the frog problem. The first applies XCS-LP as described above. The second applies (standard) XCS. The third changes the sensory and payoff functions of the problem and again applies XCS-LP. The idea was first to test XCS-LP, then compare with the XCS solution, then check XCS-LP's performance when the functions were less simple than those so far defined.

In each experiment, the system alternated between learning problems and test problems. In a learning problem, d was chosen randomly from [0.0,1.0], the corresponding x was input to the system, the system chose one of the $k = 5$ actions a_j at random, and the corresponding payoff was received from the environment and used for updating parameters (the GA was also enabled). In a test problem, again beginning with a random d, the system chose the action whose system prediction was highest and the corresponding payoff was received (with no updating or GA). The probability that a problem would be a learning problem was 0.5. Each experiment consisted of 5 runs each with a different random seed. Each run began with an empty classifier population and was carried out to 300,000 learning problems to be sure results had stabilized. The results were recorded as moving averages over the previous 50 test problems of: payoff, system error, population size (in macroclassifiers), and population generality. The curves given in the figures are averages over the 5 runs.

Common parameter settings for the experiments were: population size $N = 500$, learning rate $\beta = 0.2$, error threshold $\epsilon_0 = 0.01$, fitness power $\nu = 5$, GA threshold $\theta_{GA} = 48$, crossover probability $\chi = 0.8$, mutation probability $\mu = 0.04$, deletion threshold $\theta_{del} = 50$, fitness fraction for accelerated deletion $\delta = 0.1$. Also, mutation increment $m_0 = 0.1$ and covering interval $r_0 = 0.1$. GA subsumption was enabled, with $\theta_{GAsub} = 100$. For XCS-LP, $\eta = 0.2$ and $x_0 = 1.0$.

5.1 XCS-LP on the Frog Problem

Results for XCS-LP are shown in Figure 2. From (1) and Figure 1 we can calculate that if the system chooses the best action every time it should receive an average payoff of 0.9375. The payoff curve appears to reach such a value, as was confirmed directly by the data. Thus on the time scale of the experiment, the system almost immediately learned to pick the best action. The system error also quite quickly went to a low value, which from the data was approximately 0.004 by the end of the experiment, actually less than the experiment's error threshold, $\epsilon_0 = 0.01$. During the first 50,000 problems the population size and

generality moved to values that remained fairly steady subsequently. By the end of the experiment, the average population size was 24.6 classifiers.

Figure 3 gives insight into the evolved population. Shown are the classifiers of a typical population, at 300,000 learning problems. A special graphic notation is used to represent the conditions. The range of x is divided up into subranges of length 0.1. A "." appearing in a subrange means that the condition's interval predicate will accept no x value in that subrange. "0" means that every x will be accepted, while "o" means some will be accepted because one or both boundaries of the predicate fall somewhere in that subrange. The condition of classifier 0 accepts every x—it is completely general. Classifier 1 is almost so—in fact the value of u is 0.999. The actual ranges for classifiers 2 and 3 are (0.474, 1.0) and (0.0,0.527), and so forth.

The first 13 classifiers have high numerosity and fitness compared with the remainder. Because they dominate the calculation of the system prediction, they can be regarded as the system's solution to the task. Inspection shows that the conditions of the classifiers quite accurately correspond to the x ranges of the straight-line segments of $P(x, a_j)$ for each action. Furthermore, the weight vectors have slopes and intercepts that also reflect the $P(x, a_j)$. For instance, classifiers 2 and 3 reflect $P(x, 0.5)$ in that they approximately equally divide the x range and their weight-vector intercepts (first components) and slopes are close to those of the ideal weight vectors which are (1.5, -1.0) and (0.5, 1.0). Similarly for the classifiers corresponding to actions 0.0, 0.75, and 1.0. However the upper straight-line segment for $P(x, 0.25)$ seems to be weakly defined (classifiers 4, 7, and 10) in that the slope is around -0.68 whereas it should ideally be -1.0.

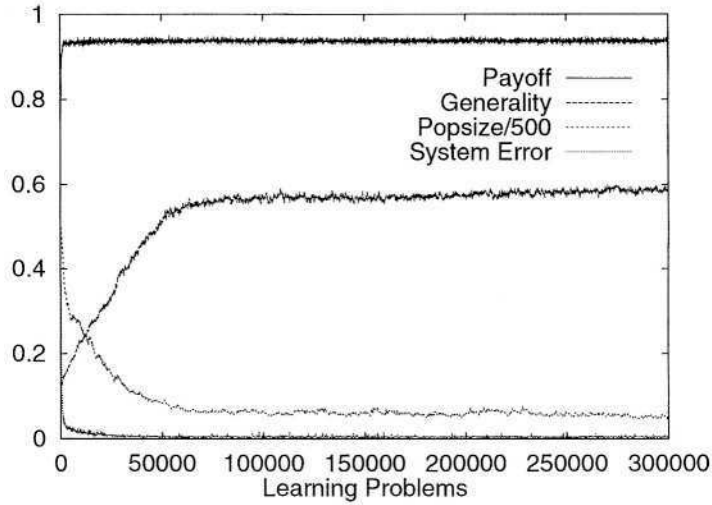


Fig. 2. Results for XCS-LP on frog problem (legend shows curve order at right).

	CONDITION	ACT	WTVECTOR	ERR	FITN	NUM	EXPER
0.	|0000000000|	1.00	1.00 -1.00	0.000	.997	76	28224
1.	|000000000o|	.00	0.00 1.00	0.000	.970	67	8274
2.	|....o00000|	.50	1.49 -0.98	0.003	.916	56	7463
3.	|00000o....|	.50	0.50 0.98	0.003	.769	48	3057
4.	|........o0|	.25	1.46 -0.69	0.010	.902	40	2584
5.	|0000000o..|	.25	0.25 0.99	0.007	.567	38	21004
6.	|00o.......|	.75	0.76 0.95	0.003	.942	38	1450
7.	|........oo|	.25	1.47 -0.68	0.013	.260	28	2999
8.	|..o0000000|	.75	1.23 -0.98	0.005	.555	26	5231
9.	|0000000o..|	.25	0.25 0.99	0.007	.364	23	801
10.	|.......oo.|	.25	1.50 -0.68	0.007	.478	21	880
11.	|..o0000000|	.75	1.23 -0.97	0.013	.044	20	1088
12.	|..o0000000|	.75	1.25 -1.00	0.000	.335	11	3776
13.	|.........o|	1.00	-0.99 1.01	0.013	.000	1	6
14.	|.........o|	.25	1.36 -0.65	0.198	.000	1	12
15.	|.........o|	.50	1.23 -0.78	0.180	.000	1	10
16.	|........o0|	.25	1.47 -0.69	0.014	.001	1	39
17.	|00000000o.|	.25	0.25 0.99	0.007	.017	1	133
18.	|.........o|	.75	-0.33 -0.34	1.658	.001	1	4
19.	|.........o|	.00	-1.00 -1.00	0.005	.002	1	0
20.	|00000o....|	.50	0.50 0.98	0.013	.008	1	22

Fig. 3. Entire population from one run of the experiment of Figure 2 ordered by decreasing numerosity. (ACTion, WeighTVECTOR, ERRor, FITNess, NUMerosity, EXPERience.)

5.2 XCS on the Frog Problem

Figure 4 shows frog problem results using standard XCS. Compared with XCS-LP, the payoff curve is essentially the same, reaching its steady value slightly quicker. The system error is significantly higher, approximately 0.03 vs. 0.004. The population size is much higher, averaging 192 classifiers at the end vs. 24.6, and does not decrease significantly over time. Generality is about 0.11 vs. 0.59 and also does not change over the experiment.

Figure 5 suggests the reason for these changed results. Shown are the 11 classifiers at the top of the numerosity ranking at the end of a typical run. They are very specific (interval predicates are small) as were the other 170 classifiers in this population. If all classifiers are quite specific, average population generality will be low, and the population size must be high in order to cover all inputs.

Thus XCS was capable of learning the frog problem, but at the cost of a large population of highly specific classifiers. In this problem the payoff function varied significantly with small changes in x. The scalar prediction of an XCS classifier can only stay accurate over a short interval, so that large numbers of them were required. Moreover, in this experiment the system error substantially exceeded ϵ_0, suggesting that the population was actually too small for good coverage. To check this we re-ran the experiment with $N = 2000$. Then the average system error fell to about 0.01, equaling ϵ_0. However, the classifiers were even more specific than before and the population size was 478.

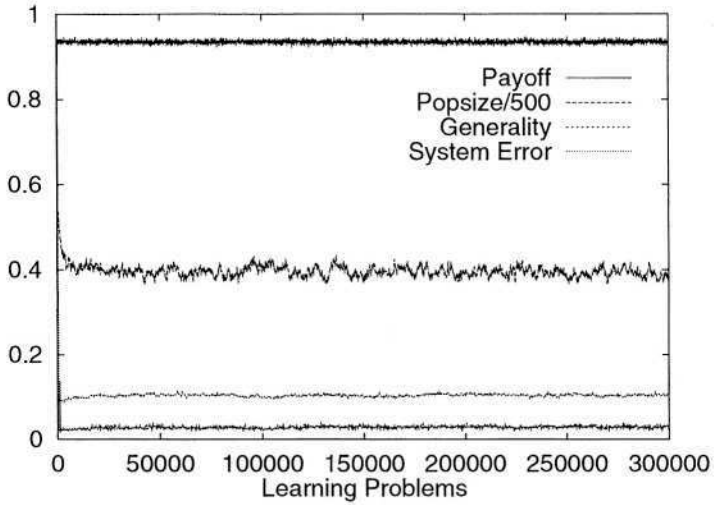


Fig. 4. Results for XCS on frog problem.

	CONDITION	ACT	PRED	ERR	FITN	NUM	EXPER
0.	|o.....|	1.00	.948	.011	.601	11	167
1.	|oo.....|	.50	.608	.020	.252	8	232
2.	|..o.....|	.00	.238	.008	.551	7	46
3.	|.....o0|	.75	.332	.022	.148	7	83
4.	|.....o...|	.75	.587	.008	.461	7	23
5.	|.....oo|	.25	.845	.026	.208	7	37
6.	|....oo....|	.50	.982	.013	.286	7	39
7.	|.....o0o.|	.25	.940	.023	.224	7	46
8.	|o000o.....|	.75	.911	.036	.424	7	82
9.	|..oo.....|	.00	.319	.018	.238	7	15
10.	|..o0o.....|	.25	.552	.035	.385	7	11

Fig. 5. Top 11 of 181 classifiers from one run of experiment of Figure 4. (PRED = scalar prediction)

5.3 XCS-LP on a Modified Frog Problem

The third experiment returned to XCS-LP but changed the sensory function to $x(d) = e^{-d}$. This changed the payoff function to

$$P(x,a) = \begin{cases} xe^{a} & : \quad a \leq -ln(x) \\ (1/x)e^{-a} & : \quad a \geq -ln(x) \end{cases} \tag{5}$$

Now the function—still continuous—is both nonlinear in having a tent-like structure and nonlinear in half of the tent "sides". However, the results shown in Figure 6 are similar to those of Figure 2. The evolution is slower and the final population somewhat larger (57.8). This is apparently due to the curvature of

the tent sides, which requires additional classifiers to approximate accurately, as can be seen in listings of the population (omitted here for lack of space).

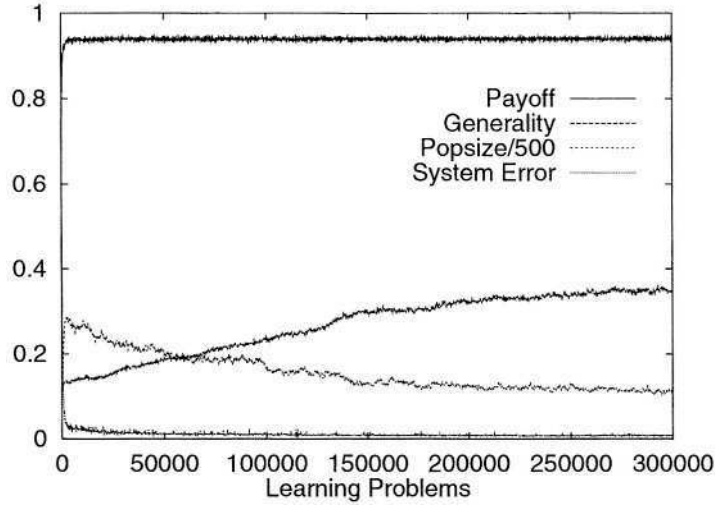


Fig. 6. Results for XCS-LP on modified frog problem.

6 Discussion

The experiments demonstrated a feasible classifier system technique for learning in environments where the payoff for a given action is a continuous and nonlinear function of the state x. In addition, the first and second experiments showed that, compared with traditional scalar predictions, *calculating* the predictions via a linear approximation to the payoff function can yield a substantial reduction in population size while increasing the transparency of the system's knowledge.

Further work is needed, in a variety of environments, to increase understanding of the technique. In particular, the evolution is quite slow, and there may be other methods of updating the weight vectors that make it faster. The approximation technique needs also to be extended to cover the action variable(s), since in many environments (including the frog problem) the payoff is a continuous function of the action (Reynolds's "Hoverbeam" task [6] is an interesting further example). Moreover, besides linear functions, other approximation bases should be looked at.

In dealing with continuous payoff, XCS-LP advances the ability of classifier systems to *generalize,* that is, to evolve compact and at the same time readable representations of complex payoff functions. Traditionally, effort has focused on making classifier condition syntax more expressive. The present research points out a separate path to compact representation via viewing the prediction as a function of the input and then approximating it.

References

1. mathworld.wolfram.com/ContinuousFunction.html.
2. Larry Bull and Toby O'Hara. Accuracy-based neuro and neuro-fuzzy classifier systems. In W. B. Langdon et al, editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference,* pages 905–911. Morgan Kaufmann, 2002.
3. Martin V. Butz and Stewart W. Wilson. An Algorithmic Description of XCS. In Lanzi et al. [4], pages 253–272.
4. Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors. *Advances in Learning Classifier Systems,* volume 1996 of *LNAI.* Springer-Verlag, Berlin, 2001.
5. Tom M. Mitchell. *Machine Learning.* WCB/McGraw Hill, Boston, MA, 1997.
6. Stuart I. Reynolds. A description of state dynamics and experiment parameters for the hoverbeam task. Technical report, University of Birmingham, School of Computer Science, 2000.
7. Bernard Widrow and Marcian E. Hoff. Adaptive switching circuits. In James A. Anderson and Edward Rosenfeld, editors, *Neurocomputing: Foundations of Research,* pages 126–134. The MIT Press, Cambridge, MA, 1988.
8. Stewart W. Wilson. Classifier Fitness Based on Accuracy. *Evolutionary Computation,* 3(2): 149–175, 1995.
9. Stewart W. Wilson. Get Real! XCS with Continuous-Valued Inputs. In Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors, *Learning Classifier Systems. From Foundations to Applications,* volume 1813 of *LNAI,* pages 209–219, Springer-Verlag, Berlin, 2000.
10. Stewart W. Wilson. Function approximation with a classifier system. In Lee Spector et al, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001),* pages 974–981. Morgan Kaufmann, 2001.
11. Stewart W. Wilson. Mining Oblique Data with XCS. In Lanzi et al. [4], pages 158–174.
12. Stewart W. Wilson. Classifiers that approximate functions. *Natural Computing,* 1(2-3):211–233, 2002.

Confidence and Support Classification Using Genetically Programmed Neural Logic Networks

Henry Wai-Kit Chia and Chew-Lim Tan

School of Computing,
National University of Singapore,
3 Science Drive 2, Singapore 117543
{hchia, tancl}@comp.nus.edu.sg

Abstract. Typical learning classifier systems employ conjunctive logic rules for representing domain knowledge. The classifier XCS is an extension of LCS with the ability to learn boolean logic functions for data mining. However, most data mining problems cannot be expressed simply with boolean logic. Neural Logic Network *(Neulonet)* learning is a technique that emulates the complex human reasoning processes through the use of *net rules.* Each *neulonet* is analogous to a learning classifier that is rewarded using support and confidence measures which are often used in association-based classification. Empirical results shows promise in terms of generalization ability and the comprehensibility of rules.

1 *Neulonet* Association Rules as Learning Classifiers

Current research in Neural Logic Network *(Neulonet)* Learning using genetic programming has been shown to be an effective system for data mining [1]. Its novelty lies in its ability to emulate human decision logic through the use of rudimentary neural nets, or *net rules,* that represent the core human logic operations. These richer logic operations can be used to represent common human decision making processes such as a *priority* operation that depicts the notion of assigning varying degrees of bias to different decision factors, or a *majority* operation that involves some strategy of vote-counting. *Net rules* supplement the boolean logic operations of conjunction, disjunction and negation, so as to allow for more elegant expressions of complex logic in any problem domain.

Each *neulonet* represents a condition-action rule in much the same way as a learning classifier in LCS [2] and its corresponding extension to data mining using XCS [3]. The condition part of the classifier in LCS is a string in the alphabet {0,1,#} while inputs to a *neulonet* are ordered pairs (0, 1), (1, 0), (0, 0) representing false, true and unknown. Other than the use of a more expressive set of logic operators, each *neulonet* is evolved using genetic programming [4] based on fitness proportionate selection, as opposed to the use of genetic algorithm in typical LCS. The novelty in this present work lies in using a fitness criterion which accounts for the confidence and support levels that are extensively used in association-based classifiers such as CBA [5] where conjunctive logic rules termed CARs are discovered. The *confidence* of a classifier provides a measure on its accuracy, while *support* gives an indication of the amount of data

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 836–837, 2004.

Table 1. Experimental results depicting predictive errors (%) using NARs and CARs. Numbers inside brackets denotes the average number of rules used.

Data Set	CARs	NARs	Data Set	CARs	NARs
anneal	3.6 (34)	**0.3** (18.0)	horse	18.7 (97)	**14.2** (38.4)
auto	27.2 (54)	**16.9** (34.2)	hypo	1.7 (35)	**1.2** (44.3)
breast-w	4.2 (49)	**3.3** (40.7)	iono	8.2 (45)	**6.5** (21.4)
cleve	16.7 (78)	**16.2** (42.1)	iris	7.1 (5)	**4.0** (6.3)
crx	14.1 (142)	**13.3** (82.0)	labor	17.0 (12)	**6.7** (3.1)
diabetes	25.3 (57)	**23.7** (20.8)	lymph	19.6 (36)	**12.0** (21.7)
german	26.5 (172)	**24.8** (63.4)	pima	27.6 (45)	**23.8** (19.9)
glass	27.4 (27)	**21.5** (24.2)	sonar	21.7 (37)	**14.0** (21.6)
heart	18.5 (52)	**14.2** (38.5)	vehicle	31.3 (125)	**29.2** (34.7)
hepati	15.1 (23)	**13.6** (15.1)	wine	8.4 (10)	**1.1** (5.1)

consistent with it. Moreover, generality is achieved by considering parsimony of the *neulonet* structure within the fitness criterion. Such an evolved *neulonet* shall henceforth be termed a *neulonet* association rule (NAR). Another distinct difference from classical LCS, is the two-phase process adopted [5] in ordering the NARs for eventual prediction. Briefly, in the first rule-generation stage, all independent NARs are generated, while the second classifier-building stage provides a methodology to group these NARs to form an accurate predictor.

2 Empirical Study and Discussion

Empirical study is undertaken to compare the predictive ability of systems constructed with NARs as opposed to CARs in CBA. Experiment results based on ten-fold cross validation on twenty data sets from the UCI Machine Learning Repository are presented in table 1. It is evident that *neulonet* classifiers using human logic *net rules* is generally the better choice. The enhanced logic expression in NARs is the primary contributing factor towards the construction of better systems as association-based classifiers using conjunctive logic rules do not have the additional expressiveness and flexibility in handling complex logic inherent in most real-world data. Moreover, the *net rules* represent common human decision processes and therefore amenable towards human comprehension.

References

1. Chia, H.W.K., Tan, C.L.: Neural logic network learning using genetic programming. International Journal of Computational Intelligence and Applications (IJCIA) **1** (2001) 357–368
2. Holland, J.H.: Adaptation in natural and artificial systems. University of Michigan, Ann Arbor. Republished by MIT Press, Cambridge, MA, USA (1975, 1992)
3. Wilson, S.W.: Mining oblique data with XCS. Lecture Notes in Computer Science **1996** (2001) 158–174
4. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, Mass. (1992)
5. Liu, B., Hsu, W., Ma, Y.: Integrating classification and association rule mining. In: Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining. (1998) 80–86

An Evolutionary Constraint Satisfaction Solution for Over the Cell Channel Routing

Adnan Acan and Ahmet Unveren

Computer Engineering Dept., Eastern Mediterranean University, Gazimağusa, T.R.N.C. Mersin 10, TURKEY
{adnan.acan, ahmet.unveren}@emu.edu.tr

Abstract. A novel combination of genetic algorithms and constraint satisfaction modelling for the solution of two and multi-layer over-the-cell channel routing problems is presented. The two major objectives of the optimization task are to find an optimal assignment of nets to over-the-cell and within the channel tracks, and to minimize the channel widths through a simple but effective iterative routing methodology. Two genetic algorithms cooperate in a nested manner to perform the optimization task. The results obtained using the benchmark problems published in literature indicate that, without any predefined fixed upper/lower channel widths, the implemented algorithm outperforms well-known channel routers.

1 Introduction

Well known conventional channel routers are developed to find a least-width channel sufficient to realize interconnection requirements in a rectangular region which carries fixed terminals along its two opposite sites (See Figure 1). In order to reduce the channel width below the channel density, some channel routers use the extra routing area over the cells, in addition to the channel area, for interconnections. Routers employing this approach are called over the cell (OTC) channel routers. In most cases, OTC channel routers can complete the interconnections using fewer tracks than the channel density. Since a large portion of the area of a VLSI circuit is used for wire routing, savings in channel area obtained by using OTC routers are usually significant. In fact, this is why OTC channel routers became attractive in research and practical applications [1], [2], [3], [4], [5].

In this paper, a novel OTC channel routing algorithm is presented. Two genetic algorithms (GAs) work together in a nested scheme in such a way that the outer GA tries to find an optimal assignment of nets to the available routing areas while the inner GA computes the fitness value of each generated assignment. Specific advantages of the proposed algorithm over the other well-known OTC channel routers are its simpler problem representation, more powerful optimization capability, low computational complexity, and consequently its low execution time. The presented algorithm can be used for any multi-layer OTC channel routing. On the one hand, it is an up-to-date routing method, which can

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 838–849, 2004.

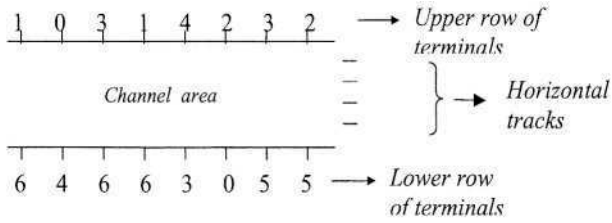


Fig. 1. A standard channel routing problem.

be implemented in the most recent cell-based design technologies for integrated circuits. On the other hand, the algorithm simultaneously optimizes the widths of over-the-cell and within the channel areas. That is, unlike the other OTC channel routers, which predefine and fix the number of tracks in over-the-cell areas, the implemented algorithm finds the widths of all channel areas through an evolutionary optimization procedure.

This paper is organized as follows. In Section 2, the basic definitions of constraint satisfaction modelling are introduced. In Section 3, constraint satisfaction modelling as applied to OTC channel routing problems is described in detail. The proposed nested genetic algorithms approach for two- and multi-layer OTC channel routing are presented with all implementation details in Section 4. The experimental results for two-layer benchmark problem instances are given in Section 5. Section 6 covers the results for multi-layer benchmark problem instances and related discussions. Conclusions and future research directions are given in Section 7.

2 The Constraint Satisfaction Modeling

Constraint satisfaction problems (CSPs) provide a general representation and solution model to a large class of combinational and other discrete structural problems. Those problems that can be represented as a CSP and solved with constraint satisfaction search techniques are those that involve one or more sets of entities (real world objects, variables, or concepts), a set of constraints (restrictions) on how entities are related to each other, and the range of values that each entity can take within a predefined domain [6].

The definition of a CSP includes the following features: a finite set of variables $V = \{V_1, V_2, V_3, \cdots, V_N\}$, finite sets of values, $D_i, i = 1, \cdots, N$, corresponding to domains of variables, and finite sets of constraints, $C_i, i = 1, \cdots, N$, that must be satisfied by any assignment of values to variables. Each variable can be assigned any of the values in its associated domain. Each constraint in the problem puts a restriction on the values that a group of variables can take in combination.

An assignment of a value to a variable is known as a binding or instantiation. The binding of a variable v to a value u is denoted as u/v. If the variable V_1 has a domain $\{u_1, u_2, u_3\}$, then its possible bindings are $u_1/V_1, u_2/V_1 and u_3/V_1$.

A constraint that can be explicitly enumerated is commonly represented by a set of possible legal substitutions. For example, a constraint C for variables V_1 and V_2 may be denoted by $C(V_1, V_2) = \{(u_1, u_2), (u_3, u_4)\}$. For V_1 and V_2 to have legal bindings according to this constraint, either u_1/V_1 and u_2/V_2 or u_3/V_1 and u_4/V_2 should be followed.

In order to find a solution for a CSP all variable assignments in the set of solutions must not violate any constraints. A solution is found only if a legal assignment can be made to all variables of the CSP.

3 A CSP Model for Multilayer OTC Channel Routing

A channel routing problem is traditionally described by a netlist. A predefined number of wiring layers are available for the routing of horizontal and vertical net segments. Layers dedicated to horizontally oriented wire segments are called horizontal layers (H-layers) and layers dedicated to vertically oriented wire segments are called vertical layers (V-layers). The two-layer routing of the problem presented in Figure 1 is given in Figure 2.

In order to represent physical constraints among net segments, graphs of vertical and horizontal constraints, VCG and HCG respectively, are used (See Figure 3). In this study, an OTC channel routing problem is defined as a constraint-satisfaction problem where the variables are nets to be routed. Constraints on net tracks (values to be assigned to nets) are extracted from VCG and HCG and are automatically transformed into a set of inequalities of net tracks. According to the described constraint satisfaction modelling, the representation of the problem in Figure 1 is illustrated in Table 1. A set D_i in Table 1 holds the possible routing tracks for *neti,* which are determined by using the constraints available in set C_i. Constraints in C_i are extracted from VCG and HCG.

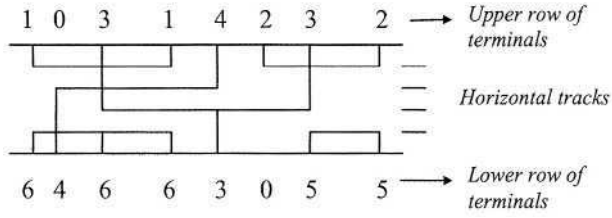


Fig. 2. A valid solution to the standard channel routing problem given in Figure 1.

For multi-layer channel routing, a router uses several H- and V-layers for the routing of horizontal and vertical net segments with no physical constraint

Table 1. Constraint function modelling of the problem given in Figure 1

$N = \{N_1, N_2, N_3, N_4, N_5, N_6\}$	
$D_1 = \{1, 2, 3\}$	$C_1 = \{d_1 < d_6, d_1 \neq d_3, d_1 \neq d_4\}$
$D_2 = \{1, 2, 3\}$	$C_2 = \{d_2 < d_5, d_2 \neq d_3\}$
$D_3 = \{2, 3\}$	$C_3 = \{d_3 > d_4, d_3 < d_5, d_3 < d_6, d_3 \neq d_1, d_3 \neq d_2\}$
$D_4 = \{1, 2, 3\}$	$C_4 = \{d_4 < d_3, d_4 < d_5, d_4 < d_6, d_4 \neq d_1\}$
$D_5 = \{3, 4\}$	$C_5 = \{d_5 > d_2, d_5 > d_3, d_5 > d_4\}$
$D_6 = \{3, 4\}$	$C_6 = \{d_6 > d_1, d_6 > d_3, d_6 > d_4\}$

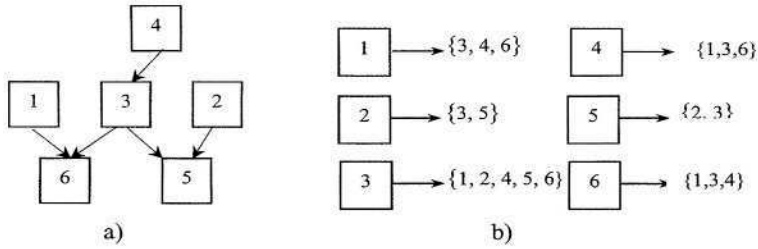


Fig. 3. a) Vertical constraint graph (VCG) of the problem in Figure 1, b) Horizontal constraint graph (HCG) of the problem in Figure 1.

violations. In this case, H-layers and V-layers are usually grouped in a way that the placement of each net's vertical and horizontal wire segments is simple and efficient. For this purpose, VH and HVH models are conventionally used for grouping the layers. The VH model allows using one horizontal layer and one vertical layer for routing, whereas the HVH model allows two horizontal layers and one in between vertical layer for routing. The presented algorithm makes use of a predefined number of H- and V-layers for OTC and within the channel areas as follows; let L be the total number of layers, for the cases $L = 2$ or L = 3, trivially the HV and HVH models are used for over the cell and within the channel areas respectively. For cases $L \geq 4$, over the cell routing uses one V-layer and $(L - 1)$ H-layers. As there is no vertical constraint available among nets in over the cell area, one V-layer is enough to route the nets assigned to this area. For example, HVH and HHVHH models are used for three- and five-layer routing for over the cell area respectively. For the cases $L \geq 4$, the presented algorithm uses one of the following cases to route the nets within the channel area [7], [8], [9], [10]:

- Case 1 ($Lmod3 = 0$ and $L \geq 4$):
  The case has $2L/3$ H-layers and $L/3$ V-layers and the layers are grouped as HVH, HVH, … , HVH.

- Case 2 ($Lmod3 = 1$ and $L \geq 4$):
  The case has $(2(L/3 - 1) + 2)$ H-layers and $((L/3 - 1) + 2)$ V-layers and the layers are grouped as, HVH, HVH, … , HVH, VH,VH.
- Case 3 ($Lmod3 = 2$ and $L \geq 4$):
  The case has $((2L/3) + 1)$ H-layers and $(L/3 + 1)$ V-layers and the layers are grouped as, HVH, HVH, … , HVH, VH.

4 Channel Width Optimization

The developed algorithm aims to find an optimal assignment of nets to upper, lower, and within the channel areas such that the number of upper, lower and within the channel tracks, hence the three channel widths, are minimized. The method employs a novel evolutionary optimization approach where two GAs cooperate in a nested manner [11], [12], [13], [14]. The outer GA tries to find an optimal assignment of nets to three available channel areas over a population of different assignments, and the inner GA, called by the outer GA for each net assignment, tries to minimize the channel width by optimally routing the area-assigned nets. In this way, the inner GA's results are used as the fitness values of the outer GA's individuals. Layer assignment of nets over the available multiple layers in over the cell and within the channel areas are made by outer and inner GAs. The algorithmic framework of the proposed approach is illustrated below:

1. Initialize *Outer_Population*
2. For $i = 1$ *to* $|Outer\_Population|$
3. Fitness(*Outer_Population*(i))=Inner_GA(*Outer_Population*(i))
4. DONE=FALSE
5. While (NOT DONE)
6. Mating_Pool=Selection(*Outer_Population)*
7. *New_Outer_Population*=Crossover(Mating_Pool,P_c)
8. Mutation(*New_Outer_Population*,P_m)
9. For $i = 1$ *to* $|New\_Outer\_Population|$
10. Fitness($New\_Outer\_Population(i)$)=Inner_GA($New\_Outer\_Population(i)$)
11. *Outer_Population=New_Outer_Population*

Chromosomes of individuals in the outer GA are vectors where each vector element holds a double containing two integer numbers. The two integer numbers determine the state and OTC layer of the corresponding net; the first integer identifies the routing state, either OTC or within the channel, while the second determines the routing layer. For the first integer number; '0' represents a net to be routed in over the cell area, and a '1' represents a net to be routed within the channel area. For example, $(0, k)$ represents a net to be routed in over the cell area and on the $k - th$ OTC H-layer. Since only one V-layer is available in over the cell area between H-layers, there is no need to identify this V-layer within each chromosome of outer GA. If a net is to be routed within the channel area, the first integer of the corresponding double is 1, then the second vector element is ignored because the inner GA will optimally make the layer assignment of these nets. For example, (1, ?) represents a net to be routed within the channel

area whose layer assignment will be made by the inner GA during its course of optimization procedure. The question mark indicates that the layer assignment for this net is currently a don't care. For example, the chromosome representation of the routing given in Figure 4 is {(0,1), (1, ?), (1, ?), (1, ?), (0,1), (0,1)}.

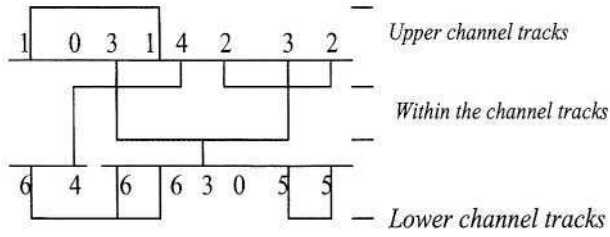


Fig. 4. Over the cell channel routing of the problem in Figure 1.

The initial population of the outer GA is easily formed by generating randomly initialized vectors of length N, where the vector elements are integer-valued doubles representing different area and layer assignments for each net. For OTC routing, the nets to be routed in OTC areas are identified from the first values of chromosome elements and, then, placed onto the appropriate tracks by checking their horizontal constraints within their H-layers, starting from the first track.

Routing within the channel area is critical in overall reduction of the channel width and it is carried out optimally by the inner GA. For each chromosome of the outer GA, first those nets to be routed within the channel area are determined. Then, chromosomes of the inner GA are randomly initialized as fixed-length vectors where each vector element is an integer-valued double; the first integer determines the horizontal routing layer (H-layer) whereas the second one corresponds to the horizontal routing track on this layer. V-layer assignments of the net segments are done according to the following cases automatically by using the H-layer numbers. Let h be the H-layer number of a net, H and V be the total number of H- and V-layers available, respectively, and L be the total number of layers.

- Case 1 ($Lmod3 = 0$):
  If $(hmod2 = 0)$, current net is assigned with $(h/2) - th$ V-layer.
  If $(hmod2 = 1)$, current net is assigned with $h/2 - th$ V-layer.

- Case 2 ($Lmod3 = 1$):
  For all $(h < H - 1)$, Case 1 is used for V-layer assignment.
  If $(h = H - 1)$, current net is assigned with $(V - 1) - th$ V-layer.
  If $(h = H)$, current net is assigned with $V - th$ V-layer.

– Case 3 ($Lmod3 = 2$):
For all ($h < H$), Case 1 is used for V-layer assignment.
If ($h = H$), current net is assigned with $V - th$ V-layer.

An important feature of inner GA is that all optimizations are carried out over a population of feasible potential solutions. For an initial population of different feasible solutions, H-layer number is randomly assigned, and track numbers are quickly obtained using a modified form of the well-known left-edge algorithm [1], as illustrated in the following algorithm, MLEA(N_w, I_w, H_w).

1. Construct_VCG(N_w).
2. Compute Max_Path_Length(VCG,N_w).
3. d=max(Max_Path_Length(VCG(N_w)).
4. Let $T = \{T_1, T_2, \cdots, T_d\}$ denote the set of routing tracks from top to bottom.
5. Selected_Nets=Select_Nets(Max_Path_Length(VCG(N_w), N_w).
6. If Selected_Nets={}
7. Goto 24.
8. Sort_Selected_Nets(Selected_Nets, I_w).
9. For $i = 1$ to |Selected_Nets|
10. SUCCESSFUL=FALSE.
11. For $j = 1$ to d {
12. H_Track(Selected_Nets(i))=j.
13. If (Check_Constrainst(j, C(Selected_Nets(i), H_w) == TRUE)
14. SUCCESSFUL=TRUE.
15. }
16. If SUCCESSFUL==TRUE
17. N_w=N_w {Selected_Nets(i).
18. Else
19. $d = d + 1$.
20. H_Track(Selected_Nets(i))=d.
21. }
22. Construct_VCG(N_w).
23. Goto 5.
24. End.

For example, let $L = 5$ and $(3, 2)$ represents a net to be routed on third H-layer within the second track, then by the above cases it must use second V-layer. The inputs for the modified left-edge algorithm are a set of nets to be routed within the channel area $N_w = N_1, N_2, \cdots, N_m$, a set of intervals $I_w = I_1, I_2, \cdots, I_m$ corresponding to leftmost and rightmost terminal positions, and H-layer numbers of nets $H_w = H_1, H_2, \cdots, H_m$. First, this algorithm selects all the nets which are on the maximum length path of the updated VCG. Next, the selected nets are sorted in ascending order of their leftmost terminal positions. Then, they are assigned horizontal tracks, in order, within their H-layers without violating the constraints defined in problem representation. If a vertical

constraints violation cannot be solved, additional tracks are used to complete the routing. After the selected nets are assigned to horizontal tracks, they are removed from the set Nw and the VCG is updated accordingly. This process is repeated until all the nets assigned within the channel area are routed. In this way, a population of different feasible solutions is formed.

The reproductive phases of outer and inner GAs are quite similar except the point that inner GA moves over feasible solutions only. The outer and inner GAs are employing uniform crossover and mutation operators with probabilities P_C and P_M, respectively. The outer GA recombines two vectors without any wonder of feasibility, however the inner GA recombines two channel routings such that the offspring produced are still feasible channel routings with different, and hopefully smaller, channel widths.

The uniform crossover and mutation operators applied on the chromosomes of outer and inner GAs are illustrated below where (↓) shows the crossover and mutation points:

Uniform crossover for outer GA:

Parent 1:	↓(0,1)	(1,?)	↓(1,?)	↓(0,2)	(1,?)
Parent 2:	↓(1,?)	(1,?)	↓(0,2)	↓(1,?)	(1,?)
Offspring 1:	(1,?)	(1,?)	(1,?)	(1,?)	(1,?)
Offspring 2:	(0,1)	(1,?)	(0,2)	(0,2)	(1,?)

Uniform crossover for inner GA:

Parent 1:	↓(3,2)	(1,2)	↓(2,2)	↓(1,2)	(4,1)	(4,2)
Parent 2:	↓(2,1)	(2,1)	↓(3,1)	↓(1,1)	(4,2)	(4,1)
Offspring 1:	(2,1)	(2,1)	(2,2)	(1,1)	(4,2)	(4,1)
Offspring 2:	(3,2)	(1,2)	(3,1)	(1,2)	(4,1)	(4,2)

Mutation for outer GA:

Parent:	(0,1)	↓(1,?)	(1,?)	(0,2)	(1,?)
Offspring:	(0,2)	(0,?)	(1,?)	(1,2)	(1,?)

Mutation for inner GA:

Parent:	(3,2)	↓(1,2)	(2,1)	(1,2)	(4,1)	(4,2)
Offspring:	(3,2)	(2,2)	(2,1)	(1,2)	(4,1)	(4,2)

The fitness values of individuals in the outer GA are the results of optimizations carried out by the inner GA. The inner GA takes the CSP representation of each individual in the outer GA and finds an optimal routing for it. The fitness function used in the inner GA is :

$$f(C) = W1 * No\_of\_Tracks + W2 * Total\_Vert\_Segments \qquad (1)$$

where *No_of_Tracks* is the number of horizontal tracks used in the routing of an input area assignment, *Total_Vert_Segments* is the total length of the vertical segments of nets in the obtained solution, and W_1 and W_2 are coefficients in (0,1). This fitness function is primarily directed to the minimization of the channel width and the total length of connections, which also helps to minimize upper and lower channel areas.

5 Experimental Results for Two-Layer Routing

For the seven benchmark channel routing problems collected from literature, the presented method is compared with the algorithms in [2] and [15] in Table 2. These two algorithms, 'A Permeation Router' [2] and 'Power Driven Routing Using GA' [15], are purposely selected because they both use evolutionary solution strategies. In Table 2, the presented algorithm is named as NGAR (Nested GA Router). It can easily be seen that NGAR outperforms its counterparts for all benchmark problem instances. For example, in row 6 of Table 2, the presented method found solutions with 7 / 7 tracks for the top and bottom over-the-cell areas respectively, compared to 18 / 15 and 12/9 tracks achieved for the same areas in [2] and [15]. Inside the channel, the presented method achieved a solution using 8 tracks compared with 15 and 11. The presented method is also compared with other powerful channel routers. Figure 5 illustrates the results for all benchmark problems for the comparison of NGAR with the method given in [4] which uses a planar over-the-cell channel routing technique. NGAR performed better or as good as [4] in all trials, which demonstrate the power of he presented GA approach.

6 Experimental Results for Multi-layer Routing

For further reducing the channel width, experiments are conducted with three and five layers within the channel area and two layers in top and bottom routing regions. Figure 6 and Figure 7 present the three-layer and five-layer routing results, respectively. The proposed method achieved routings with much smaller channel widths in all benchmark problem instances. Also, the results demonstrate both the success and generality of the proposed approach for multi-layer channel routing. Figure 8 illustrate the 5-layer routings of Yk3a, which clearly demonstrates the reductions in channel widths in all routing areas.

Table 2. Comparison of the proposed algorithm with the algorithms in [2] and [15]

		[2],[15]			NGAR		
Prob.	d	T	W	B	T	W	B
Yk1	12	11, 4	6, 6	7, 7	4	4	6
Yk3a	15	13, 9	15, 8	10, 6	8	8	5
Yk3b	17	8	15	8	8	7	8
Yk3c	18	18, 12	15, 11	15, 9	7	8	7
Yk4b	17	28, 8	6, 6	24, 11	7	9	8
Yk5	20	24,10	9,9	18,7	8	9	8
Deutsch	19	21, 14	12, 13	18, 14	6	13	9

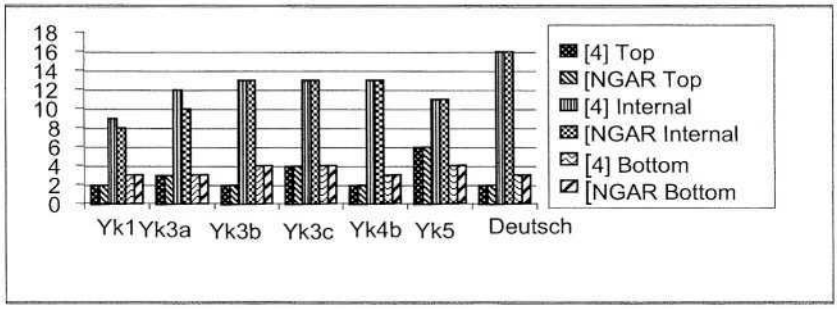


Fig. 5. Performance comparison of NGAR and [4].

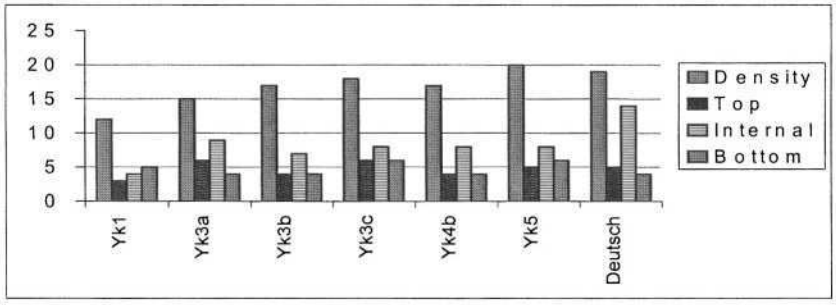


Fig. 6. Performance of NGAR in 3-layer OTC channel routing.

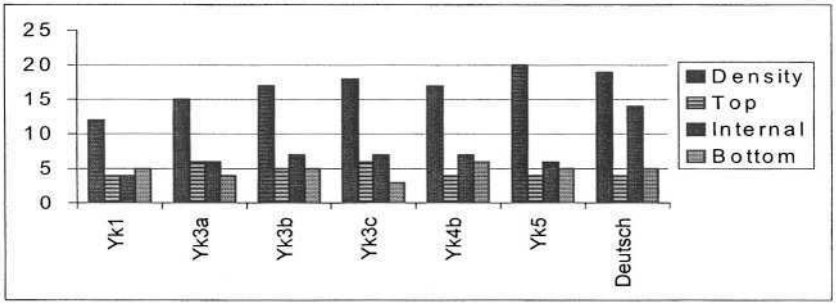


Fig. 7. Performance of NGAR for 5-layer OTC channel routing.

7 Conclusions

A constraint satisfaction modelling and a nested genetic algorithm approach is presented for the solution of well-known OTC channel routing problem. This problem is an interesting representative instance of NP-hard combinatorial opti-

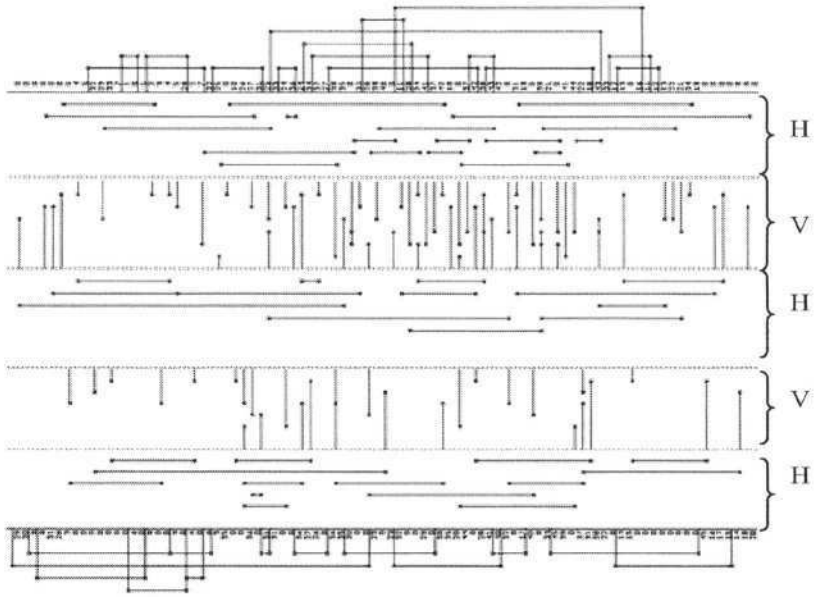


Fig. 8. Five-layer routing of Yk3a.

mization problems. The proposed algorithm combines the power of genetic evolutionary optimization with constraint satisfaction modelling to obtain a highly efficient and simple to implement method for the problem under consideration.

The novelties of the presented method can be stated in two ways. On the one hand, the proposed algorithm optimizes the assignment of nets to different routing areas using a GA which employs another GA for optimal routing of nets in their assigned areas such that the widths of routing areas are minimized. Unlike the other methods, there are no pre-specified restrictions on the widths of routing regions. On the other hand, the problem is modelled as a constraint satisfaction problem and using an evolutionary approach without passing through complicated heuristic value assignments and backtracking procedures.

The results obtained from the experiments demonstrate the success of the presented method compared to well-known OTC channel routers and its generality for the solution of two- and multi-layer problems. Simplicity of the proposed approach is another advantageous point to be stated because its implementation does not require any problem specific representations or heuristics to be used.

Further research currently undergoes in the development of evolutionary hybrid methods for the solution of well-known complex VLSI layout optimization problems.

References

1. Yoshimura, T., Kuh, A.S.: Efficient algorithms for channel routing, IEEE Trans. on Computer Aided Design of ICAS, Vol. CAD-1, pp. 25-35, (1982).
2. Shiraishi, Y.Y., Sakemi, J.: A permeation router, IEEE Transactions Computer-Aided Design, Vol. CAD-6, pp. 462-471, (1987).
3. Cong, J., Preas, B., Liu, C.L.: General models and algorithms for over-the-cell routing in standard cell design, 27th ACM/IEEE Design Automation Conference, pp.709-715, (1990).
4. Lin, M.S., Wern, P., and Lin, P.Y.L.: Channel density reduction by routing over the cells, 28th ACM/IEEE Design Automation Conference, pp.120-125, (1991).
5. Das, S., Nandy, S.C., Bhattacharya, B.B.: An Improved heuristic algorithm for over-the-cell channel routing, Proc. of ISCAS, vol 5, pp. 3106-3109, (1991).
6. Russel, R., Norvig, P.: Artificial Intelligence: A Modern Approach, Prentice-Hall, (2003).
7. Fuji, T., Mima, Y., Matsuda, T., Yoshimura, T.: A multi-layer channel router with new style of over-the-cell routing, 29th ACM/IEEE Design Automation Conference, pp. 585-588, (1992).
8. Ho, T.T., A density-based greedy router, IEEE Transactions on CAD of Integrated Circuits and Systems, Vol. 12, No. 7, pp. 973-981, (1993).
9. Madhwapathy, S., Sherwani, N., Bhingarde, S., Panyam, A.: An efficient four layer over-the-cell router, Proc. of ISCAS, pp. 187-190, (1994).
10. Madhwapathy, S., Sherwani, N., Bhingarde, S., Panyam, A.: A unified approach to multilayer over-the-cell routing, 31st ACM/IEEE Design Automation Conference, pp. 182-187, (1994).
11. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley Pub. Co., (1989).
12. Holland, J.H: Adaptation in Natural and Artificial Systems: An introductory Analysis with Applications to Biology, Control, and Artificial Intelligence, MIT Press, (1992).
13. Back, T.: Evolutionary Algorithms in Theory and Practice, Oxford University Press, (1996).
14. Miettinen, K., Neitaanmaki, P., Makela, M.M., Periaux, J.: Evolutionary Algorithms in Engineering and Computer Science, John Wiley & Sons Ltd., (1999).
15. Goni, B.M., Arslan, T., Turton, B.: Power driven routing using a genetic algorithm, 3rd World Multiconference on Systemics, Cybernetics and Informatics and 5th International Conference on Information Systems Analysis and Synthesis, Orlando, (1999).

Solution to the Fixed Airbase Problem for Autonomous URAV Site Visitation Sequencing

Amit Agarwal[1], Meng-Hiot Lim[2], Chan Yee Chew[3], Tong Kiang Poo[4], Meng Joo Er[1], and Yew Kong Leong[1]

[1] Intelligent Systems Center, Techno Plaza, Nanyang Technological University Singapore 639798
{pg0212208t, emjer}@ntu.edu.sg, leongy.dynamics@stengg.com
[2] School of Electrical & Electronic Engineering, Nanyang Technological University Singapore 639798
{emhlim, 810304085117, S7931481A}@ntu.edu.sg

Abstract. In [1], we modeled the problem of safe site visitation sequencing for an unmanned reconnaissance aerial vehicle (URAV) in graph-theoretic language. Our goal was to uncover several alternative flight paths along which the URAV can fly safely, no more than once, to all sites of interest. We offered a solution to a simpler abstraction of this problem using GAs – if we could not uncover any Hamiltonian cycles we reported maximally long paths between any two vertices. This work reports two orthogonal extensions of [1] – a. In addition to possibly reporting Hamiltonian cycles, the GA reports several alternative long paths that begin at the airbase. b. A randomized version of the gene migration scheme is used. The randomization does not appear to affect the quality of the best plan. However it raises the average quality of the best *k* plans. We give a modified gene encoding scheme and report simulation results.

1 Background

In this work, and in [1], our focus is on planning the point-to-point route of a single unmanned reconnaissance aerial vehicle (URAV) from the airbase, *A,* to possibly each nonoverlapping useful vicinity, $s_i \in S$ of site(s) of interest and back. The presence of $O(|S|)$ threats, *T,* in the reconnaissance zone is assumed. Without loss of generality, it is also assumed that these threats are evenly distributed in the region of reconnaissance interest. As discussed in [1], the structure that represents the union of minimal risk density flight paths and *S* is often a multigraph with $O(|S|)$ edges. Several graphs may be defined on this multigraph, each being optimal in a certain local sense. In [1], for the lack of a better term, any such graph was referred to as an approximated Voronoi graph *(A – Vor(G)).* However, in order to avoid contributing to the rather misleading notion that this graph is an approximation of a Voronoi graph [2, 3] in any sense, we shall call it *minimal risk density target visitation* (MRDTV) graph.

Since MRDTV graphs are often sparse, suboptimal site visitation sequencing will cause the URAV to revisit a useful vicinity in order to continue flying to other useful

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 850–858, 2004.

vicinities, strictly along minimal risk density paths. Revisitations can lead to an increase in the total time for mission completion, expose the URAV to threats for a longer duration than absolutely necessary, take away the element of surprise and may lead to decreased useful range of the URAV due to unnecessary fuel consumption. Thus the following problem, whose mathematical structure is akin to that of the NP-complete Hamiltonian cycle problem [4], gains importance –

Given an airbase, A, and a set of nonoverlapping useful vicinities of reconnaissance sites, S, find orderings $\prod_s^*$ such that a URAV can take-off from A, fly to the useful vicinities of each site, $s_i \in S$, along a subset of the edges of an MRDTV graph exactly once in the order defined by $\prod s^*_i \in \prod_s^*$ and, fly back safely to A.

If the MRDTV graph under consideration is non-Hamiltonian or, alternately, sufficient number of alternative solutions in $\prod_s^*$ are difficult to uncover then solutions that give sufficiently long acyclic paths beginning at A can still help greatly in lowering the risk to the URAV.

We use genetic algorithms to solve this problem. Even while genetic algorithms have not been particularly popular for use on TSP related problems on sparse graphs, we found them attractive since their intrinsic parallelism is naturally suited to our need for several alternative site visitation sequences. The interested reader may look at [5, 6, 7, 8, 9] for surveys or discussions on techniques for solving the Hamiltonian cycle problem.

In [1], our algorithm suffered from one major weakness: in case several different Hamiltonian cycles matching the number of plans specified by the user were not found (or, did not exist), our hybrid genetic algorithm (HGA) reported paths which though were maximally long but did not begin (or, end) in the airbase. Fig 1a shows a typical example of a long path reported. Its length is 18 (maximum possible length is 20). However, this visitation sequence plan does not begin at the airbase. And thus, its maximum useful length is only 12.

In this work we remedy this problem by changing the fitness function and making minor changes to the crossover and mutation operators [1]. Also, it was slightly more efficient to change the encoding scheme to restrict the first gene in any chromosome to corresponding to A. Fig 1b shows a typical example of the plans we now obtain – it originates at the airbase. In this example, the maximum useful length of the plan (17) and the maximal length are the same. The changes we have made to our earlier HGA do not however affect its ability to discover Hamiltonian cycles.

We had another, though minor, problem with our earlier HGA. At times, it was unable to uncover sufficient number of alternative plans that met the user's qualitative requirements even while they were known to exist. In this work, the next gene to be migrated (see Section 4) is chosen probabilistically (as opposed to sequential consideration in our former HGA) from the candidate set. This has, permitted us to obtain a greater number of alternative good plans.

The remaining paper is organized as follows. The larger focus of our work is on enabling a greater level of autonomy in remotely piloted vehicles. We briefly discuss in Section 2, the benefits of enabling intelligent onboard decision making capabilities

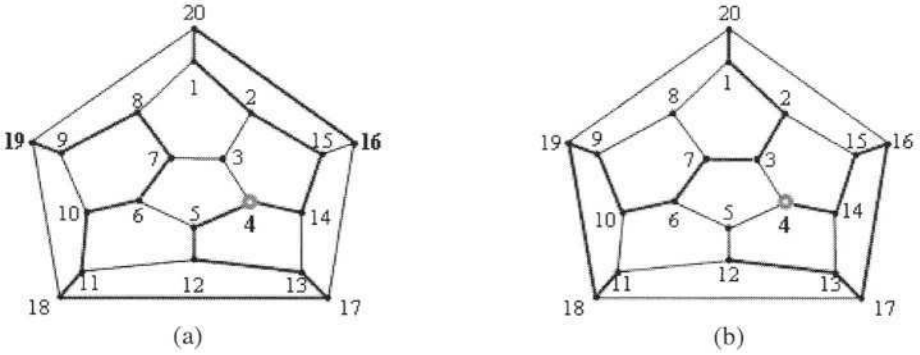


Fig. 1. An MRDTV graph with 19 useful vicinities of reconnaissance sites and an airbase (labeled *4*). Thick blue lines represent a visitation plan: (a) represents a solution typically uncovered using our earlier HGA technique in [1] and (b) represents a solution typically uncovered using the technique described in this paper

in UAVs. In Section 3 we briefly discuss how an MRDTV graph may be generated. In Section 4 we discuss our HGA. We report the results in Section 6 and conclude in the following section.

2 Unmanned to Autonomous

UAVs can be loosely classified into two categories – *a.* remotely piloted *b.* autonomous. Remotely piloted vehicles still carry out tasks such as information acquisition and transmission, takeoff and landing, monitoring of onboard systems and complex flight control from outside the loop of the remote operator. However, several other critical tasks such as terrain masking, basic flight maneuvers, ammunition or sensor selection, target and role selection, decisions regarding cooperation, information processing, waypoint path planning and visitation sequencing are carried out with the help of varying degrees of remote human assistance.

Here we briefly discuss some basic motivations for the interest in going further, from unmanned, to autonomous. Some of our arguments possibly extend to other unmanned systems as well.

Bandwidth is an important resource in a digitized operation theatre [10, 11]. Due to the need for reconnaissance information in near real-time, a URAV must transmit images or videos of objects of interest while in flight. In the former case, the demand on the downlink bandwidth can be lowered by transmitting at a lower data-rate. The later case may not permit such a convenience since the video data could quickly saturate the onboard memory buffer. However, in both cases, transmissions are necessitated only by the capture of potentially useful information and thus, are intermittent and permit the use of asynchronous time division multiplexing to improve bandwidth efficiency. This bandwidth requirement can be further lowered by only transmitting extracted information upon intelligent onboard processing.

Operator-at-the-joystick requires continuous, real-time reception of high definition video at the control station. In certain scenarios, especially those involving low flying

vehicles, operations in civilian airspace or the presence of hostile units the demand on situation awareness can be very high. This can greatly increase the remote operator's workload and require a downlink bandwidth ranging from a few tens of MHz to about 10 GHz [12]. Energy and time efficient onboard algorithms for terrain masking, obstacle recognition and avoidance and, map updating can thus greatly reduce the overall demand on the downlink bandwidth.

Recently Veverka and Campbell [13] report some experiments simulated to study the impact of complexity of operation, number of UAVs, desired speed of response and the nature and time duration of tasks on operator performance. Their results suggest that appropriate automation of vehicular activities help in increasing operator performance in situations that involve complex decision making or attention to 'several' vehicles.

The last issue we discuss relating to the need for increased autonomy is cooperative waypoint path planning. Site visitation sequencing is a sub problem of this problem and the focus of interest of this work. Often, due to sensor errors, changes in wind conditions, non-availability of complete or high definition maps of the terrain, hostile units or sites of interest, replanning at various levels is necessary. Replanning can often be as difficult as planning and the problem of finding a good plan that meets the constraints is NP complete in general. Also, a remote station doing the job of replanning, specially based on inputs due to changes in the perception of the inflight vehicles themselves of their environment, can require the transmission of high definition digital data. Thus, centralized planning at the base station, especially for several UAVs, can create an avoidable computing bottleneck. Moreover, in the absence of onboard replanning capabilities, the UAV may need to pass sensitive information not available to the base station and thereby permitting the non benign units a greater opportunity for jamming or even eavesdropping. Beyond these considerations, two-way propagation delay, limits on maximum downlink signal strength, possible loss of contact for short intervals of time and other factors could limit the radius of operation of a UAV. This can be particularly limiting for an unmanned combat aerial vehicle (UCAV) in hot pursuit or an 'escaping' URAV.

A transition from manned to unmanned systems promises to liberate man from dangerous, time-consuming and repetitive tasks. Further movement, in the direction of onboard autonomy, is likely to relegate several complex tasks concerning the operation, task execution and cooperative planning to onboard decision support systems [14, 15, 16].

3 Generating MRDTV Graphs

A basic element in our solution methodology for obtaining several alternative site visitation plans for a URAV is a *minimal risk density target visitation,* or the MRDTV, graph. This data structure was introduced in [1] as $(A - Vor(G))$. It is a discreet connected subset of a suitably bounded 3D space over S. Under the assumptions stated in [1] and an expectation that only those hostile ground units will likely target the URAV that are closest to it, its edges represent the set of all possible minimal risk density flight paths for the URAV.

The algorithm for generating an MRDTV graph takes the following inputs –

a. A – the airbase
b. S – the set of all nonintersecting useful vicinities of reconnaissance sites that have been assigned to the URAV or acquired by the URAV.
c. T– the set of threats

The algorithm is outlined as follows. These steps yield an MRDTV multigraph. Further processing using information related to user-requirements and vehicular limitations generates an MRDTV graph. Details regarding this step are not included here.

STEP 1: Generate the Voronoi graph on the set T using the algorithm outlined in [17] or any other optimal algorithm. Elements in T are assumed, without loss of generality, to be points.

STEP 2: Suitably limit the Voronoi graph using a sufficiently large bounding box.

STEP 3: Compute a minimal risk density multigraph as shown in [1].

Our HGA works with an MRDTV graph as input. Specifically, the vertices of the graph bijectively map to the genes in a chromosome. Any two genes in a particular chromosome are said to be *connected* iff in the corresponding MRDTV graph, there exists an edge between the useful vicinities identified by those genes. The HGA uncovers several alternative site visitation sequences of maximum or maximal length (Hamiltonian cycles or acyclic paths) in the MRDTV graph. In case the maximal length paths are acyclic, they are guaranteed to begin at the airbase.

4 Genetic Algorithm

Often in applications of genetic algorithms to vehicle routing problems, researchers do not explicitly consider the issue of starting point of the route [18, 19] since either a complete graph representation is assumed or the vehicles have the possibility of beginning its route at any location. In our application, the URAV must takeoff from a particular site; its airbase. Also, MRDTV graphs are not complete (if fact, they are usually, $O(|S| + |T|)$) and thereby, the Hamiltonian cycles are not a given or trivial to uncover. Thus, in case the HGA is not able to uncover sufficient number of different visitation sequences for the URAV that form a Hamiltonian cycle in the MRDTV graph, it must report the best maximally long paths that originate at the airbase.

In this section, in order to keep this paper as self-contained as possible, we reproduce elements of our HGA from [1] which remain the same in this paper. In addition, we report changes to the HGA which enable us to take into account the problem of starting from a fixed airbase and address the minor problem of generating sufficient number of good alternative plans. Though we report our simulations in the single URAV case, our algorithm can be used in certain cases involving multiple URAVs. The flowchart of the HGA is given in Fig 2.

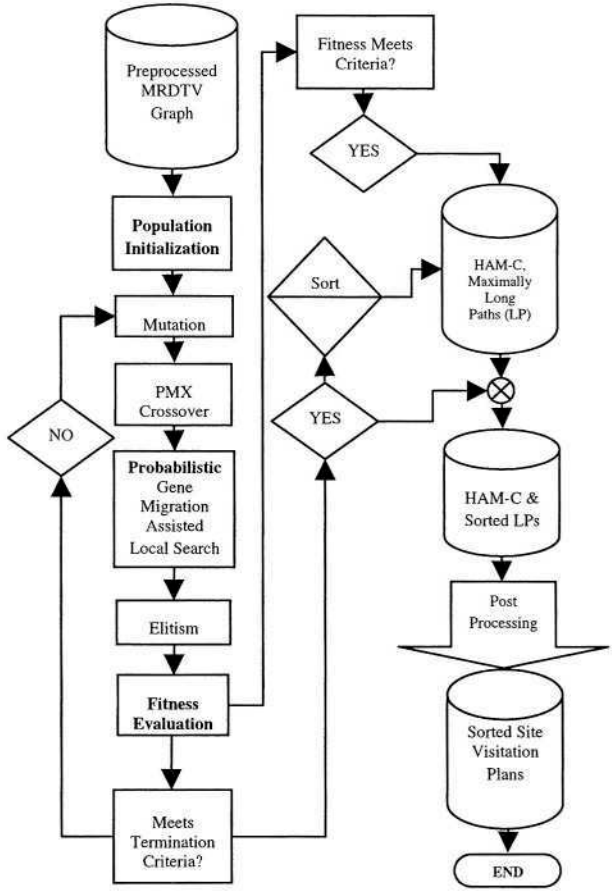


Fig. 2. HGA flowchart. Portions of this algorithm which differ from the one discussed in [1] are in bold

Gene Encoding. The mapping from the set of vertices of the MRDTV graph and the set of genes in a chromosome is bijective. Thus, each chromosome has $|S + A|$ genes. Also, the allelic value of any gene is the same as the numeric label of the corresponding vertex. One of the differences in the HGA used here is that the first gene in all chromosomes is fixed and always has the allelic value of the label for A.

Population Initialization. The population of chromosomes is randomly initialized. However, the allelic value of the first gene in all chromosomes is fixed.

Mutation. Chromosomes are probabilistically selected for mutation. Once selected, a multiple site, paired mutation is performed. Mutation must occur in pairs else they will result in the violation of the bijective mapping scheme. A restriction is that no gene can mutate to the allelic value of the first gene.

PMX Crossover. We select chromosomes in pairs for crossover using a roulette wheel scheme. We used the PMX operator [20] which is a popular scheme in tour-related problems. It guarantees feasible children chromosomes.

Fitness Function. The idea of *connected* genes is mentioned in Section 3. A chromosome's fitness is the length of the uninterrupted sequence of connected genes beginning at the first gene minus one. The objective of our HGA is to maximize this length as opposed to maximize the length of a connected sequence beginning at any gene in [1]. In the limit that one is able to uncover a Hamiltonian cycle in the process, the two objectives lead to the same solution topology – a ring. However, our current method is superior as discussed in Section 1.

Probabilistic Gene Migration Assisted Local Search. The gene migration scheme is a local search scheme, without which, the performance of the HGA is extremely poor. It works as follows. Define *nonconnected* genes as an idempotent negation of *connected* genes in a chromosome. Select a *nonconnected* gene in a chromosome and check if its insertion in any position in the *connected* sequence equals or improves the fitness of the corresponding chromosome; if it does then preserve the migration. This procedure should be repeated for all *nonconnected* genes until all of them are moved to the connected set or, alternately, no such movement is possible within a given chromosome. Currently, we use a uniform probabilistic scheme for selecting the gene that is the next candidate for potential migration as opposed to a sequential selection scheme in [1]. While this doesn't seem to affect the quality of the solutions in our simulations, it does appear to help generating a greater variety of good alternative solutions.

The GA outputs alternative plans in a nonincreasing order of their fitness values.

5 Dataset and Results

The dataset used here are simulated MRDTV graphs which are topologically isomorphic to the dataset for [1]. However, in this work, we generated 3 graphs from each of the 14 graphs in [1]. Each of these triplets differs from each other in the location of the airbase even while being topologically isomorphic to one another. Thus, in all, our data comprises 42 labeled graphs. It is available at [21].

The algorithm was coded in Java++. 15 simulation runs were conducted for each graph on a Pentium IV, 2.4 GHz PC. We generated up to 15 alterative site visitation plans in all simulations discussed here. Some results are tabulated in Table 1. For each triplet of graphs whose unlabeled graphs are isomorphic, the simulation results did not differ much. This suggests that at least for the dataset considered here, the algorithm's performance does not depend much on where the airbase is located. Instead, the overall graph topology is far more important. Thus, in order to conserve space, in each row of results in Table 1, we have given the average values obtained from simulations on three graphs that differ only in the node which represents the airbase.

Table 1. Simulation Results. The dataset was generated from 14 nonisomorphic graphs. It comprises 3 graphs generated from each of these 14 graphs that differ in the vertex which is used to represent the airbase.

Simulated MRDTV GRAPHS	Max. known Path Length (PL)	Results with Graph Compression Preprocessing		
$V_{G_{CG}}$ = number of vertices in compressed graph		Avg. Best PL	Max. PL	Avg.CPU Time (s)
G01(15, 20), $V_{G01_{CG}} = 13$	14	14.0	14	0.040
G02(20, 30), $V_{G02_{CG}} = 20$	20	20.0	20	0.063
G03(20, 37), $V_{G03_{CG}} = 19$	20	20.0	20	0.066
G04(46, 69), $V_{G04_{CG}} = 46$	45	45.0	45	1.04
G05(46, 71), $V_{G05_{CG}} = 46$	46	45.1	46	0.56
G06(46, 93), $V_{G06_{CG}} = 46$	46	45.7	46	0.781
G07(49, 96), $V_{G07_{CG}} = 46$	48	48.0	48	0.461
G08(49, 142), $V_{G08_{CG}} = 49$	48	48.0	48	0.443
G09(100, 197), $V_{G09_{CG}} = 61$	99	95.6	98	4.458
G10(100, 197), $V_{G10_{CG}} = 60$	100	98.1	100	5.986
G11(100, 203), $V_{G11_{CG}} = 60$	100	96.4	99	4.324
G12(100, 207), $V_{G10_{CG}} = 77$	99	97.5	98	7.053
G13(100, 217), $V_{G10_{CG}} = 86$	99	97.6	99	10.913
G14(100, 217), $V_{G10_{CG}} = 85$	100	98.5	99	6.647

6 Conclusion

We have given a hybrid genetic algorithm which finds up to a user-specified number of alternative reconnaissance site visitation plans for an unmanned reconnaissance aerial vehicle. The plans are Hamiltonian cycles and acyclic paths beginning at the airbase in a *minimal risk density target visitation* graph along which the URAV can possibly fly to a large number of useful vicinities of reconnaissance sites without any revisitation. This helps lower the risk to the URAV operating in a hostile zone. A probabilistic gene migration scheme helps us obtain a greater diversity in visitation orderings while not affecting the fitness. Thus a larger number of alternative good site visitation sequencing plans are made available to the user. This can be particularly useful in cases involving large uncertainties or requiring further decision refinements. Our work helps in building onboard decision making capabilities in unmanned reconnaissance aerial vehicles using genetic algorithms. We believe that our general solution methodology is applicable to any unmanned vehicle system (ground, or marine) operating in threat zones. However, the utility of such solutions to other classes of vehicles probably needs further investigations.

References

1. Agarwal A., Lim, M.H., Xu Y.: Graph Compression via Compaction of degree-2 nodes for URAV Visitation Sequencing. UVS Tech 2003. Brussels, Belgium, December 2003
2. Vleugels J., Overmars, M.: Approximating Generalized Voronoi Diagrams in Any Dimension. Intl. J. on Computational Geom. and Applications. 8:201-221, 1998.
3. Arya S., Vigneron, A.: Approximating a Voronoi Cell. Research Report HKUST-TCSC-2003-10, 2003
4. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press, 2001
5. Wagner, I.A., Bruckstein, A.M.: Hamiltonian(t) – An Ant Inspired Heuristic for Recognizing Hamiltonian Graphs. Special Session on Ant Colony Methods, 1999 Congress on Evolutionary Computation. Washington DC, USA, July 1999
6. Rubin, F.: A Search Procedure for Hamiltonian Paths and Circuits. J. of the Assoc. of Computational Machinery. Vol. 21, No. 4, October 1974
7. Frank, Martel, J.C.: Phase Transitions in Random Graphs. 1st Intl. Conf. on Principles and Practice of Constraint Programming, Cassis, France, September 1995
8. Frieze, A. Yukich, J.: Probabilistic Analysis of the Traveling Salesman Problem. Book Chapter, June 2001
9. Vandegriend, B.: Finding Hamiltonian Cycles: Algorithms, Graphs and Performance. MS Thesis, University of Alberta, Spring 1998.
10. Mazade, L. (ed.): The Army's Bandwidth Bottleneck, Congressional Budget Office (USA), August 2003
11. Klausner K.A.: Command and Control of Air and space Forces Require Significant Attention to Bandwidth. Air & Space Power J., November 2002
12. Kopp, C: The UCAV Ascendancy: What are the Problem Issues? UAV Australia Conference. Melbourne, Australia, February 2001
13. Veverka, J., Campbell M.: Experimental Study of Information Load on Operators of Semi-Autonomous Systems. AIAA Guidance, Navigation and Control Conf. and Exhibit. Texas, USA, August 2003
14. CD ROM, Proceedings. UVS Tech 2003. Brussels, Belgium, December 2003
15. Unmanned Systems 2003 Proceedings. Association of Unmanned Vehicle Systems. Maryland, USA, July 2003
16. Unmanned Aerial Vehicles Roadmap 2002 – 2027. US DoD, December 2002
17. Van Kreveld, M., Overmars, M., Schwarzkopf, O., De Berg, M. (ed.): Computational Geometry. Algorithms and Applications. Springer-Verlag, 2nd Revised Edn. 2000
18. Braysy, O.: Genetic Algorithms for the Vehicle routing Problem with Time Windows. Arpakannus 1/2001 - Special Issue on Bioinformatics and Genetic Algorithms. 2001
19. Pereira, B.F., Tavares, J., Machado, P., Costa, E.: GVR: a New Genetic Representation for the Vehicle Routing Problem. 13th Irish Conference on Artificial Intelligence and Cognitive Science. Limerick, Ireland, September 2002
20. Goldberg D.E., Lingle, R.: Alleles, Loci, and the TSP. Proc. of the 1st Intl. Conf. on Genetic Algorithms, New Jersey, USA, 1985
21. http://www.ntu.edu.sg/home/emhlim/dataset/ALCT_SFAPAURAVSVS_GECCO04/

Inflight Rerouting for an Unmanned Aerial Vehicle

Amit Agarwal[1], Meng-Hiot Lim[2], Maung Ye Win Kyaw[2], and Meng Joo Er[1]

[1] Intelligent Systems Center, Techno Plaza, Nanyang Technological University
Singapore 639798
{pg0212208t, emjer}@ntu.edu.sg
[2] School of Electrical & Electronic Engineering, Nanyang Technological University
Singapore 639798
{emhlim, p193494}@ntu.edu.sg

Abstract. The availability of new information can often necessitate rerouting of a vehicle. In [1], the set of nonoverlapping useful vicinities of the reconnaissance sites, S, and the 3D space over them was represented using a discrete graph-model. Here, we consider two cases in which an unmanned reconnaissance aerial vehicle must be rerouted. The first scenario involves not visiting useful vicinities, $S_t \subset S$, due to newly reported unacceptable threat-levels in their neighborhood. In the second scenario, visitations to useful vicinities, $S_r \subset S$, is made redundant due to decreased importance of the sites to which they offer a vantage observation-point. Our earlier graph-model [1] is general enough to provide a basic framework for consideration of rerouting in both cases. We give an algorithm to obtain a graph-model, from the earlier one, that allows efficient gene encoding and efficient application of GA operators to obtain new alternative rerouting plans.

1 Introduction and Background Work

Unmanned reconnaissance aerial vehicles (URAVs) are often used for obtaining high definition photographs or thermographs of sites that are an object of battlefield reconnaissance or those hit by natural disasters such as forest fires or an earthquake [2]. Often, the URAVs use maps of threat, terrain and interesting-site information that are obtained *a priori* using satellite imagery, by high altitude aircrafts or other sources. These maps can however suggest false alarms regarding the presence of threats or potential sites of further interest or, can hide the threats or sites of interest in their resolution. Also, the objects that define these interesting features on the map need not be static in time even in the duration of operation of the URAV. In this work, we address a couple of rerouting problems that potentially arise due to one or more of these factors. Our model is more suited for URAVs that operate in hostile zones in which $|T| = \Theta(S)$ (see the following page for the definitions of the symbols).

Terrains masking permits a URAV to fly largely undetected and obtain very high resolution pictures. However, on the downside, several sites of interest about which information could have been obtained from only a few high altitude observation points now need to be covered from several tens of physically separated useful vicinities. Assuming the user's interest in obtaining information on as many sites in a

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 859–868, 2004.

single sortie as possible, terrain masking imposes a natural increase in the number of geographically separated useful vicinities and thereby, the combinatorial problem of planning the visitation-order to these sites becomes interesting. This problem often involves minimization of multiple parameters such as fuel consumption, time-in-the-threat-zone, distance of travel, total turn-angle or threats that are a nonincreasing function of the Euclidean distance to the vehicle. In this work, our rerouting model accounts for the constraints imposed by the last of these parameters.

Our model for rerouting is obtained from a minimal risk density target visitation (MRDTV) graph. In [1] we gave an algorithm to obtain a minimal risk density multigraph from information on the location of the airbase, *A,* location of useful vicinities, *S,* and the location of known threats, *T,* in the theatre of operation. An MRDTV graph can be derived from the multigraph using certain additional information related to operational requirements. An MRDTV graph can better account for threats that occur in route as opposed to those that are almost collocated with the useful vicinities. Moreover, we take care of risks due to revisitations to any useful vicinity in the gene encoding (Section 3.1) and objective function design (Section 3.3). Fig 1 shows an illustration of a typical input. Fig 2a and Fig 2b show the resulting minimal risk density multigraph and an MRDTV graph respectively. The interested reader may look up the algorithmic details in [1].

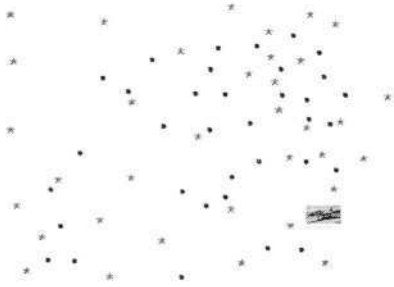


Fig. 1. Input to the algorithm for generating MRDTV graphs. The airbase is represented by the picture (copyright: NFTC, Aeronautical Development Agency). The useful vicinities of sites are the blue dots. The relevant threat sites are the red stars

The input (see Fig 1) information comprises the latitude and longitude coordinates of *A, S* and *T* and the strength of each threat $t_i \in T$. Without loss of generality, for the purpose of this illustration, all threats are assumed equipotent. As a brief note on the time complexity, an MRDTV multigraph can be generated in $\mathbf{O}\,S(\log S)$ time. Certain classes of MRDTV graph can be derived from an MRDTV multigraph in $\mathbf{O}(S)$ time.

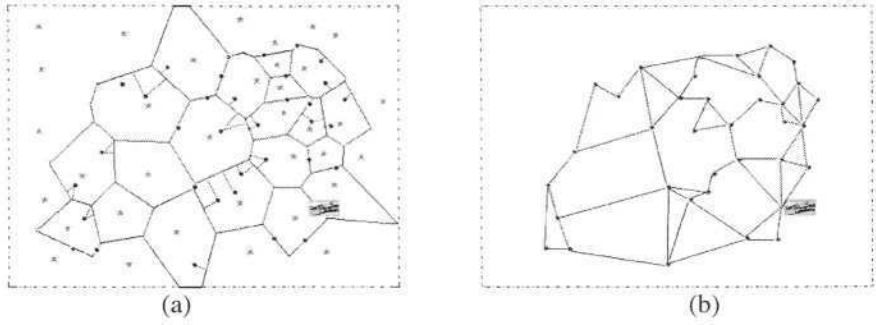


Fig. 2. The resulting minimal risk density multigraph, $G_M((S \cup A), E)$ and the MRDTV graph $G((S \cup A), E)$

An MRDTV graph is a labeled (not shown in Fig 2b) graph. Each label uniquely identifies the corresponding useful vicinity $s_i \in S$. Thus the problem of finding suitable plans in a continuous domain, such as the one shown in Fig 1, is translated to the problem of finding them in a discrete space, such as the one shown in Fig 2b.

2 Inflight Rerouting

Inflight rerouting is necessitated by the reception of new information at the ground station or changes in the vehicle's perception of its own environment during flight. Two basic scenarios arise - *a.* visitations to certain vicinities $S_r \subset S$ are 'now' found to be unnecessary for the mission, *b.* visitations to certain vicinities $S_t \subset S$ should be suspended due to new knowledge of unacceptable threat-levels present in their neighborhood. The former case differs from the latter in that that the URAV can fly over the no-go useful vicinities in case needed without exposing itself to unacceptable threat levels. In our discussion, for simplicity we assume that $S_t \cap S_r = \Phi$. However, the case, $S_t \cap S_r \neq \Phi$, can be handled with trivial increase in programming effort.

2.1 Algorithm for Generating $MRDTV_R$ Graphs

As mentioned in Section 1, a minimal risk density target visitation graph for rerouting ($MRDTV_R$ graph) can be directly obtained from an MRDTV graph. In this section we give an algorithm to do this and illustrate it via an example. We leave the consideration of special cases out.

The inputs to the algorithm are sets S, S_t and S_r and, the set of useful vicinities that have already been visited.

By definition, sites in S_t must be completely avoided. This idea is consistent with removing all vertices corresponding to the elements of S_t and, the edges they induce from the MRDTV graph.

Vertices corresponding to useful vicinities that have been visited before the consideration of rerouting may also be treated as above.

Handling S_r is a little less straightforward – in the gene encoding scheme used here, the mapping from the vertices of a graph to the genes in a chromosome is bijective. Since the objective is to find maximally long sequences of 'connected' (see Section 3.3) useful vicinities so that all sites that should be visited can be visited without requiring revisitations, the HGA will have a strong tendency to include elements of S_r in 'good' flight plans. If however we treat the elements in S_r in the same way as suggested for elements in S_t then, it is certain that no plan will reflect the choice of flying over one or more sites in S_r. This is excessively restrictive for the URAV given the relatively relaxed definition of sites in S_r. Thus, the vertices in the MRDTV graph corresponding to the elements of S_r are handled as follows.

Augment the edge-set of the MRDTV graph with edges that belong to the cliques formed by each set of adjacent vertices of each vertex corresponding to the useful vicinities in S_r. Remove all vertices from the MRDTV graph corresponding to the useful vicinities in S_r.

The application of these techniques yields an $\mathrm{MRDTV_R}$ graph. Discovering rerouting plans will now be equivalent to exploring the topology of this graph. A graphical illustration is shown in Fig 3. Vertex *5* is the airbase.

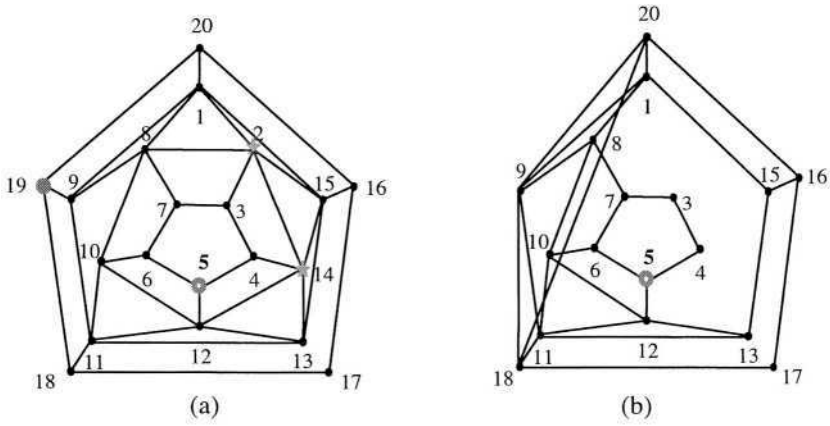


Fig. 3. An MRDTV graph (a) and the resulting $\mathrm{MRDTV_R}$ graph (b) for $S_t = \{2, 14\}$ and $S_r = \{19\}$

The vertices of $\mathrm{MRDTV_R}$ graph are a subset of the vertices of the MRDTV graph. Also, its edges are a subset of the union of edges of the MRDTV graph. Thus, an $\mathrm{MRDTV_R}$ graph retains all the minimal risk density properties of the corresponding MRDTV graph.

The $\mathrm{MRDTV_R}$ graph has $|S + A - S_r - S_t|$ vertices. Upon the application of the algorithm outlined above, its vertices are not labeled by integers from a continuous set. As $|S_r + S_t|$ grows large in proportion to $|S|$, the application of all GA operators whose method includes a random number generator will become inefficient. In order to avoid this, before the GA is applied to discover flight plans, its vertices are

relabeled (not shown in Fig 3b) with a continuous set of integers 1 to $|S + A - S_r - S_d|$. The vertex relabeling information is stored in an array $A[|S + A - S_r - S_d|]$ as follows –

initialize $j = 1$
for $i = 1$ to $|S + A|$,
    if $s_i \in (S + A - S_r - S_d)$
        $A[j] = s_i$
        i++, j++
    else i++

The GA is applied to the $\mathbf{MRDTV}_R$ graph whose vertices are relabeled as above. Once alternative site visitation sequences are generated, the original labeling can be obtained from $A[|S + A - S_r - S_d|]$.

3 The Hybrid Genetic Algorithm

The availability of several competitive site visitation sequences can aid a URAV operator in better decision making. Alternatively, since multi-objective optimization (see Section 1 for its relevance to our problem) is far more difficult, a staged-solution can be helpful. Once several solutions meeting a criterion are obtained, another procedure that better satisfies the constraints on the other parameters can help select between those plans. The implicit parallelism of genetic algorithms [3, 4, 5] in yielding a fairly large set of solutions is thus useful for our application. Our HGA attempts to uncover as many visitation plans, having at least the minimal quality, specified by the user. Critical portions of the hybrid genetic algorithm we used for solving the URAV rerouting problem are outlined in the succeeding subsections.

3.1 Gene Encoding

We use a k-ary gene encoding scheme. Upon the application of the algorithm for generating the $\mathbf{MRDTV}_R$ graph as discussed in Section 2, the vertices are labeled with a continuous set of integers from 1 to $|S + A - S_r - S_d|$. The mapping from each vertex of the MRDTV graph to the genes in a chromosome is bijective. Thus, for example, a gene with allelic value 2 represents the vertex labeled 2, and vice versa. A restriction is placed on this encoding that the vertices corresponding to the useful vicinity that the URAV is heading towards at the time rerouting is necessitated and the airbase are always the first and last gene respectively in any chromosome. Simulations with these restrictions are however deferred.

3.2 Standard Genetic Operators

We use multiple site mutation, and elitism in our HGA. The crossover is implemented with the PMX operator [6] since the standard, single site crossover violates the feasibility of the solutions.

3.3 Fitness Function

Within the consideration of risk minimal rerouting itself (see Section 1 and Section 2 for details), several, not necessarily mutually compatible, measures of a good plan may be defined. A basic could be to discover a set of new alternative routes that begin at the current useful vicinity, or, from the next useful vicinity to which the URAV is in route at the moment when the need for rerouting is communicated or decided onboard, visit all useful vicinities belonging to the set $(S - S_r - S_t)$ and possibly (only if absolutely necessary) fly over the useful vicinities in S_r and get back to the airbase without revisiting any useful vicinity in $(S - S_t)$. This however would be impossible in some cases, for example, in which the next useful vicinity is a critical node in the underlying MRDTV graph and the airbase and at least one useful vicinity in $(S - S_t)$ which has not yet been visited lie in the two unconnected graphs resulting from the removal of the critical node. Other alternative criteria may permit revisitations if made necessary by the underlying graph structure and penalize the number of revisitations. Our goal in this work is to introduce a GA-based approach and additional necessary algorithms for the problem of inflight rerouting for a URAV. In simulations presented here, we have kept the fitness function simple.

We call two genes in a chromosome *connected* if the vertices corresponding to them in the MRDTV graph are adjacent. Let S_c be the useful vicinity that the URAV is about to visit before deciding the new plan for revisitations. Then the fitness of any chromosome is defined as the path length in the MRDTV graph corresponding to the longest, uninterrupted sequence of *connected* genes in that chromosome beginning at S_c.

3.4 Single Gene Migration

In our simulations, the genetic algorithm with only the standard operators performed extremely slowly and, poorly on our test graphs with number of vertices, $|V|$, greater than 20. Using different combinations of GA parameter rates did not help much in improving the results. The GA would converge to a local optimum. The problems with the exclusive application of the standard operators are two folds. First, the standard operators do an almost blind search – good solutions for our problem cannot be constructively, efficiently, built without exploiting the underlying topology since the MRDTV_R graphs are sparse. The second problem with the exclusive application of standard operators is that that the solution-space is vanishing small in comparison with the problem-space. In fact, part of the answer we are trying to seek is not an optimization problem but, rather an NP-complete decision problem. This, in itself, makes the problem difficult for GAs.

Single gene migration is used to mitigate the aforementioned problems. It is a technique for extensive local search and may be applied in each generation. It takes $O(|X|^2)$ time per chromosome where $|X|$ is the length of a chromosome. It works as follows.

In a chromosome we locate the set of genes that define the chromosome's fitness (Section 3.3). Imagining the set of all genes in the chromosome to constitute a universal set, the negation of this set is the set of all genes that are a candidate for migration. Each gene from the candidate set is taken one at a time and inserted between successive consecutive pairs of genes that define the chromosome's fitness. Each time an insertion is made, the chromosome is checked locally if its fitness has decreased; if not then, the migration for the particular gene is preserved. Else the candidate gene remains in its place.

4 Results

Our algorithm is designed to find as many site visitation plans of the specified quality as requested by the user. The running time varies slightly with the number of such plans requested by the user. We generated up to 10 alterative site visitation plans in all simulations discussed here.

Our dataset comprises of fourteen simulated MRDTV graphs. Randomly, a few vertices in each of these graphs are assigned to the sets S_r and S_t and, a vertex is chosen as S_c. The data is available at [7].

Ten simulations were conducted for each graph on a Pentium IV, 2.4 GHz PC. The RAM of the machine is not a particularly important factor since neither our dataset nor the GA chromosomes require excessive memory.

GA Parameters. We tested our HGA using various combinations of crossover rates, mutation rates and elitism figures. Our simulations showed that for graphs with less than 20 vertices, a population size of about 25 proved good. Also, for graphs with vertices less than 25, sufficient number of good site visitation plans could be obtained in about 20 generations. In graphs with less than 50 vertices, we could obtain sufficient good quality results in about 35 generations. However, for graphs with 100 vertices, we ran the simulation for a maximum of 100 generations. The other GA parameters for simulations reported here are as follows –

Mutation probability = 0.05
Number of mutations per chromosome = 20% of the length of the chromosome
Crossover rate =1.0

Figures 4 through 7 plot a typical solution in the HGA run versus the generation number. A preprocessing algorithm called *degree-2* graph compression [1] is run on each graph before the HGA. The plots exhibit an interesting feature during the initial generations – they are always extremely poor but rise quickly to near-optimal levels. The single gene migration operator strongly contributes to the steep upward slope of the graph exhibited during the initial generations.

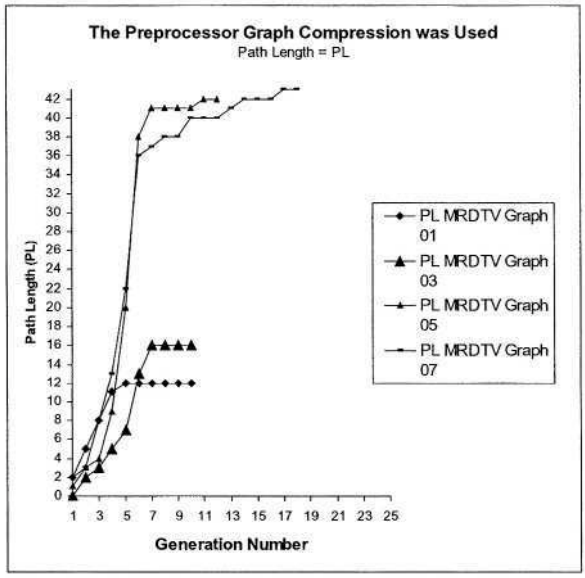


Fig. 4. Typical plots of path lengths obtained using the HGA in graph 01, graph 03, graph 05 and graph 07 v/s the generation number

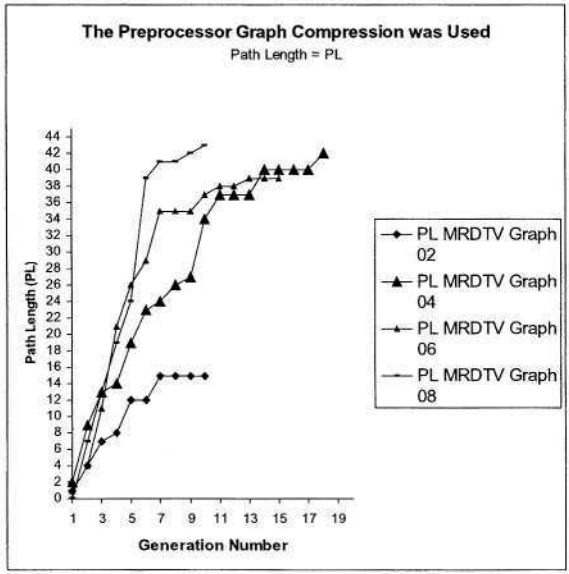


Fig. 5. Typical plots of path lengths obtained using the HGA in graph 02, graph 04, graph 06 and graph 08 v/s the generation number

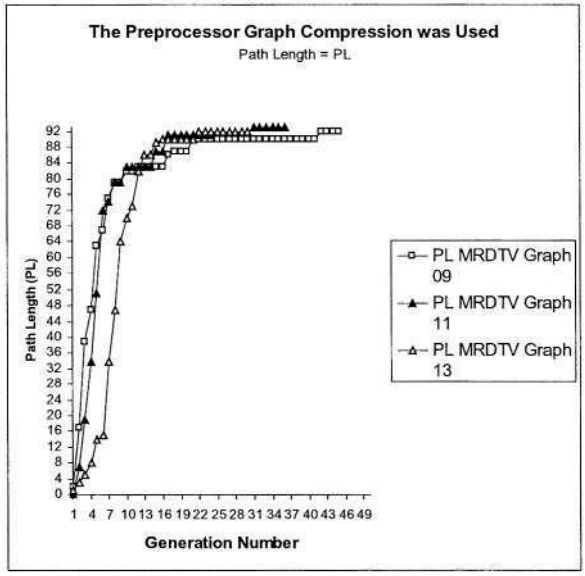


Fig. 6. Typical plots of path lengths obtained using the HGA in graph 09, graph 11 and graph 13 v/s the generation number

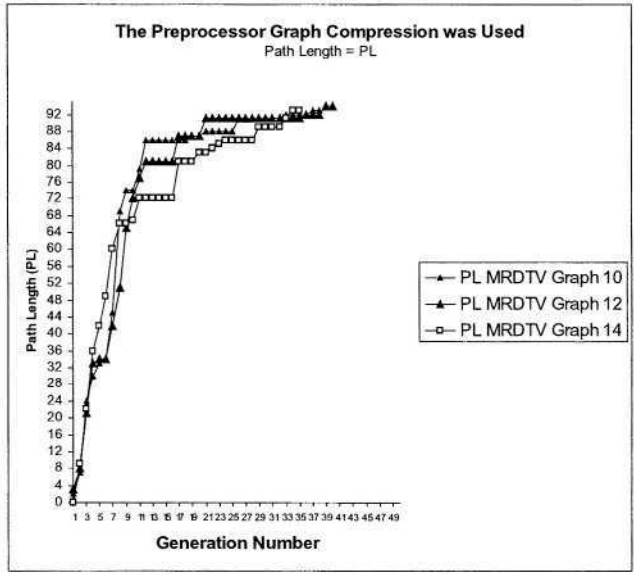


Fig. 7. Typical plots of path lengths obtained using the HGA in graph 10, graph 12 and graph 14 v/s the generation number

5 Conclusion

In this paper, we discussed tow scenarios that can necessitate rerouting for an unmanned reconnaissance aerial vehicle. We used hybrid genetic algorithms to give competitive sets of solutions comprising alternative useful vicinity visitation plans that enable the vehicle to avoid in route risks and conserve time and fuel by not flying over sites with mitigated importance. The input to the HGA is an MRDTV_R graph. We provided an algorithm to generate it from a *minimal risk density target visitation* graph. A simple algorithm was given to translate the labels of an MRDTV_R graph to another set of labels. This translation helps greatly in making the implementation of all GA functions that call a random number generator more efficient; especially if $|S_r + S_d| = O(S)$.

References

1. Agarwal A., Lim, M.H., Xu Y.: Graph Compression via Compaction of degree-2 nodes for URAV Visitation Sequencing. UVS Tech 2003. Brussels, Belgium, December 2003
2. Unmanned Systems 2003 Proceedings. Assoc. of Unmanned Vehicle Sys. Maryland, USA, July 2003
3. Erick C.P., James, A.F., Deb, K., Lawrence, D., Roy, R. (eds.): Proc. Genetic and Evolutionary Computation – GECCO 2003. Springer Verlag Pub. Chicago, USA, July 2003.
4. Mitchell, M.: An Introduction to Genetic Algorithms, Reprint Edn. MIT Press, February 1998
5. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Pub. Co. January 1989
6. Goldberg D.E., Lingle, R.: Alleles, Loci, and the TSP. Proc. of the 1st Intl. Conf. on Genetic Algorithms, New Jersey, USA, 1985
7. http://www.ntu.edu.sg/home/emhlim/dataset/ALY_IRUAV_GECCO04/
8. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. 2nd edn. MIT Press, 2001
9. Unmanned Aerial Vehicles Roadmap 2002 – 2027. US DoD, December 2002
10. Pereira, B.F., Tavares, J., Machado, P., Costa, E.: GVR: a New Genetic Representation for the Vehicle Routing Problem. 13th Irish Conference on Artificial Intelligence and Cognitive Science. Limerick, Ireland, September 2002

Memetic Optimization of Video Chain Designs

Walid Ali[1] and Alexander Topchy[2]

[1] Philips Research Laboratories, 345 Scarborough Rd, Briarcliff Manor, NY, 10510, USA
walid.ali@philips.com
[2] Department of Computer Science and Engineering
Michigan State University, East Lansing, MI, 48824, USA
topchyal@cse.msu.edu

Abstract. Improving image quality is the backbone of highly competitive display industry. Contemporary video processing system design is a challenging optimization problem. Generally, several video algorithms must be sequentially applied to real-time video data. Overall image quality depends on the nonlinear interactions between multiple design parameters: variable settings for each module (algorithm), the amount of data being transferred in the video processing chain as well as the order of the cascading modules. Unfortunately, no systematic techniques are currently available to configure the video chain without lengthy trial and error process. We propose a rapid and reliable method for optimization of composite video processing systems based on genetic algorithm coupled with local search heuristics. Video system configuration is evolved toward the best image quality, driven by an objective video quality metric. We analyze several local search approaches, including hill-climbing, simplex and estimation-of-distribution algorithms. Experimental study demonstrates superior performance of memetic strategies over the conventional genetic algorithm. We obtain novel and practical video chain solutions that are typically not attainable by regular design process.

1 Introduction

Picture quality is a key factor influencing consumer brand name preference for video communication devices [1, 25, 27]. These appliances range from small hand-held devices to a large complicated television sets. They all deploy a number of video processing modules, which interact together to create the desired output picture. Generally, video algorithms are developed and evaluated in isolation from the actual video processing system, of which they will be a part in a consumer product. Obviously, the final quality depends on the interaction of the constituent algorithms. This interaction depends on the order in which these modules are applied, as well as on the settings of each algorithm's programmable parameters. Whereas the development of individual video processing algorithms (modules) involves a lot of analysis and simulation, the development of larger chains often involves a more ad-hoc approach. However, a thorough analysis of the inter-algorithm interaction is required in order to

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 869–882, 2004.

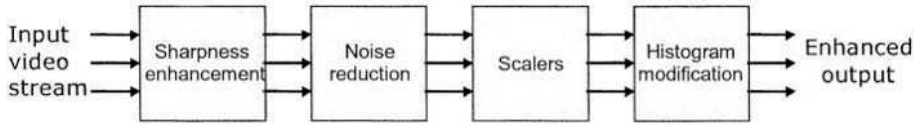


Fig. 1. An example of video chain configuration with four modules (algorithms).

find the optimal system architectural design and the best tuning of each individual module's parameters.

This challenge can be defined as a formal optimization problem called Video Chain Optimization (VCO) [24]. A genetic algorithms approach has been proposed to drive the optimization process [4, 7, 10, 24] and has proven to be successful in leading the search toward the global optima. However, this method is hindered by the time it requires to converge to satisfactory design solutions. Simulation time on order of days is usually necessary to reach the appropriate designs. This is mainly due to the computational complexity of video algorithms and the need to simulate a large number of video systems before converging to (sub)optimal configurations and settings of video chain.

This paper presents our approach for speeding up both the convergence rate and the execution time of optimization. We speed up the convergence by augmenting gray-coded genetic algorithm (GA) with a less complicated search method (local search). Thus, we benefit from the GA's ability to traverse the overall search space without being trapped in local optima while converging faster using neighborhood move operators. Furthermore, the execution speed is improved by keeping the samples visited over the search path in memory and using them for future search trials.

2 Background on Video System Design

The video-processing filter to be optimized consists of multiple video-processing modules, which are considered essential for high-end and top-end television sets. We deal primarily with video signals in the YUV and RGB domains, i.e. with image enhancement and display adaptation functions [9, 14]. Tuning, IF/color decoding, and channel/source decoding are not considered for now. The functions used are luminance peaking by sharpness enhancement, spatial scaling, noise reduction and histogram modification [9]. For example, Fig. 1 shows a particular video chain configuration with these functions assigned to separate modules.

Sharpness enhancement, which nowadays is a common feature in TV sets, focuses on improving the perceived sharpness of the luminance signal. Boosting the higher frequencies in the luminance signal can enhance the sharpness. The *noise reduction* unit reduces the higher frequency components based on measuring the presence of noise. The *scalers* are implemented using polyphase FIR filters. The horizontal scalers process each line of input video data and generate a horizontally scaled line of output video data. In the case of expansion, this is done by up-sampling that is per

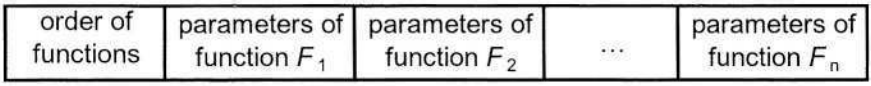


Fig. 2. A general structure for the chromosome representing a video processing chain

formed either by a polyphase filter for which the horizontal expansion factor determines the filter phases required to generate each output pixel, or by a filter that uses this factor to interpolate the output pixels from the input pixels. In the case of compression, a transposed polyphase filter is used to down-sample the input data, and the horizontal compression factor determines the required filter phases. The vertical scalers, however, generate a different number of output video lines than were input to the module, with input and output lines having the same numbers of pixels. Vertical scalers may either compress or expand the number of output video lines. *Histogram modification* stretches out the luminance value for the black color and the white color to better represent the color contents of the video sequence.

3 Applying Genetic Algorithms to the Video Chain Design

The optimization process utilizes genetic algorithms (GA) to search for the parameter settings, implementation alternatives and an interconnection scheme delivering the best objective picture quality. In optimizing the video-processing scheme, a chromosome defines a certain way in which different video processing modules are connected and video stream is transformed. A chromosome consists of a number of genes. The genes in the video optimization problem encode the parameters of video processing functions as well as the order of modules, which determines the connection scheme. Fig. 2 shows a general structure of the string (chromosome) encoding a video processing chain with *n* functions.

We optimized a video processing system, which consisted of four cascaded video processing modules, namely, a spatial poly-phase scalar, a noise reducer, a sharpness enhancer and a histogram modification module. The optimization algorithm deals with each module as generically as possible. It assumes no prior information about the modules nor the connectivity constraints on the cascaded modules. The search process seeks not only the optimal values of pre-defined set of parameters within each module, but also the data bus parameters. The data precision (number of bits in a data bus, i.e., bus width) between two cascaded modules is considered a parameter to be optimized. We elected to use this set of video processing modules because of their vital role in any video system [9, 23]. Moreover, some of these modules are competing modules [9, 14], e.g., increasing the sharpness would enhance the perceived existing noise and reducing the noise will blur the picture resulting in the loss of appealing crispiness.

The complete optimization scheme consists of three main components: (i) model of the video processing system, (ii) the objective image quality measurement component, and (iii) the search procedure with genetic algorithm in its core. It must be noted

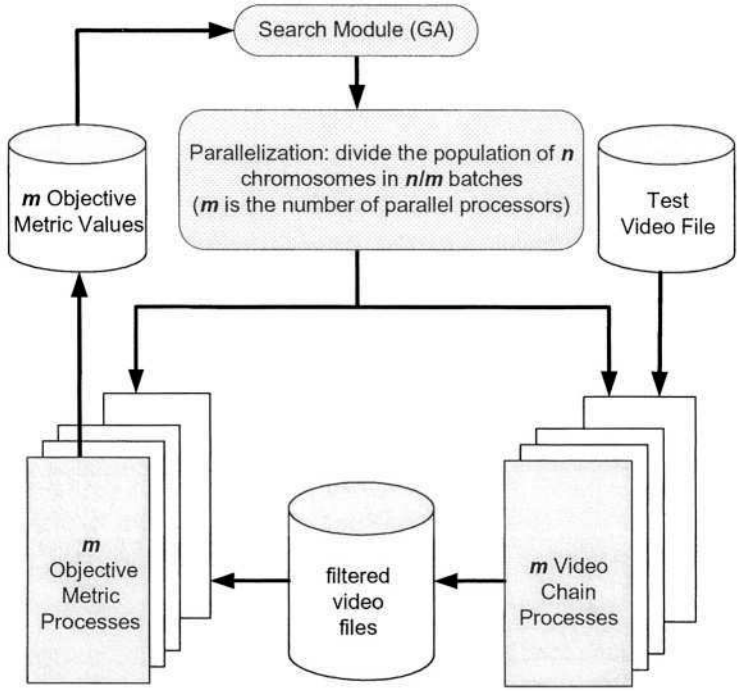


Fig. 3. A schematic diagram for optimizing a video system

function order 5 bits	Noise Reduction 2 bits	sharpness 3 bits	Data Bus Width 3 bits	Data Bus Width 3 bits	Data Bus Width 3 bits

Fig. 4. The chromosome encoding for a constraint-free video chain

that the video system is entirely modeled in software. However, the obtained design solutions are easily transferred to the ultimate display product based on specialized hardware.

The computational bottleneck in this scheme results from the simulation complexity of the video processing system. We run a number of video processing systems in parallel (depending on the available computation processors on a parallel computer), as well as a number of the objective image quality metric components. Parallelizing the computationally demanding portions of the system significantly improves the performance. Fig. 3 shows a schematic diagram of the overall system.

3.1 A Scalable Random Order Video Processing System

The primary goal of this study is the optimization of scalable random order video system. In particular, we consider the video-processing filter without any prior constraints on the modules' parameters, order and the data bus width. The noise reduction unit has a smearing effect, which can be controlled by four settings: 1, 2, 3 or 4, thus 2 bits are needed to represent it. The sharpness enhancement has a parameter with five settings (3 bits). The number of bits transferred between any two cascaded video processing units could range between 8 and 12 bits (5 settings; 3 bits are needed for encoding). There are 24 possible ways of cascading 4 video-processing modules, and 5 bits are needed to represent it. Thus, the chromosome needed to represent this video-processing filter comprises 19 bits. The video filter has 60,000 possible variations. Fig. 4 shows the chromosome structure in this case.

4 Objective Image Quality Metrics as Fitness Function

Evaluation of video quality has always been achieved using subjective methods [1, 11]. Since the subjective results vary according to the variability between the viewing audiences, subjective results, which are solely based on perception, have to be statistically post-processed in order to remove the ambiguity resulting from the non-deterministic nature of these results. Linear and nonlinear heuristic statistical models [21] have been proposed to normalize these results, and produce certain figures of merit to represent the goodness/degradation of the video quality. However, relying on human evaluation is expensive and sometimes impossible to adopt. Apparently, a subjective evaluation cannot be used within the optimization loop, when tens of thousands of trials are performed. Hence, automatic methods to evaluate video quality are necessary.

Ideal automatic and objective assessment of video quality should highly correlate with subjective testing [5, 25]: the higher the correlation, the better the objective method is. In practice, different methods are investigated for objective image quality. They vary widely in complexity and performance and can be categorized in several ways, measuring traditional analog vs. digital artifacts, measuring the general perceptual quality of a video sequence vs. measuring a specific artifact only, and finally still image (frame/field) evaluation vs. temporal evaluation. For instance, nine proponent models were proposed to the Video Quality Expert Group (VQEG) [25].

We employ a composite scalable objective metric, which consists of a set of metrics, each of which is geared toward measuring a certain feature of the video sequence. Each of these n metrics gives a reading, f_i, $(1 \le i \le n)$, which measures a certain feature of the video sequence I. These readings are weighted by a weight factor each, w_i $(1 \le i \le n)$ and linearly combined:

$$F = \sum_{i=1}^{n} w_i f_i(I) \tag{1}$$

Of course, the set of weights $\{w_i\}$ must be determined in advance. Weights are found by maximizing the correlation factor R between human perceptual evaluation and the composite objective measure F on the baseline training data. In particular, we maximized Spearman rank order [21] that measures the correlation factor R_S between the subjective evaluations X_r and the objective metrics $Y_r = F(I_r)$ over a number of training video samples $\{I_r\}$:

$$\max_{w_1,\ldots,w_n} R_S(w_1, w_2, \ldots, w_n) \tag{2}$$

$$R_S = 1 - \frac{6\sum_{r=1}^{m}(X_r - Y_r(w_1, w_2, \ldots, w_n; I_r))^2}{m(m^2-1)} \tag{3}$$

A more detailed explanation and illustration of the objective image quality metric and its constituent metrics is given in [24].

5 Memetic Optimization Approach

The problem we are trying to solve, video chain optimization, has a distinctive set of features. Studying the nature of these features is the best way to decide on the most suitable optimization method to adopt. VCO problem is computationally expensive, since it deals with video processing, has a highly nonlinear, non-differentiable cost function, and is strongly constrained, (e.g., the admissible range of any parameter), and finally, it can be parallelized.

For several decades it has been well understood that competitive optimization methods should include both global and local search as a tradeoff between exploration and exploitation of search space. The evolutionary search is no exception. Indeed, evolutionary search proved to be efficient in exploration of large solution spaces. Genetic algorithms deserve special attention since they are known to be robust across the wide spectrum of optimization problems [8] including real world applications. However, genetic algorithms are not particularly successful in fine-tuning of the candidate solutions. GAs can quickly identify promising search areas. Implicitly operating with many relatively short building blocks, a GA quickly converges toward better solutions. But if only a few bits are not correctly set in the chromosome, it may be very difficult for a GA to find them. An extra search is often necessary for the fine-tuning of the solutions. For solutions that are time-critical, proper local search component plays important role.

Though many hybrid optimization techniques are in use with or without GA playing a part in a search, we focus on the genetic algorithm and local search (LS) combination. Several names are applied to this combination. It is known as particular case of metaheuristic algorithms [28], Lamarckian evolution, Baldwin effect [26], genetic local search [22], or more recently as memetic algorithms [19]. Actual problem dictates the choice of local search component. Many successful applications utilized hill-climbing [16], gradient-descent [15] and Nelder-Mead's Simplex [20]. Extra local

search usually exploits the best solutions within population and moves to new points via small modification of the old solutions. Thus, exploitation of the existing information is achieved by local search in the vicinity of the best solutions. Certainly, the definition of the small moves depends on the metric imposed upon the search space and corresponding encoding of solutions. Typically, an elementary move causes minimal change in the solution under some distance definition. For example, choice of Hamming distance between solutions in binary representation naturally induces a bit flip as elementary modification. The immediate neighborhood of a point consists of all points reachable via a single bit flip. With different representation of solutions neighborhoods may radically differ.

In order to improve optimization speed, a knowledge-based search strategy can be used. Since no prior information on the search space is available, we extract this information from the GA population. Thus, it is important to decide on how to combine GA with local search. There are many ways to achieve this. First, we have to decide on whether the local search interleaves with evolutionary operators during a regular GA run or if that happens only at the end of a GA run as a form of fine-tuning. Taking into account that GA rather quickly provides good indications for best global regions in the search space, it is reasonable to allow limited amount of search in the vicinity of the current best solutions during the GA run.

It is also important to decide if the changes due to the local search are coded back to the chromosome or not, that essentially is the choice between Lamarckian and Baldwin type of evolution [26]. In Lamarckian evolution, learning (via local search) affects fitness distribution as well as the underlying genotype, while the Baldwin effect is mediated via the fitness values only. In our case, the question is whether locally learned values are copied back into the genotype (Lamarckian) or whether the chromosome remains unmodified while the individual's fitness is changed by local search (Baldwin).

Even though there are indications that Lamarckian and Baldwin evolution demonstrate comparable performance in some circumstances [26], our approach is to allow modified genetic code to be written back to the chromosome, thus following Lamarckian style, which proves to better improve the GA solution. Below we will discuss three different implementations of hybrid GA used in our experimental study.

5.1 Next-Ascent Stochastic Hill Climbing

This type of *hill climbing* makes a change in a random gene. If this change results in better or equal fitness value, then a new solution is accepted. It differs from steepest ascent since we do not exhaustively search through the neighborhood for the best improvement, but rather accept *any* improvement. The accepted solution becomes a member of the current population at this moment. In a parallel implementation, several current best solutions are subjected to hill-climbing modifications simultaneously.

5.2 Simplex Method

The Nelder-Mead's simplex algorithm [13] uses a geometric figure consisting of d+1 points in d dimensions. In this method, a new simplex is created from an old simplex by replacing the worst evaluation point with a new point. The previous worst point is reflected over the mean vector $\mathbf{c}$ of d best points to create a new point:

$$\mathbf{c} = \frac{1}{d}(\mathbf{x}_1 + ... + \mathbf{x}_d) \tag{4}$$

$$\mathbf{x}_{\text{new}} = \mathbf{c} + \mathbf{c} - \mathbf{x}_{\text{worst}} \tag{5}$$

The new point is evaluated, and simplex transformation continues as before. In order to create every new point, d+1 points are drawn at random from the current population. Many modifications to this basic procedure are known. The most popular approach [13] uses adaptive expansion and contraction of the simplex depending on the objective function value of a new point. A variant of Nelder-Mead's simplex was also used in hybrid GAs [20, 29, 3]. Our GA-Simplex hybrid iterates GA with limited amount of simplex search every generation.

5.3 Local Search Directed by Estimation Distribution Algorithm

Any knowledge about the problem is potentially valuable because it allows us to bias the search toward more promising solutions. Often this knowledge is gained only during the optimization itself. Therefore, it is important to control the search on the fly by the information that is explicitly accumulated with the experience. Explicit search statistics [2] can be used to shape this information and make it useful for generating new solutions. A whole family of such algorithms relies on learning the probability distributions among the best solutions [17]. New points are sampled from such a distribution, evaluated, and used to refine the distribution, in turn. These methods are known as *estimation of distribution algorithms* (EDA) or *probabilistic modeling* and can be used instead of GA for all optimization purposes. In our approach we do not abandon GA as an exploration tool, but use explicit probability distributions for the local search only. Though many models are suitable for capturing the distribution, we use the simplest and most inexpensive one: marginal distribution of individual alleles. This distribution is simply summarized by the vector of probabilities of alleles at every locus $\mathbf{P}_{\text{good}}=\{P_{\text{good}}(x_1), P_{\text{good}}(x_2),\ldots, P_{\text{good}}(x_n)\}$ assuming that alleles are independent. It approximates the full joint probability distribution of the alleles of the best solutions as the product of unconditional probabilities of individual alleles:

$$P(x_1, x_2,\ldots, x_n) = P_{\text{good}}(x_1)\, P_{\text{good}}(x_2) \ldots P_{\text{good}}(x_n) \tag{6}$$

With binary coding, the components of vector $\mathbf{P}_{\text{good}}$ contain the probabilities of 1's at the corresponding genes. This distribution is used to guide the local search. Every candidate solution subjected to the local search is modified by the distribution $\mathbf{P}_{\text{good}}$.

In contrast to other EDA-type algorithms, we blend the distribution with the candidate individual. Thus, instead of drawing completely new individuals only from the distribution $\mathbf{P}_{good}$, the candidate solution is modified by the information on good solutions available at the current generation. The modified individual is sampled from the distribution $\boldsymbol{Q}(x_1, x_2, \ldots, x_n)$:

$$\boldsymbol{Q}(x_1, x_2, \ldots, x_n) = \left\{ \frac{x_1 + P_{good}(x_1)}{2}, \frac{x_2 + P_{good}(x_2)}{2}, \ldots, \frac{x_n + P_{good}(x_n)}{2} \right\} \tag{7}$$

Sampling from the distribution $\mathbf{Q}$ produces new solutions closer to the old solution in comparison with simply drawing from the distribution $\mathbf{P}_{good}$. This is more adequate to our goal of making just slight modifications in the vicinity of the old solutions. Unlike mutations, EDA-based modifications are not uniformly random, but rather directed by the currently known distribution of the good solutions. In general, one can apply unequal weights to the shares of probabilities coming from $\mathbf{x}$ and from $\mathbf{P}_{good}$:

$$\mathbf{Q} = \lambda \mathbf{x} + (1-\lambda)\mathbf{P}_{good} \tag{8}$$

In addition, we need to take care that the information in $\mathbf{P}_{good}$ is properly updated every generation. In our implementation, only solutions whose fitness value exceeds the mean population fitness contribute to $\mathbf{P}_{good}$. Distribution of good solutions from the *current* population is summarized in $\mathbf{P}_{curr.good}$ at each generation and is used to update the distribution $\mathbf{P}_{good}$ estimated from the previous generations:

$$\mathbf{P}_{good}(t) = \alpha \mathbf{P}_{good}(t\text{-}1) + (1\text{-}\alpha)\mathbf{P}_{curr.good} \tag{9}$$

By altering the factor $0<\alpha<1$ we can reduce the contribution of the previous distribution and introduce new information updating $\mathbf{P}_{good}$ incrementally, from generation to generation.

6 Empirical Study

A real video chain described in section 3.1 was optimized in the experiments. The GA package *dCHC* by Eshelman [6] served as a basis for all hybrid algorithms. We have studied the performance of three different combinations of a genetic algorithm with local search procedures, and compared them to the pure GA solution obtained by dCHC algorithm. The following memetic algorithms were implemented:

1. GA + simplex local search
2. GA + next-ascent stochastic hill-climbing
3. GA + EDA-based local search

The dCHC algorithm was used as the baseline genetic algorithm. Both half uniform (HUX) and 2-point crossover were employed for genetic recombination [6]. Other typical features of dCHC include cross-generation selection, maintaining diversity and soft restart. All the experiments were conducted with population size 50. Variable amounts of 10-20 modifications by local search were performed on the best individu-

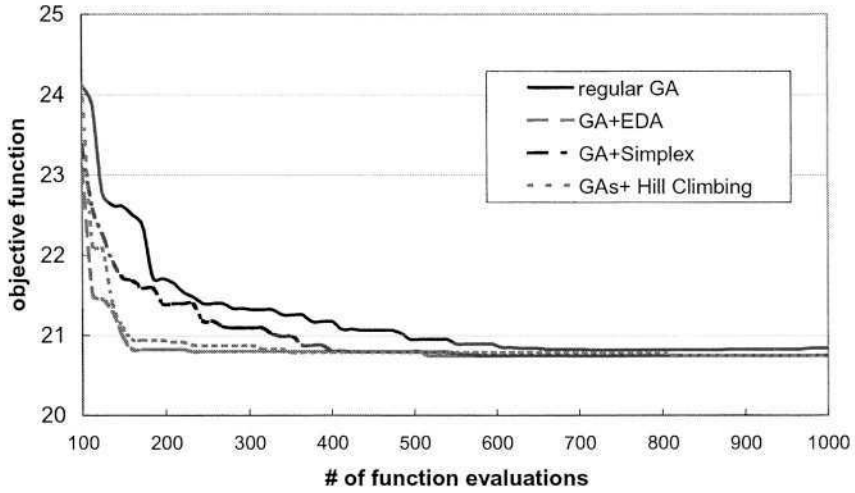


Fig. 5. The best solution found after a set of trials for different memetic heuristics

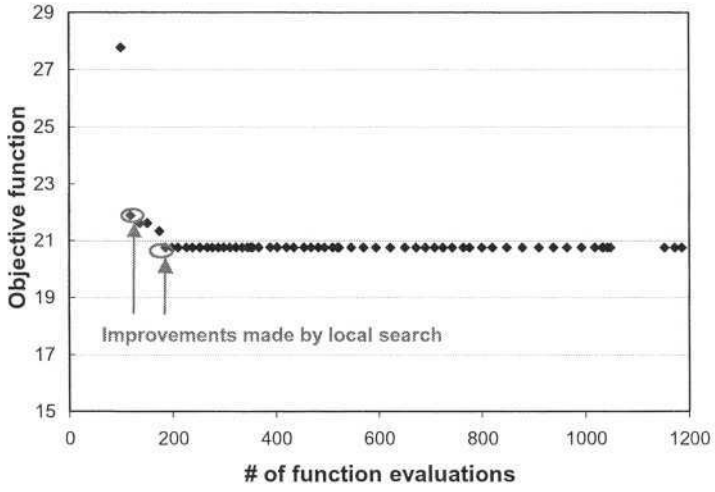


Fig. 6. A typical improvement provided by local search in hybrid GA+EDA algorithm.

als every generation. The number of modifications depends on the number of off spring produced, which varies from generation to generation due to incest prevention mechanism [6]. Since we evaluate the children in parallel, some processors may be free and therefore can be used to run extra evaluations issued by the local search. On average, the relative effort by the local search was lower in the beginning of the run and higher before the convergence, since the number of new children produced by dCHC in every generation is smaller as the population converges. A video chain configuration was represented by a chromosome, which has 6 different genes and 19 total

bits. A high quality solution for this problem was known in advance as found by multiple runs of pure GA, each run taking several days to complete. All the trials, including local search, were done in parallel in batches of 10 evaluations (number of used parallel processors = 10). Two types of information were of particular importance: the average number of trials to reach the best solution and the dependence of the current best performance on the number of evaluations.

Fig.5 shows the objective function value (image metric) of the best solution as a function of the number of trials (objective function evaluations) averaged over 20 different runs. The total count of trials includes both GA trials as well as local search trials. The main observation is that all the hybrid algorithms performed better than the original genetic algorithm. All hybrid algorithms operated with strings in Gray coding representation for the parameters of the video chain. A minor difference in favor of the Gray representation was found in the experiments. The sample VCO problem was solved fastest by the hybrid algorithm with the local search guided by the estimation distribution algorithm. The least improvement was achieved by the GA + simplex hybrid algorithm.

As seen from the Fig. 5, hybrid GAs have converged faster than the pure GA. This is simply because of the local search's better ability to fine-tune the solution faster than a population-driven (like GAs) search method. Fig. 6 shows a typical improvement achieved by introducing local search to pure GAs.

Table 1 summarizes trial statistics to reach the known global optima optimal solution for the video chain optimization problem using different combinations of GA with local search procedures. We report the average and the standard error of the number of trials needed to reach the best solution.

Table 1. Average number of trials necessary to reach the best known VCO solution for different hybrid algorithms, averaged over 30 runs each.

Algorithm	Average	Std. Err.
GA + EDA	**268**	21
GA + Hill-climbing	336	48
GA + Simplex	481	40
GA	635	141

6.1 Improving Execution Time by Solution Caching

Since the most time consuming process is applying the objective image quality metric, we can use a fast retrieval memory mechanism (e.g., a hash table) to store each evaluated chromosome (video system) together with its associated image quality value. Utilizing the memory unit prevents many costly evaluations, which would otherwise unnecessarily slow down the optimization process. Without caching the evaluated configurations, we may need to re-evaluate a chromosome if redundant

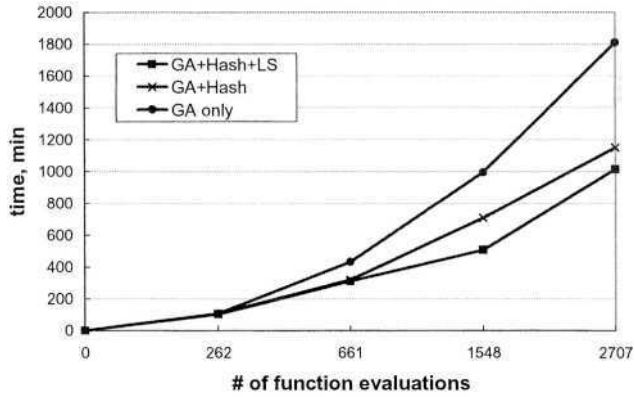


Fig. 7. Optimization speed-up due to candidate solutions caching.

encoding results in two different bit representations of the same physical solution configuration (over-mapping) or when population close to convergence.

In general, if a GA needs to find the fitness (the image quality) of N video designs, out of which only M are not tested before, then we expect to get a gain in computation time:

$$\Delta T = (N - M)T_{eval} - N \cdot T_{look-up} - M \cdot T_{update} \tag{10}$$

Here T_{eval} is the time of a single evaluation, $T_{look-up}$ and T_{update} are the times required for finding a record in memory and for writing a record, respectively. The gain is positive if

$$T_{eval} > \frac{N \cdot T_{look-up} + M \cdot T_{update}}{N - M} \tag{11}$$

This is obviously the case when dealing with video systems, where the evaluation time may extend to 6-10 minutes, while the look-up and update time is of the order of nanoseconds. Our implementation of this memory is based on a classical linked list hash. Hash memory has reasonable space requirements for keeping all the visited solutions and their values. Figure 7 shows the actual speed-up of GA variants with solution caching.

7 Conclusion

We presented a method for automatically optimizing a complicated real-world video processing system, without prior information about the constituent video processing components. Using an automatic optimization method necessitates the use of a cost function, which evaluates the perceptual image quality automatically. We introduced a method to combine a number of image quality metrics to maximize the correlation between the perceived quality and the measured objective quality.

We described several approaches to speed up the execution time needed for finding the (sub) optimal solution in a GA driven optimization problem. The solution quality was improved by hybridizing GA with different local search methods, while the run time has been reduced by using memory hash table of previously visited sample points. Several implementations of memetic algorithms have been presented and their performances have been illustrated.

Memetic optimization allows us to keep the versatility provided by GA while speeding up the rate of convergence toward the global optima. In addition, memory-supported GA proves to be very useful, as it avoids re-calculating the objective function especially toward the final convergence.

References

1. Ahumada, A J. Jr.: Computational image quality metrics: A review, in SID Symposium, Digest, vol. 24, (1993) 305-308
2. Baluja, S.: Genetic Algorithms and Explicit Search Statistics, In Mozer, M.C., Jordan, M.I., Petsche, T. (Eds.) Advances in NIPS 9, MIT Press, Cambridge, MA, (1997) 319-325
3. Boschetti, F., Dentith, M., List, R.: Inversion of seismic refraction data using Genetic Algorithms, Geophysics, (1996) 1715-1727
4. Chipperfield, A., Fleming, P.: Parallel Genetic Algorithms, in Parallel and Distributed Computing Handbook, A. Y. H. Zomaya ed., McGraw Hill, (1996) 1118-1143
5. Daly, S.: The visible differences predictor: An algorithm for the assessment of image fidelity, in Digital Images and Human Vision, ed. A. B. Watson, MIT Press, (1993) 179-206
6. Eshelman, L., The CHC Adaptive Search Algorithm. How to Have Safe Search When Engaging in Nontraditional Genetic Recombination, in G. Rawlins, editor, Foundations of Genetic Algorithms, Morgan Kaufmann, (1991) 265-283
7. Goldberg, D. E.: Genetic Algorithms in Search, Optimization and Machine Learning, Reading, Mass. Addison-Wesley, (1989)
8. Goldberg D.E.: Zen and the Art of Genetic Algorithms, In Proc. of the Third Intl Conf. on Genetic Algorithms, Fairfax, VA, ed. by J.D. Schaffer, Morgan Kaufmann, San Mateo CA, 80 (1989)
9. de Haan G., Video Processing for Multimedia Systems, CIP-Data Koninklijke Bibliotheek, The Hague, (2000)
10. Husbands P.: Genetic Algorithms in Optimization and Adaptation, in Advances in Parallel Algorithms, Kronsjo and Shumsheruddin (eds.), (1990) 227-276
11. ITU-R Recommendation 500-7: Methodology for the subjective assessment of the quality of television pictures, ITU, Geneva, Switzerland, (1995)
12. Morris, G.M., Goodsell, D.S., Halliday, R.S., Huey, R., Hart, W.E., Belew, R.K., Olson, A.J.: Automated docking using a Lamarckian genetic algorithm and an empirical binding free energy function. J. Comput. Chem., 19, (1998) 1639-1662
13. Nelder, J. A., Mead, R.: A simplex method for function minimization. Computer Journal, **7** (1965) 308-313
14. Ojo, O.A.: An Algorithm for Integrated noise Reduction and Sharpness Enhancement, Proceedings of International Conference on Consumer Electronics, Los Angeles, (2000) 58-59

15. Quagliarella, D., Vicini, A., Hybrid genetic algorithms as tools for complex optimisation problems, in New Trends in Fuzzy Logic II, Eds. P.Blonda et al., World Scientific, Singapore (1998)
16. Parmee, I.C.: High-Level Decision Support for Engineering Design using the Genetic Algorithms and Complementary Techniques. Developments in Artificial Intelligence for Civil and Structural Engineering, Civil-Comp Press, Edinburgh, UK, (1995), 197-204
17. Pelikan, M., Goldberg, D.E., Lobo, F. : A Survey of Optimization by Building and Using Probabilistic Models, Proceedings of the American Control Conference (ACC-00). Chicago, IL, (2000)
18. Quagliarella, D. and Vicini, A.: Coupling Genetic Algorithms and Gradient Based Optimization Techniques, in Genetic Algorithms and Evolution Strategies in Engineering and Computer Science, Eds. D. Quagliarella et al., Whiley, (1998)
19. Radcliffe, N. J.: Formal memetic algorithms. In Evolutionary Computing: AISB Workshop, T.C. Fogarthy (ed.), Springer Verlag, LNCS 865, (1994) 1-16
20. Renders J. M., Bersini H.: Hybridizing genetic algorithms with hill-climbing methods for global optimization: Two possible ways. In Proceedings of the First IEEE Conference on Evolutionary Computation, (1994) 312-317
21. Scharf L. L.: Statistical Signal Processing: Detection, Estimation, and Time Series Analysis, Addison Wesley Longman, (1991)
22. Ulder, N.L.J., Aarts, E.H.L., Bandelt, H.-J., Van Laarhoven, P.J.M., Pesch, E.: Improving TSP exchange heuristics by population genetics. Proceedings International Workshop on Parallel Problem Solving from Nature, Dortmund, Germany, (1990) 109-116
23. Van den Branden, C. J., Verscheure, O.: Perceptual Quality Measure Using a Spatio-Temporal Model of Human Visual System, Proc. SPIE, vol. 2668, San Jose, CA, (1996) 450-61
24. van Zon K., Ali W.: Automated Video Chain Optimization. IEEE Transactions of ICCE, **47** (3), (2001) 593-603
25. VQEG, Final Report From the Video Quality Experts Group on the Validation of Objective Models of Video Quality Assessment, March 2000. Available at http://www.its.bldrdoc.gov/vqeg/
26. Whitley, D., Gordon, S., Mathias, K.: Larmarckian Evolution, The Baldwin Effect and Function Optimization. In Proc. Parallel Problem Solving from Nature, PPSN III, (1994) 6-15
27. Wolf, S., Pinson, M.: Video Quality Measurement Techniques. National Telecommunications and Information Administration (NTIA) Report 02-392, US Department of Commerce, (2002)
28. Yagiura, M., Ibaraki, T.: On metaheuristic algorithms for combinatorial optimization problems. Systems and Computers in Japan **32**(3), (2001) 33-55
29. Yen, J., Liao, J. C., Randolph, D., Lee, B.: A hybrid approach to modeling metabolic systems using genetic algorithm and simplex method. In Proceedings of the 11th IEEE Conference on Artificial Intelligence for Applications (CAIA95), Los Angeles, CA, (1995) 277-283

A Broad and Narrow Approach to Interactive Evolutionary Design – An Aircraft Design Example

Oliver Bandte and Sergey Malinchik

Icosystem Corporation, 10 Fawcett Street, Cambridge, Massachusetts 02138
{oliver, sergey}@icosystem.com
http://www.icosystem.com

Abstract. While Interactive Evolutionary Computation (IEC) is starting to penetrate a larger scientific community, only few researchers have applied IEC to the design of complicated artifacts like machines or transportation systems. The present paper introduces a specific approach to Interactive Evolutionary Computation that breaches the two historical categories of user-defined fitness and selection in each generation (narrow) and occasional user-intervention of an automated evolutionary process to correct the fitness function used for (multi-objective) optimization (broad). To highlight the approach, a real world aircraft design problem is employed that demonstrates the relevance and importance of both features for an effective design process.

1 Introduction

Interactive Evolutionary Computation (IEC) has started to capture the fascination of researches from fields as diverse as art, architecture, data mining, geophysics, medicine, psychology, robotics, and sociology. Takagi outlined many of these applications in his overview paper [1]. However, to this day only very few researchers have applied IEC to the problem of engineering and design of complicated artifacts. While the main reason for the slow pace of adoption in engineering is mostly open for speculation, it is partially a result of the field's reluctance to accept new methods, like Genetic Algorithms, as well as the field's already heavy reliance on automated optimization processes that leave decidedly little room for subjectivity. Among the few that did apply IEC to engineering design, Parmee has to be given the adequate credibility for his research. [2], [3]

However, without notable exception, all research currently done in applying IEC to engineering design seems to belong to the category of broad-IEC. Takagi introduced this term in [1], separating the applications of IEC into two categories, *broad* and *narrow*, based on the type of interaction the human has with the evolutionary process. According to that definition, a narrow-IEC uses the human input as the fitness measure for a population member, while the broad-IEC utilizes a numerical fitness value in conjunction with an interface through which the human can guide or interact with the evolutionary process. That means, the broad-IEC can accommodate systems for which we already have a numerical fitness measure, and take advantage of the interactive capability to benefit the optimization procedure. On the other hand,

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 883–895, 2004.

narrow-IEC can be utilized for systems that do not provide a numerical fitness measure, and rather rely on the (subjective) judgment of a human individual or group.

Accordingly, the aforementioned strong reliance on numerical evaluation in engineering design, see [4], [5], [6], [7], resulted in research efforts focusing on broad-IEC techniques. For example, Parmee introduced an Interactive Evolutionary Design System that enables designer interaction after a certain number of generations, presenting the current population and allowing for a redirection of the search through a change in preferences for multiple objectives. [2], [3] All fitness evaluations are performed internal to the evolutionary process, utilizing an analysis tool that calculates multiple system performance values, i.e. objectives, as well as the preference information supplied (interactively) by the designers.

While the reliance of engineers on analysis tools requires interactive evolutionary techniques to utilizes them in the fitness generation, it is also true that many design decisions in practice are made through gut feel and intuition rather then analysis. Recognizing that fact, this paper identifies an IEC approach to design that can be called broad and narrow, allowing for automatic fitness calculation through analysis as well as selection and fitness assignment by the human designers directly.

2 Proposed Process

The approach proposed in this paper is intended to be comprehensive with regard to the implementation into an actual engineering design process. Hence, a process, outlined in Figure 1, was established first that identifies ten steps from problem set-up to deciding on a final design.

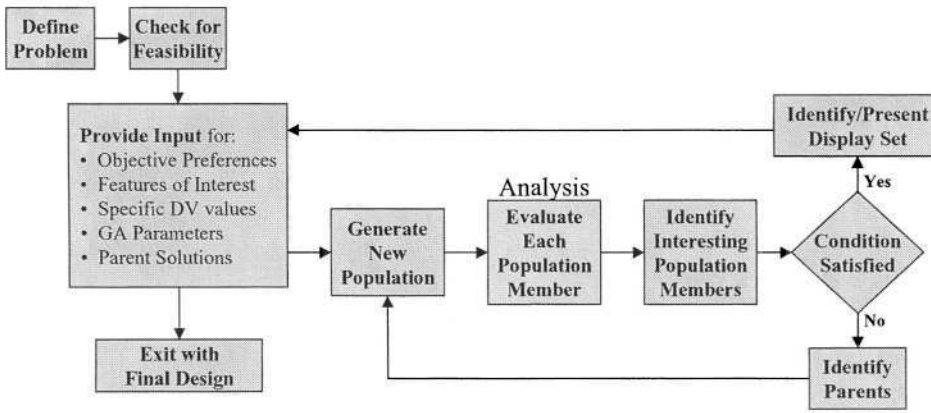


Fig. 1. Integrated Interactive Evolutionary Design Process

Step 1 – Define Problem
As in any design problem, the first step is to define the independent parameters, objectives and constraints, as well as evaluation functions that describe the objectives' dependencies on the independent variables. Additionally, for this interactive

evolutionary design environment this step identifies the genotype representation of a design alternative, the fitness evaluation function, influenced by the objectives, and how to handle design alternatives that violate constraints.

For many design problems the fitness of a solution is based on the satisfaction of multiple objectives. In such cases, a wide variety of approaches to fitness assignment and selection can be employed for this interactive evolutionary design process. Naturally, such established MOGA methods as Goldberg's Pareto ranking [8] or the utility function method with weights representing the relative preferences for the objectives [9] lend themselves as immediate choices. However, without presenting proof here, such techniques as Branke's Focused Pareto Frontiers [10], Deb's Goal Programming adaptation [11], Parmee's Fuzzy Preferences [2,3], or even currently un-proposed uses of TOPSIS [12], LINMAP [13], and MinMax/MaxMin could present a useful solution to the well-known shortcomings of the established methods.

Related to the problem of selecting a fitness assignment approach is the issue of proper treatment of problem constraints. While solutions with good fitness values are desired, they cannot be considered a final solution, as long as they violate the identified constraints. However, in order to enhance the speed with which the genetic algorithm can find good feasible solutions, it may be beneficial to allow for a certain level of infeasibility in other population members. Again, without going into details here, such ideas as inclusion of constraints in the fitness function, fitness discounting or repairing of infeasible alternatives, flat out rejecting infeasible solutions, or treatment of the constraints as objectives in one of the Multi Objective Optimization techniques have been proposed in the past. [14]

Step 2 – Check for Feasibility

The next step entails a check for sufficient feasible design space to search for an interesting optimal solution. While different techniques can be employed here, the simplest approach is to test a large random sample of design variable values, generated from uniform distributions, and to count the number of times constraints were violated. If less then 10% of the sample satisfies all constraints, an effort should be made first to relax the constraints or introduce new technologies to the system, altering the dependency of the constraint functions on the design variable settings. If neither is possible, an interactive evolutionary approach might simply be too expensive for evaluating a small set of possible solutions.

Step 3 – Provide Input

This step represents the central interaction point of the designers with the environment. Here they process the information displayed and communicate preferences for objectives, features of interest in particular designs, whether specific design variable values should be held constant in future iterations, what parameter setting the GA should run with in the next iteration (e.g. a condition that identifies the end of the GA iteration), or whether specific design alternatives should serve as parents for the next generation.

The difficulty in implementing this step is identifying *what information* to display and *how*, so that the designers can easily process it. For example, a general area of research that will benefit this step is the display of multi-dimensional data. Since part of the information conveyed to the designers is data for the objective functions and

design variables, it is guaranteed to be multi-dimensional (higher than three!), for which data visualization is not straightforward.

After analyzing the displayed information, giving feedback is the next critical element in which the designers exert their influence on the optimization process through parent selection for the GA, objective preferences for the automated alternative evaluation procedure, or guiding the search for a design with specific features. Unfortunately from a scientific perspective, what feedback is deemed relevant is highly project and designer experience specific, so that no further guidelines can be given at this point. Hopefully experience gained from future work will yield some insights into some must-haves and must-avoids.

Finally, most actual engineering design work is done by a team of designers (or experts representing different disciplines) rather then one person, hence the issue of differing or opposing opinions among experts/designers needs to be addressed in order for any interactive process to be implemented. Again without proof, several voting and scoring approaches (see Hwang [15]) can be used to accommodate the different opinions and provide a fair framework to generate a collective group opinion.

Step 4 – Generate New Population

Utilizing the information provided by the designers previously, this step simply generates a new population of alternatives. Here the selection of appropriate variation operators (mutation, crossover, weighted average, permutation, or representation specific) becomes important to guarantee the efficiency of the process.

Step 5 – Evaluate Each Population Member

Next, a numerical analysis is employed to distinguish the different designs in the population by their objective and constraint values. In order to provide direct and immediate information to the designers, an *integrated* analysis tool is critical to the success of any "broad" interactive evolutionary design process. Having to wait for a lengthy period of time, just to see the result of a particular action the designers took, would clearly interrupt the dynamic of an interactive environment. While most complex engineering design projects inherently provide such analysis, some of these tools can be very time consuming (e.g. Finite Element Analysis). In many of these cases metamodel analysis (see Mavris [16]) is recommended to facilitate a speedy evaluation during the interactive process, allowing for detailed optimization with the original analysis tool at a later time.

Step 6 – Identify Interesting Population Members

While guaranteeing an efficient search with a genetic algorithm requires a large population size, it seems impractical and unnecessary to subject the designers to the entire (large) population, including the highly dominated alternatives. It is therefore proposed to down-select the population to a more manageable group of interesting solutions. For example, such set of solutions could be the Pareto frontier itself, or alternatives dominated by at most one other population member.

Step 7 – Condition Satisfied
This step simply tests whether the designer-specified condition for ending an iteration is satisfied. This condition can vary from simply reaching the number of generations an iteration was supposed to entail, to reaching a specified fitness or objective function level, to reaching a specified minimum distance level between design alternatives, that guarantees a diverse population. In some cases the designers may elect to execute an iteration with only a few generations, but in general it is anticipated that the GA is supposed to identify significantly improved design alternatives requiring several generations. However, to guarantee continuous involvement of the designers, each iteration should be limited to a reasonable length. Once the condition is satisfied, the set of interesting solutions is (possibly) further reduced to a display set. Otherwise, new parents are being selected for the GA's next generation.

Step 8 – Identify Parents
In this proposed interactive evolutionary design process this selection step closes the loop of the evolutionary process. Specifically, it determines, based on the fitness of each population member and/or the designers' feedback, which solution participates in the creation of the next generation, and which solution has to leave the pool of useful solutions.

Step 9 – Identify and Present "Display Set" of Solutions
For problems with a large number of objectives, it is possible that the set of interesting solutions, e.g. the set of Pareto optimal solutions, is very large and it becomes infeasible to display all interesting solutions. In such cases a further step of down-selecting the alternatives to be displayed is needed. Promising approaches are cluster analysis, displaying a representative alternative that is central with respect to objective or design variable values, as well as simply selecting a fixed number of alternatives with highest fitness to be presented to the designers.

Step 10 – Exit with Final Design
Finally the process ends with the designers simply selecting the solution that satisfies their design requirements best. Naturally, the point at which this step happens is a function of time constraints the design project is subject to.

3 Implementation Example

To demonstrate the interactive evolutionary design approach proposed here, a supersonic business jet was chosen as a complex aerospace design problem for implementation over a simple benchmark problem. The analysis tool used in this example is a system of Response Surface Equations (see Buonanno [17] for details) that allows for a rapid fitness evaluation.

Problem Definition
For this supersonic business jet example, 35 design variables were chosen, separated into five groups as presented in Figure 2. The first group, General, consists of

variables for the vehicle, some of which could be designated Design Requirements. The other four groups contain geometric parameters for the wing, fuselage, empennage, and engine, while the engine group also entails performance parameters relevant to the design.

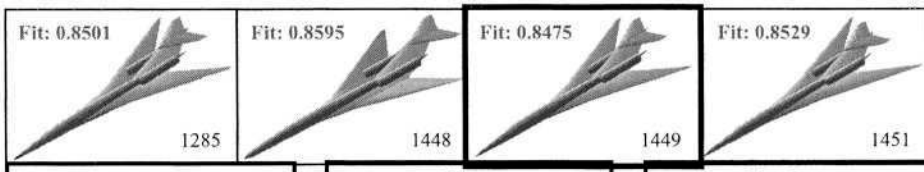
A mix of economic, size, and performance parameters were chosen as the eight objectives to be minimized in this example. Special emphasis is put on noise generation, since it is anticipated to be a primary concern for a supersonic aircraft. Hence, for the initial loop the Boom Loudness and, as a counter weight, the Acquisition Cost are given slightly higher importance of 20%, while all other objectives are set at 10%. Furthermore, certain noise levels could be prohibitively large and prevent the design from getting regulatory approval. Hence, the Sideline and Flyover Noise objectives have to have constraint values imposed on them. In addition to these constraints, the design has to fulfill certain FAA requirements regarding take-off and landing distances as well as approach speed. Furthermore, in order for the design to be physically realizable, the amount of available fuel beyond what is needed to complete the design mission needs to be positive and include a certain amount of reserve. Finally, fitness is calculated via a weighted sum of normalized objective values, penalized by a 20% increase in value whenever at least one constraint is violated. Note that the best solution is identified as the one with the **lowest** fitness. Both, objectives and constraints are tabulated in Figure 3.

Genetic Algorithm

The GA chosen for this example has a population size of 20 and makes use of a real valued 35-gene representation. New generations are created from an elite pool for the two individuals with best (lowest) fitness, and proportionate probabilistic selection for crossover. The crossover algorithm utilizes a strategy with one splice point that is selected at random within the chromosome. Since the design variables are grouped in logical categories, this crossover algorithm enables a complete swap of the engine or fuselage-engine assembly between parents. Parent solutions are being replaced with offspring. Each member of the new population has a 15% probability for mutation at ten random genes, sampling a new value from a uniform distribution over the entire range of design variable values. The GA in this example is used for demonstration purposes only and therefore employs just a small population. A population size of 50 to 100 seems more appropriate for a more involved version of the proposed approach.

Interactive Process

For this implementation example, the GA was interrupted at 80, 160, 240, 400, 560, and finally 800 generations to display the population of design alternatives found to that point in form of pictures with objective values and aircraft configurations. [Due to the page restriction, only a sample display can be presented here in Figures 2 and 3. Please refer to www.icoserver.com/~oliver/GECCO2004 for a complete series of screenshots, or consult [18] for a discussion on the relevance of the display information itself.] Since in this example the GA's population size is so small, no specific concentration on interesting solutions is being performed here, but in order to allow for a reasonable display size of the aircraft configuration, only the four best design alternatives, based on fitness and highest diversity in geometrical features, are being presented in detail on the top of the left-hand side of the display. A screenshot of the displayed chromosome related information is presented in Figure 2,



General

Name	Value	Min	Max
# of PAX	12	8	12
Manuf. ROI	6	6	12
# of Vehicles	500	200	500
Design Range	3595	3500	4200
Mach	1.6	1.6	1.8
TO Thr Der.	0.8475	0.8	1

Engine

Name	Value	Min	Max
Location	101.4	100	110
OPR	25	22	29
TIT	3378	3300	3400
FPR	2.8013	2.6	3.2
Throttle Ratio	1.2146	1.2	1.23
T/W Ratio	0.41	0.41	0.45

Wing

Name	Value	Min	Max
Location	55.79	45	57
AR	2.1748	2	2.5
TR	0.0714	0.05	0.3
Area	2538	2300	3100
Sweep	69.91	67	74
F Str-Bod Int.	0.6024	0.4	0.8
F Str-Wng Int.	0.3096	0.2	0.4
A Str-Bod Int.	0.6	0.4	0.6
A Str-Wng Int.	0.2186	0.2	0.5
TCR - root	0.0416	0.025	0.045
TCR - tip	0.0294	0.025	0.045
Twist – root	1.657	-2	2
Twist - tip	1.1736	0	5

Fuselage

Name	Value	Min	Max
Length	143.82	135	160
Cabin Loc.	36	36	41
Cabin Length	39.752	39	50
Diameter 1	2.3555	2.2	3
Diameter 2	7.3234	7.2	7.6
Diameter 3	7.8997	7.2	8
Diameter 4	7.3745	7.2	7.6
Diameter 5	5.2814	4.5	6.5
Diameter 6	2.3	2.3	3.1

Empennage

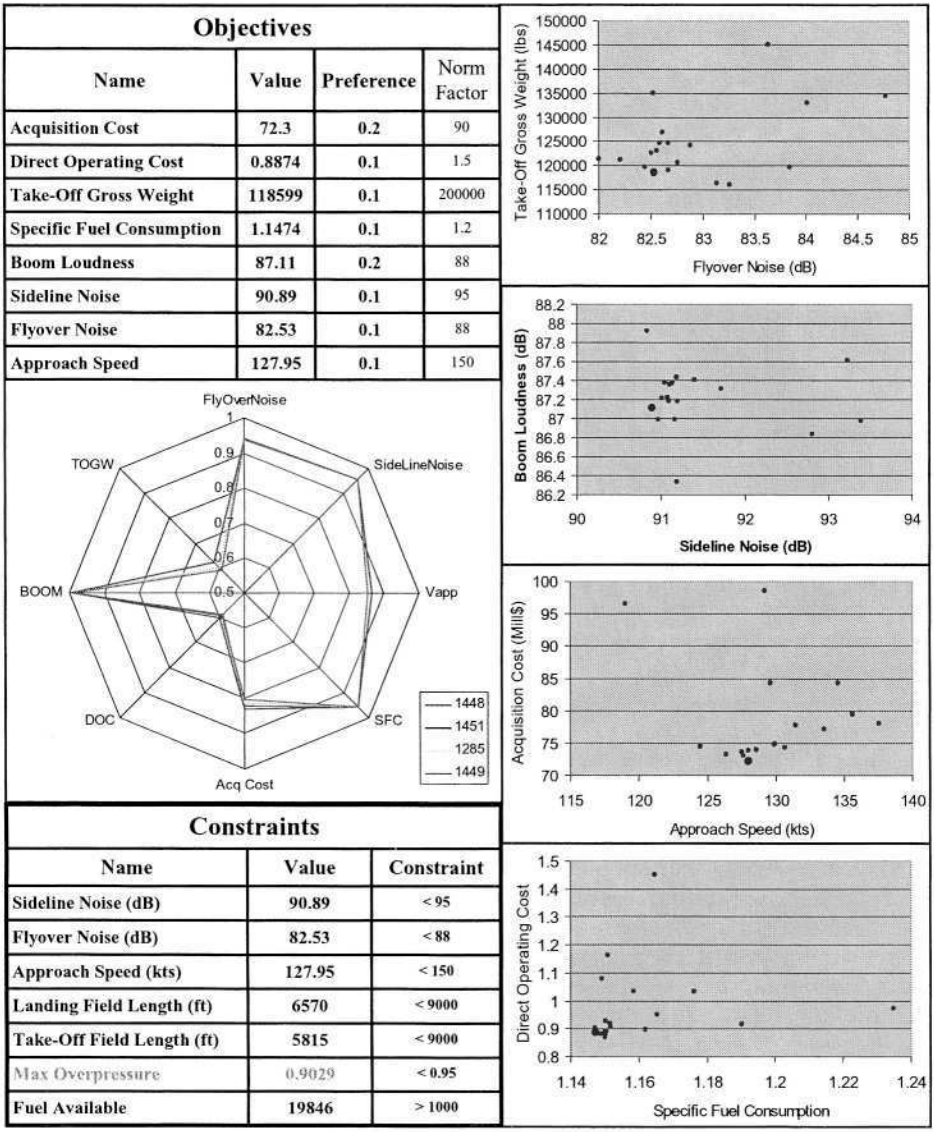
Name	Value	Min	Max
Location	88.556	87	97

Fig. 2. Left-hand Side of Screenshot – Chromosome Information

highlighting the individual with the best/lowest fitness, which is also enlarged on the screen to provide the designers with a more detailed view of the selected configuration (not displayed here). Presented below the larger image are the design variable values for the highlighted alternative, and their respective ranges, representing the actual chromosome of the highlighted alternative.

On the right-hand side of the screen, displayed in Figure 3, the designers can find the objective and constraint information pertaining to the population and the highlighted individual. On the top, a table outlines the specific objective values for the highlighted alternative, as well as the objective preferences and normalization factors used to generate the fitness values for the current population. Below the table, a spider graph compares the four presented alternatives on the basis of their normalized objective values, and to the right four graphs display the objective values for the entire population, identifying its Pareto frontier. Below the spider chart, a table lists the constraint parameter values for the highlighted alternative as well as the respective constraint values. A green font represents constraint parameter values near, orange font right around, and red font way beyond the constraint value.

Based on this information the designers can make some choices with respect to objective preferences and/or selection of features-of-interest. To limit the scope of this example, only the redirection of the search through changes in objective preferences is implemented here. However, as described before and exemplified later, designer selection of features-of-interest is an important part of interactive



Objectives

Name	Value	Preference	Norm Factor
Acquisition Cost	72.3	0.2	90
Direct Operating Cost	0.8874	0.1	1.5
Take-Off Gross Weight	118599	0.1	200000
Specific Fuel Consumption	1.1474	0.1	1.2
Boom Loudness	87.11	0.2	88
Sideline Noise	90.89	0.1	95
Flyover Noise	82.53	0.1	88
Approach Speed	127.95	0.1	150

Constraints

Name	Value	Constraint
Sideline Noise (dB)	90.89	< 95
Flyover Noise (dB)	82.53	< 88
Approach Speed (kts)	127.95	< 150
Landing Field Length (ft)	6570	< 9000
Take-Off Field Length (ft)	5815	< 9000
Max Overpressure	0.9029	< 0.95
Fuel Available	19846	> 1000

Fig. 3. Right-hand Side of Screenshot – Fitness Information

evolutionary design and should not be neglected in general. Also, all designer analysis and feedback is performed by only one individual for this implementation study, leaving the implementation of multiple designer opinions discussed in Step 3 to future research.

The first opportunity for the designers to give feedback about the design alternatives is after 80 generations. From the data displayed it is apparent that all objectives are being satisfied well except for the Boom Loudness. In an attempt to

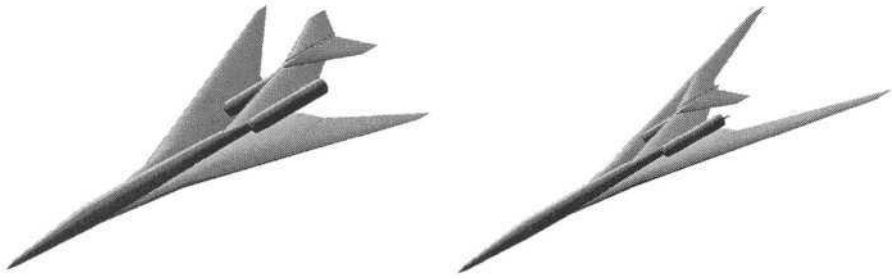


Fig. 4. Best Solutions after 400 and 800 Generations

achieve satisfactory levels for the Boom Loudness in the next iteration, its preference is increased to 30%, reducing the Acquisition Cost's importance to 10%. Note that this feedback/input constitutes a redirection of the search performed by the GA only, and does not constitute selection of a specific parent, component, or design variable value.

After 160 generations, the objective values of the best alternative indicate that the last set of preferences did not emphasize the Boom Loudness and Sideline Noise enough, since Boom Loudness did not improve and the Sideline Noise got worse. Consequently, for the next iteration the importance of both is increased to 35% while all other objectives are reduced to 5%.

After another 80 generations, the resulting population indicates that the last set of preferences still did not emphasize the Boom Loudness enough, since Boom Loudness improved only marginally. On the other hand, Sideline Noise did improve significantly, so that for the next iteration all emphasis can be given to Boom Loudness. To keep the score even for all other objectives, they are being kept at 5% with Boom Loudness at 65%.

As the interactive search progresses and the population of designs is more and more refined, it becomes more and more difficult to find significantly improved alternatives with only a few generations. Hence, 160 generations were executed for the next iteration, for which a very good solution is found in this 400^{th} generation with largely improved values for almost all the objectives. However, in order to test the ability of this interactive evolutionary design approach to support dramatic shifts in search direction, Boom Loudness is now made the primary concern for the next iteration, setting its preference value to 93% while keeping all other objectives at 1%.

After another 160 generations, this preference setting finally produced a supersonic business jet design with a dramatically reduced sonic boom loudness below 85 dB. Unfortunately but not surprisingly, this design yields values for the other objectives that are less desirable, specifically for Sideline Noise. To remedy this shortcoming, another iteration is executed to find a solution that better balances Boom Loudness and Sideline Noise, giving the former 80% and the latter 20% preference. Consequently, all other objectives have a zero preference associated with them.

As anticipated, the new preference setting succeeded in identifying a design that balances Boom Loudness and Sideline Noise at very low values for both. Obviously, this gain does not come without a penalty in the other objectives, their values having significantly increased compared to previous iterations. However, the presented solutions after 400 and 800 generations, see Figure 4, seem to satisfy the objectives

better than the published solution in [17], generated by MATLAB®'s Fmincon function [19]. So, it can be concluded that none of these objective values are dramatically out of sync or range and the presented individual could be considered the final solution. That is, if the information displayed concentrated on the objectives only. However, at this point the display of the aircraft layout becomes critical and some comments have to be made about the unusual design configuration of this solution.

The implementation example for the presented process only considered objective preferences as an input from the designers. A more elaborate example, should also consider feedback for the design variables, affecting the selection process in the GA as much as the search process. This becomes particularly apparent when examining the configurations of the final solutions after 400 and 800 generations in Figure 4. The latter has an unusually long wing extending far beyond the vehicle's fuselage. For all practical purposes, this configuration would not be able to fly, since it would be impossible to land, unless it had a very long landing gear, and very difficult to navigate on the runway. However, the GA cannot be made aware of these issues, since they are not part of the analysis code employed for fitness evaluation. Hence, the GA by itself identifies this solution as the best (after 800 generations).

Fortunately, this weakness can be remedied by incorporating this intuitive designer knowledge into the interactive evolutionary approach via selection of features-of-interest. For example, after generation 560 the visualization already indicated a drastic change in geometry, compared to presented alternatives before, e.g. after generation 400. At this point the designers should have indicated that a configuration of this kind is undesirable by, for example, restricting the sweep and not allowing for the wing tips to be located that far aback.

Summary

To understand the impact of this interactive evolutionary approach on the generated design solutions in this implementation example, a comparison of the time series data for the Fitness, Boom Loudness, Sideline Noise and Acquisition Cost is presented in Figure 5. Note that new designer input was provided after 80, 160, 240, 400, and 560 generations. As the Fitness graph indicates, the GA is successful in reducing Fitness during each iteration, while the value-increases are solely attributable to the change in Fitness composition with every new preference values. The Fitness graph also illustrates a point made earlier that for large numbers of generations it is increasingly difficult for the GA to find better solutions.

The graph for the Boom Loudness value in turn illustrates that the GA was not improving Boom Loudness until its preference was increased to 65% (240+ generations) and only saw significant improvements at a preference of 93% (560+ generations). The Sideline Noise on the other hand saw some very good values between generation numbers 300 and 500. However, the strong pull for better Boom Loudness values increased the Sideline Noise values again, a shortcoming that was only remedied by increasing its preference to 20%. Finally, as indicated in the text, the Acquisition Cost reached good values early on and continued to improve until generation 400, but lost ground when significant preference was given to Boom Loudness and taken away from other objectives.

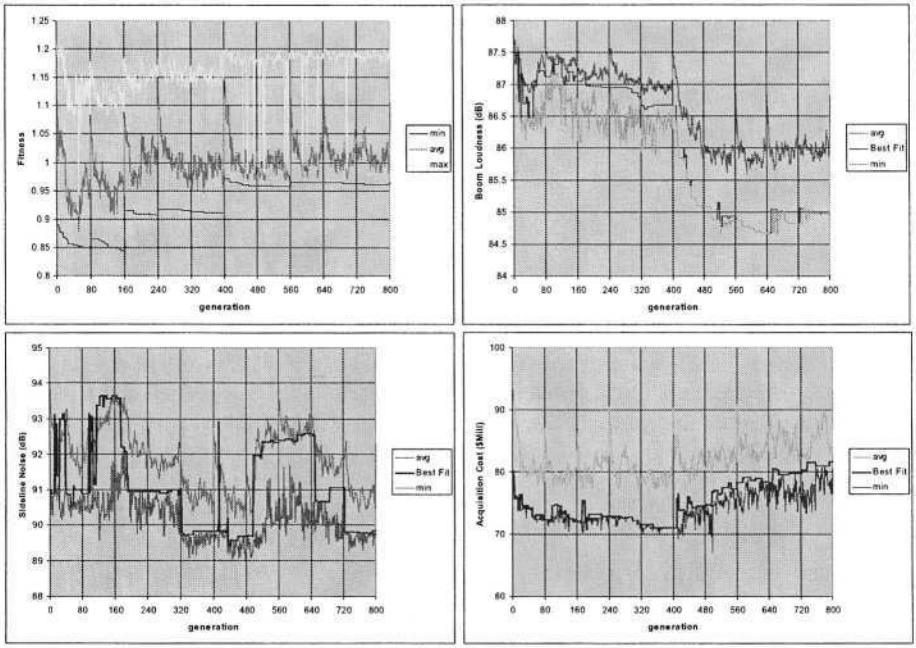


Fig. 5. Summary of Implementation Example

4 Conclusion

While the implementation example presented in this paper does not address all aspects raised in the background discussion of the proposed interactive evolutionary design approach, it is able to demonstrate that this IEC approach to design is feasible and it allows the designers to exert very specific control over the optimization process. Although it is difficult to compare this design approach to traditional techniques by virtue of their solutions, both aircraft design solutions presented in this paper achieve better objective values then a previously published solution just by redirecting the search through changing preferences for objectives that were not satisfied sufficiently by the alternatives presented.

Albeit the approach presented in this paper pushes interactive evolutionary computation toward the world of applied design, more work needs to be done before design practitioners will be able to use an interactive evolutionary design tool. Specifically, the presented example did not implement a multi-expert design feedback, although most complex design problems will draw on a multitude of experts. Hence, it is highly recommended to address this issue in the future. Furthermore, the implemented response surface equation is an easy and straightforward solution to the need for analysis, but it makes the design process susceptible to prediction accuracy. However, a more involved analysis code might increase the computational time to levels that prohibit collaborative interactive evolutionary design altogether. This issue needs further study to recommend different

levels of analysis for different stages of design. Another issue that requires more research to give practitioners some guidance is related to the amount of information communicated to the designers. While large populations are needed for the evolutionary process to be efficient, analyzing more than two dozens designs at the same time may be infeasible for the average design team. Finally, the end-of-iteration condition is currently subject to discussions within the IEC community. Unfortunately, it is not clear at this point whether this condition will also be entirely problem dependent or general guidelines can be provided.

Acknowledgments. The authors would like to extend a special thanks to Dr. Dimitri Mavris of Georgia Tech's Aerospace Systems Design Laboratory for his guidance and advice on the implementation example, Michael Buonanno of ASDL for the supply of the response surface equations as well as Jesse Eyer of ASDL for the generation of the airplane images in RAM. This work was performed under ONR Contract # N00014-03M-0355.

References

1. Takagi, H.: Interactive Evolutionary Computation: Fusion of the Capacities of EC Optimization and Human Evaluation. Proceedings of the IEEE, Vol. 89. No.9 (2001) 1275 -1296
2. Parmee, I. C., Cvetkovic, D., Watson, A., Bonham, C.: Multi-Objective Satisfaction within an Interactive Evolutionary Design Environment, Journal of Evolutionary Computation. Vol. 8. No. 2, MIT Press (2000) 197 - 222
3. Parmee, I. C., Cvetkovic, D., Bonham, C., Packham, I.: Introducing Prototype Interactive Evolutionary Systems for Ill-Defined, Multi-Objective Design Environments, Journal of Advances in Engineering Software. Vol. 32. No. 6. Elsevier (2001) 429 - 441
4. Asimow, M.: Introduction to Design, Prentice-Hall, Englewood Cliffs NJ (1962)
5. Blanchard, B. S., Fabrycky, W. J.: Systems Engineering and Analysis. 3rd edn. Prentice-Hall, New York (1998)
6. Dieter, G. E.: Engineering Design 2nd edn. McGraw-Hill, New York (1991)
7. Dixon, J.R.: Design Engineering: Inventiveness, Analysis, and Decision Making. McGraw-Hill Book Company, New York (1966)
8. Goldberg, D. E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading, MA (1989)
9. Hwang, C.-L., Masud, A. S.: Multiple Objective Decision Making – Methods and Applications. Springer Verlag, Berlin, Heidelberg, New York (1979)
10. Branke, J., Kaussler, T., Schmeck, H.: Guidance in Evolutionary Multi-Objective Optimization. Advances in Engineering Software 32. Elsevier (2001) 499 - 507
11. Deb, K.: Solving Goal Programming Problems Using Multi-Objective Genetic Algorithms, Congress on Evolutionary Computation. Vol. 1, IEEE (1999) 77 - 84
12. Hwang, C.-L., Yoon, K.: Multiple Attribute Decision Making – Methods and Applications. Springer Verlag, Berlin, Heidelberg, New York (1981)
13. Srinivasan, V., Shocker, A. D.: Linear Programming Techniques for Multidimensional Analysis of Preferences. Psychometrika. Vol. 38, No. 3, Psychometric Society (1973) 337 - 369
14. Michalewicz, Z., Fogel, D. B.: How to Solve It: Modern Heuristics. Springer Verlag, Berlin, Heidelberg, New York (1998)

15. Hwang, C.-L., Lin, M.-J.: Group Decision Making under Multiple Criteria– Methods and Applications. Springer Verlag, Berlin, Heidelberg, New York (1987)
16. Mavris, D. N., Bandte, O., Schrage, D. P.,: Application of Probabilistic Methods for the Determination of an Economically Robust HSCT Configuration. AIAA-96-4090. presented at Symposium on Multidisciplinary Analysis and Optimization, Bellevue, WA (1996)
17. Buonanno, M., Lim, C., Mavris, D. N.: Impact of Configuration and Requirements on the Sonic Boom of a Quiet Supersonic Jet. SAE 2002-01-2930. presented at World Aviation Congress, Phoenix, AZ (2002)
18. Bandte, O.: Visualizing Information in an Interactive Evolutionary Design Process. to appear at CEC 2004, Portland, OR (2004)
19. http://www.mathworks.com/access/helpdesk/help/toolbox/optim/optim.shtml

Feature Synthesis Using Genetic Programming for Face Expression Recognition

Bir Bhanu, Jiangang Yu, Xuejun Tan, and Yingqiang Lin

Center for research in intelligent systems
University of California, Riverside CA 92521-0425, USA
{bhanu, jyu, xtan, yqlin}@cris.ucr.edu

Abstract. In this paper a novel genetically-inspired learning method is proposed for face expression recognition (FER) in visible images. Unlike current research for FER that generally uses visually meaningful feature, we proposed a Genetic Programming based technique, which learns to discover composite operators and features that are evolved from combinations of primitive image processing operations. In this approach, the output of the learned composite operator is a feature vector that is used for FER. The experimental results show that our approach can find good composite operators to effectively extract useful features.

1 Introduction

Automatic face expression recognition (FER) is desirable for a variety of applications such as human-computer interaction, human behavior understanding, perceptual user interface, and interactive computer games; hence it is not surprising that automatic facial information processing is an important and highly active subfield of computer vision and pattern recognition researches [1]. In an automatic FER system, face detection or localization in a cluttered scene is usually considered the first step. Next, relevant features from the face must be extracted, and finally the expression can be classified based on the extracted features. Unlike face recognition, FER focuses on how to discern the same expressions from different individuals. Since different people may show the same expression differently, FER problem is more challenging.

People classify FER problem into two categories depending on whether an image sequence is the input or a single image is the input. For image sequence, the dynamic characteristics of expressions are analyzed. Approaches based on static difference are focused on distinguishing the face expressions from a single given image. A review of different approaches for face expression recognition can be found in [2]. In this paper, we discuss FER from static images.

Facial feature extraction attempts to find the most appropriate representation of the face images for recognition. There are mainly two approaches: holistic template matching systems and geometric feature-based systems [3]. In holistic system, after the face image is processed as a whole a template can be acquired as a pixel image or

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 896–907, 2004.

a feature vector. Padgett and Cottrell [4] used seven pixel blocks from feature regions to represent expressions. In geometric feature-based systems, major face components and/or feature points are detected in a face image. The distances between feature points and the relative sizes of the major face components are computed to form a feature vector. The feature points can also form a geometric graph representation of the faces. Feature-based techniques are usually computationally more expensive than template-based techniques, but are more robust to variation in scales, size, head orientation, and location of the face in an image.

2 Related Work and Motivation

2.1 Related Work

As compared to face recognition, there is relatively a small amount of research on facial expression recognition. Previous work on automatic facial expression includes studies using representations based on optical flow, principal components analysis and physically-based models. Yacoob and Davis [5] use the inter-frame motion of edges extracted in the area of the mouth, nose, eyes, and eyebrows. Bartlett et al. [6] use the combination of optical flow and principal components obtained from image differences. Lyons et al. [7] [8] and Zhang et al. [9] [10] use Gabor wavelet coefficients to code face expressions. In their work, they extract a set of geometric facial points on the facial expression images, and then they used multi-scale and multi-orientation Gabor wavelets to filter the images and extract the Gabor wavelet coefficients at the chosen facial points. Similarly, Wiskott et at. [11] use a labeled graph, based on a Gabor wavelet transform, to represent facial expression images. They perform face recognition through elastic graph matching.

2.2 Motivation

Facial feature extraction is the key step in facial expression recognition. For conventional methods, human experts design an approach to detect potential feature in images depending on their knowledge and experience. This approach can often be dissected into some primitive operations on the original image or a set of related feature images obtained from the original one. The experts figure out a smart way to achieve good facial feature representations by combining these primitive operations. The task of finding good composite features is equivalent to finding good points in the composite feature space. The final combination of the primitive operators is called composite operators. It is obvious that human experts can only try some limited number of conventional combinations and explore a very small portion of the composite operator space since they are biased with their knowledge and lower computation capability [14]. GP, however, may try many unconventional ways of combining primitive operations that may never be imagined by a human expert. Although these unconventional combinations are very difficult, if not impossible, to be explained by domain experts, in some cases, it is these unconventional combinations that yield exceptionally good detection/recognition results. In addition,

the inherent parallelism of GP and the high speed of current computers allow the portion of the search space explored by GP to be much larger than that by human experts, enhancing the probability of finding an effective composite operator. The search performed by GP is not a random search. It is guided by the fitness of composite operators in the population. As the search proceeds, GP gradually shifts the population to the portion of the space containing good composite operators. Tan et al. [15] propose a learning algorithm for fingerprint classification based on GP. To the best of our knowledge, unconventional features discovered by the computer are never used in facial expression classification.

3 Technical Approach

3.1 Gabor Filter Bank

The Gabor representation has been shown to be optimal in the sense of minimizing the joint two-dimensional uncertainty in space and frequency [12]. The Gabor filters can be considered as orientation and scale tunable edge and line detectors. The general form of a 2-D Gabor function is given as:

$$g(x,y)=\frac{1}{2\pi\sigma_x\sigma_y}\exp\left[-\frac{1}{2}\left(\frac{x^2}{\sigma_x^2}+\frac{y^2}{\sigma_y^2}\right)+2\pi jWx\right] \tag{1}$$

And its Fourier transform, $G(u, v)$, can be written as:

$$G(\mu,\nu)=\exp\{-\frac{1}{2}[\frac{(\mu-W)^2}{\sigma_u^2}+\frac{\nu^2}{\sigma_v^2}]\} \tag{2}$$

Where (x, y) is the spatial centroid of the elliptical Gaussian window. W is the frequency of a sinusoidal plane wave along the $x-$axis , and σ_x,σ_y are the space constants of the Gaussian envelop along the x and y axes, respectively, u, v are the frequency components in x and y direction, respectively. $\sigma_u=1/2\pi\sigma_X$ and $\sigma_v=1/2\pi\sigma_y$. Gabor function form a complete but nonorthogonal basis set. Expanding a signal using this basis provides a localized frequency description. Let $g(x, y)$ be the mother Gabor wavelet, then filters with multi-orientation can be obtained by a rigid rotation of $g(x,y)$ through the generating function:

$$g(x,y)=a^{-m}G(x',y'),a>1 \tag{3}$$

Where

$$x'=a^{-m}(x\cos\theta+y\sin\theta),\ \text{and}\ y'=a^{-m}(-x\sin\theta+y\cos\theta) \tag{4}$$

And $\theta = n\pi / K$, θ is the rotation angle and K is the total number of orientations. We designed the Gabor filter bank with the following parameters:

$$a = \left(U_h / U_l\right)^{1/S-1}, \sigma_u = \frac{(a-1)U_h}{(a+1)\sqrt{2\ln 2}} \tag{5}$$

$$\sigma_v = \tan\left(\pi / 2K\right)\left[W - \frac{(2\ln 2)\sigma_u^2}{W}\right]\left[2\ln 2 - \frac{(2\ln 2)^2 \sigma_u^2}{W^2}\right]^{\frac{1}{2}} \tag{6}$$

Where $W = a^m U_l$ and $m = 0,1,2,...,S-1$. We define W with the scale factor a^m is to ensure the energy is independent on m. U_h, U_l denote the lower and upper center frequencies of interest, respectively. $n = 0,1,2,...,K-1$. m and n are the indices of scale and orientation, respectively. K is the number of orientations and S is the number of scales. In order to eliminate sensitivity of the filter response to absolute intensity values, the real components of the 2D Gabor filters are biased by adding a constant to make them zero mean. The design strategy is to ensure that the half-peak magnitude support of the filter responses in the frequency spectrum touch each other as shown in Fig.1.

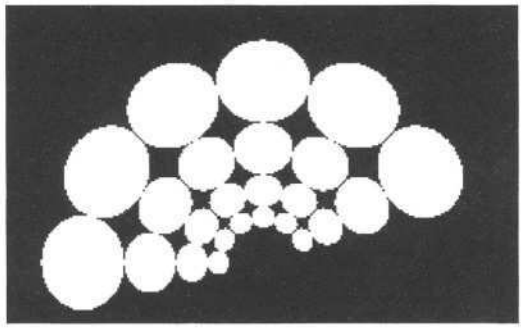


Fig. 1. The filter set in the frequency domain indicates the half-peak magnitude.

3.2 Design Considerations

Figure 2 shows the block diagram of our approach. In our approach, individuals are composite operators represented by binary trees with primitive operators as internal nodes and primitive features as leaf nodes. During the training, GP runs on primitive features generated from the raw facial expression images to generate composite operators. Feature vectors are generated by the learned composite operators, which are used for FER classification. We used Bayesian classifier for classification. During training, fitness value is computed according to the classification result and is monitored during evolution. During testing, the learned best composite operator is

applied directly to generate feature vectors. Since the parameters of Bayesian classifier are determined by the feature vectors from training, the classifier as well the composite operators are learned by GP. Not that, in our approach, we don't need to find any reference point on the image.

- **The Set of Terminals:** The set of terminals used in this paper are called primitive features which is generated from the raw facial expression images filtered by Gabor filter bank at 4 scales and 6 orientations. These 24 images are input to composite operators. For simplicity, we resize the filtered images to 32x32. GP determines which operators are applied on primitive features and how to combine the primitive operators. Figure 3 shows an example of primitive features filtered by Gabor filter bank.
- **The Set of Primitive operators:** A primitive operator takes one or two input images and performs a primitive operation on them and outputs a resultant image and/or feature vectors. In our approach, we designed two kinds of primitive operators: computational operators and feature generation operators. For computational operators, the output are images. For feature generation operators, however, the resultant output includes an image and a real number or vector. The real number or the vectors are the elements of the feature vector, which is used for classification. Table 1 shows different primitive operators and explains the meaning of each one [15].

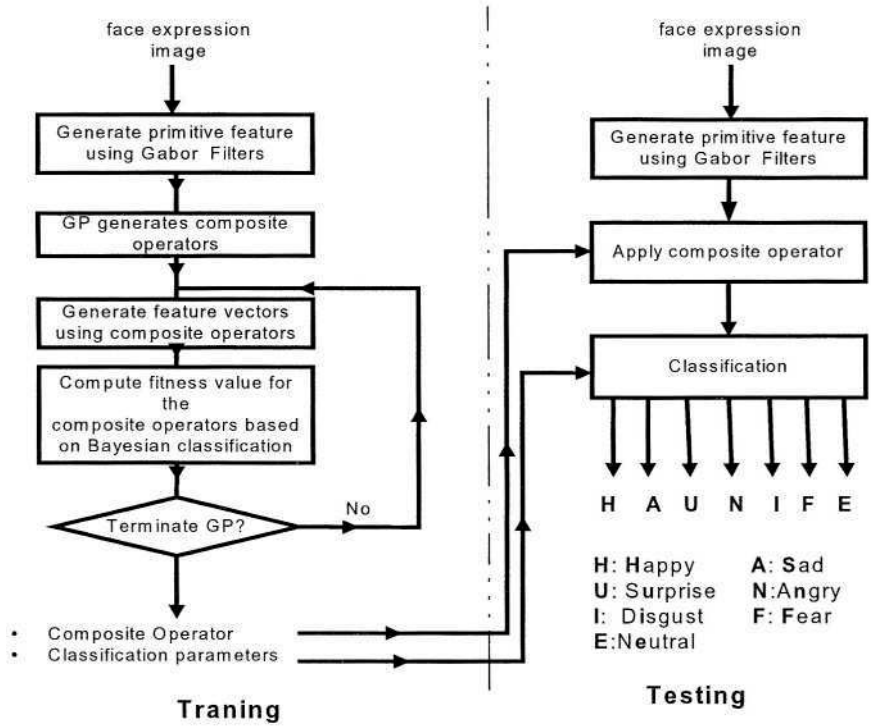


Fig. 2. Block Diagram of our approach

- **The Fitness Value:** During training, at every generation for each composite operator run by GP, we compute the feature vector and estimate the probability distribution function (PDF) for each class using all the available feature vectors for this class. For simplicity, we assume feature vectors for each class have Gaussian distribution. Then, for each class ω_i, we compute the mean and the covariance of this class:

$$u_i = \sum_{j=1,\ldots,N} x_j, \quad \Sigma_i = \frac{1}{N}\sum_j (x_j - u_i)(x_j - u_i)^T \tag{7}$$

Thus, the PDF of ω_i can be written as:

$$p(x \mid \omega_i) = \frac{1}{(2\pi)^{n/2}|\Sigma_i|^{1/2}} \exp\left(-\frac{1}{2}(x-u_i)^T \Sigma_i^{-1}(x-u_i)\right) \tag{8}$$

According to Bayesian theory, we have

$$p(\omega_i \mid x) = \frac{p(x \mid \omega_i)\, p(\omega_i)}{p(x)} \tag{9}$$

We assign $x \in \omega_k$

$$iff\ p(x \mid \omega_k) \cdot p(\omega_k) = \max_{i=1,2,3,4,5,6,7} \left(p(x \mid \omega_i) \cdot p(\omega_i)\right) \tag{10}$$

where n is the size of the feature vector, i is the class and x is a feature vector for the class.
In the classification, the Percentage of Correct Classification (PCC) is used as the fitness value of the composite operator.

$$\text{Fitness Value} = \frac{n_c}{n_s} \times 100\% \tag{11}$$

where n_c is the number of correctly classified facial expressions by GP and n_s is the size of training set.

- **Parameters and Termination conditions:** The parameters to control the run of GP is important. In our approach, we select the maximum size of composite operator 200, population size 100, number of generation 150, crossover rate 0.6, length of maximum feature vector 35, the fitness threshold 0.98, and the mutation rate 0.05. GP stops whenever it finishes the pre-specified number of generations or whenever the best composite operator in the population has fitness value greater than the fitness threshold.

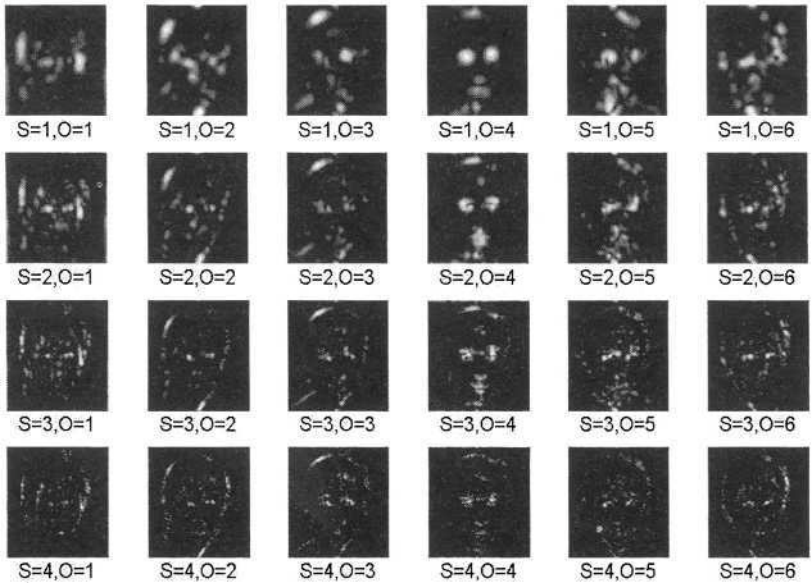


Fig. 3. An example of the primitive feature. S means Scale and O represents Orientation

4 Experimental Results

4.1 Database

The database [7] we use for our experiments contains 213 images of female facial expressions. Each person has two to four images for each of seven expressions: neutral, happy, sad, surprise, anger, disgust, and fear. Each image size is 256x256 pixels. A few examples are shown in Figure 4. This database was also used in [9] [10] [13].

4.2 Results

We perform the experiments 5 times and choose the best result as the learned composite operator. In order to deal with overfitting of this small sample size database, we use 1-fold validation. We divide the database into training set and test set, from which two-third of training data are used to generate composite operators and the remaining one-third is used for evaluating on the tree. Figure 5 shows the fitness values based on the number of the generations in GP. Figure 6 shows the best composite operator for the 7-class classification in LISP notation, which represents the structure of the tree. For 7-class classification, the composite operator's size is 112, out of which there are 19 feature generation operators and the length of the feature vector is 25. Obviously, these composite operators are not easy to be constructed by humans.

Table 1. The primitive operators in our approach

	Primitive Operator	Meaning
Computation Operators	ADD_OP, SUB_OP, MUL_OP and DIV_OP	A+B, A–B, A×B and A/B. If the pixel in B has value 0, the corresponding pixel in A/B takes the maximum pixel value in A.
	MAX2_OP and MIN2_OP	Max (A, B) and min (A, B)
	ADD_CONST_OP, SUB_CONST_OP, MUL_CONST_OP and DIV_CONST_OP	A+c, A-c, A×c and A/c
	SQRT_OP and LOG_OP	$sign\ (A) \times \sqrt{\lvert A \rvert}$ and $sign\ (A) \times \log(\lvert A \rvert)$.
	MAX_OP, MIN_OP, MED_OP, MEAN_OP and STD_OP	Max (A), min (A), med (A), mean (A) and std (A), replace the pixel value by the maximum, minimum, median, mean or standard deviation in a 3×3 block
	BINARY_ZERO_OP and BINARY_MEAN_OP	threshold/binarize A by zero or mean of A
	NEGATIVE_OP	-A
	LEFT_OP, RIGHT_OP, UP_OP and DOWN_OP	Left (A), right (A), up (A) and down (A). Move A to the left, right, up or down by 1 pixel. The border is padded by zeros
	HF_DERIVATIVE_OP and VF_DERIVATIVE_OP	HF (A) and VF (A). Sobel filters along horizontal and vertical directions
Feature Generation Operators	SPE_MAX_OP, SPE_MIN_OP, SPE_MEAN_OP, SPE_ABS_MEAN_OP and SPE_STD_OP	Max2 (A), min2 (A), mean2 (A), mean2 ($\lvert A \rvert$) and std2 (A)
	SPE_U3_OP and SPE_U4_OP	$\mu_3(A)$ and $\mu_4(A)$. Skewness and kurtosis of the histogram of A
	SPE_CENTER_MOMENT11_OP	$\mu_{11}(A)$. First order central moments of A
	SPE_ENTROPY_OP	H (A). Entropy of A
	SPE_MEAN_VECTOR_OP and SPE_STD_VECTOR_OP	mean_vector(A) and std_vector(A). A vector contains the mean or standard deviation value of each row/column of A

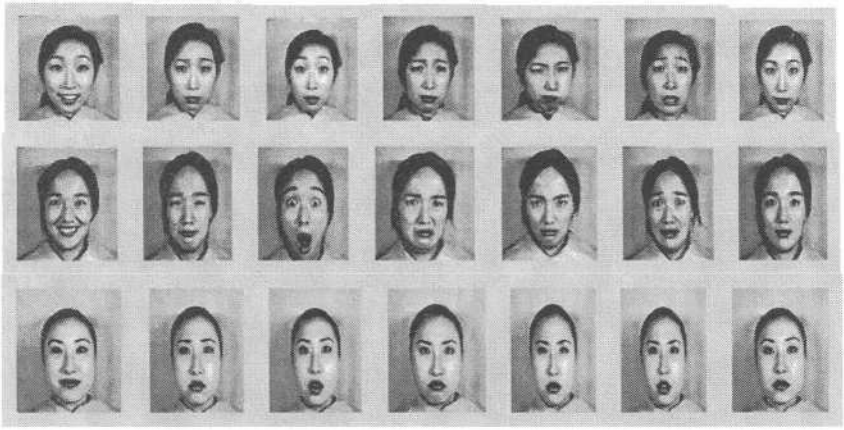


Fig. 4. A few example of the database

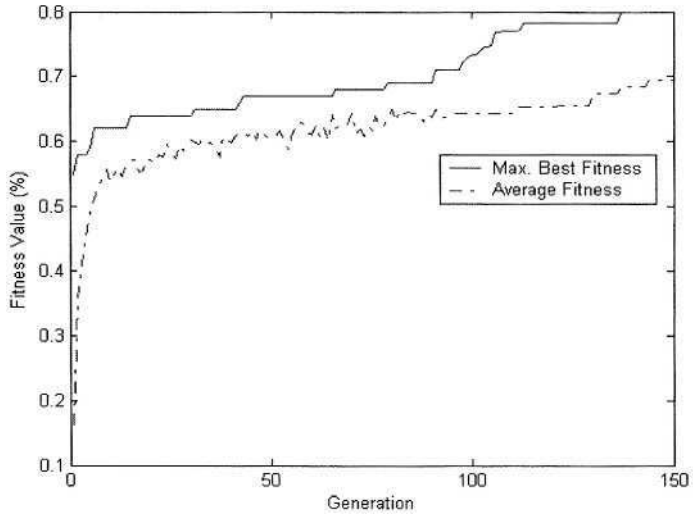


Fig. 5. Fitness value based on the number of generations

4.2.1 Comparison with Previous Approach

In [13], Guo *et al.* used a Bayesian classifier. We compare the result in Table 2.

In Table 2, Bayes All means the Bayes classifier without feature selection, Bayes FS means Bayes with pairwise-greedy feature selection, AdaBoost. From the table, we can find that GP has better performance in both accuracy and length of the feature vector implemented or obtained from [13]. In our approach, we did not do any pre-processing of the raw image. The input image is the raw facial expression image. However, the other methods in Table 2 selected the fiducial points on a face image manully and generated the Gabor coefficients as feature vector.

Table 2. Comparision of the recognition accuracy

	Bayes All [13]	Bayes FS [13]	AdaBoost [13]	GP This paper
Accuracy	63.3%	71.0%	71.9%	72.0%
# Features	612	60	80	25

```
(MEAN_OP(SPE_ABS_MEAN_OP(UP_OP(STDV_OP(SPE_MIN_OP(UP_OP(
ADD_OP(MAX_OP(HF_DERIVATIVE_OP(BINARY_MEAN_OP(HF_DERIV
ATIVE_OP(STDV_OP(MIN2_OP(DIV_OP(INPUT_OP)INPUT_OP))RIGHT_O
P(SPE_MOMENT10_OP(DIV_OP(DIV_CONST_OP(SPE_CENTER_MOMEN
T11_OP(STDV_OP)(ADD_OP(LOG_OP(BINARY_ZERO_OP(ADD_CONST_
OP(DOWN_OP(MAX2_OP(MUL_OP(SPE_MOMENT01_OP(HF_DERIVATI
VE_OP(MUL_OP(BINARY_MEAN_OP(HF_DERIVATIVE_OP(INPUT_OP))I
NPUT_OP))(SPE_MOMENT10_OP)(SPE_MOMENT01_OP(SPE_STD_OP(AD
D_CONST_OP(SPE_MOMENT01_OP(MAX2_OP(SPE_CENTER_MOMENT1
1_OP(MIN_OP(SPE_CENTER_MOMENT11_OP(ADD_OP(VF_DERIVATIVE
_OP(SPE_ABS_MEAN_OP(BINARY_MEAN_OP(MED_OP(INPUT_OP)))))(U
P_OP(MIN2_OP(SUB_OP(INPUT_OP(MAX2_OP(SUB_CONST_OP(INPUT)
OP))(INPUT_OP)))(INPUT_OP)))))))(HF_DERIVATIVE_OP(INPUT_OP))))))))
))))(SPE_STD_OP(MED_OP(SUB_OP(ADD_CONST_OP(INPUT_OP))(LOG_
OP(INPUT_OP)))))))))(HF_DERIVATIVE_OP(HF_DERIVATIVE_OP(MEAN_
OP(LOG_OP(MEAN_OP(SPE_STD_OP(MAX2_OP(BINARY_ZERO_OP(RIG
HT_OP(MIN2_OP(SPE_ABS_MEAN_OP(MED_OP(UP_OP(ADD_OP(INPUT
_OP)(INPUT_OP)))))SPE_ABS_MEAN_OP(MEAN_OP(DIV_OP(INPUT_OP)(
MAX_OP(LOG_OP(DOWN_OP(SPE_MOMENT01_OP(ADD_OP(INPUT_OP)
(INPUT_OP))))))))))(INPUT_OP))))))))))))))(ADD_CONST_OP(SPE_CENT
ER_MOMENT11_OP(MIN2_OP(UP_OP(SPE_MAX_OP(MIN_OP(MIN_OP(M
EAN_OP(ADD_CONST_OP(INPUT_OP)))))))(BINARY_MEAN_OP(ADD_OP
(INPUT_OP)(INPUT_OP))))))))))))
```

Fig. 6. Learned Composite Operators in lisp notation

4.3 Discussion

In [9] [10] [13], authors have used SVM, LDA and Neural Network for facial expression recognition. We found SVM and LDA have higher recognition accuracy (about 90%),while in [13] and our GP the perfomances with using Bayesian classifier are much less than that. In the following we present an analysis for this difference.

For a Bayesian classifier, we need to estimate the probability distribution function (PDF) $p(x \mid \omega_i)$ for each class ω_i. In the small sample case, it is hard that the estimated PDF can accurately approximate the underlying unknown densities. Thus the estimated probability distribution may be biased far away from the real one. As a consequence, low recognition accuracy can be expected. In our approach, the simplified Bayesian theory assumes independent and Gaussian distribution, which simplified the problem of class density estimation. However, in our experiments, we found the problem of overfitting is serious. However, for the margin-based discrimination, one doesn't need to estimate the underlying distribution, thus the recognition accuracy could be higher. For the future, we plan to perform experiments with SVM based classifier and with GP generated features.

5 Conclusions

In this paper, we proposed a learning paradigm for facial expression recognition based on GP. Compared wilh the previous work with the same classifier, our experimental results show that GP can find good composite operators. Our GP-based algorithm is effective in extracting feature vectors for classification, which are beyond human's imagination. In our approach, we don't need to do any pre-processing of the raw image and we don't need to find any reference points on the face.

References

1. J. Daugman, "Face and Gesture Recognition: An Overview," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 19, no. 7, pp. 675-676, July 1997
2. M. Pantic and L. J. M. Rothkrantz, Automatic analysis of facial expressions: The state of the art, *IEEE Trans. Pattern Analysis and Machine Intelligence,* 22(12), 1424-1445, 2000
3. W. Zhao, R. Chellappa, A. Rosenfeld, and P. J. Phillips, Face recognition: A literature survey. *CVL Technical Report, University of Maryland, October 2000*
4. C. Padgett and G. Cottrell, Identifying emotion in static images, Proc. 2nd Joint Symp. On *Neural Computation,* vol. 5, 91-101, 1997
5. Y. Yacoob and L. Davis. Recognizing facial expressions by spatio-temporal analysis. *In Proceedings of the International Conference on Pattern Recognition,* volume 1, pages 747-749, Oct. 1994.
6. M. Bartlett, P. Viola, T. Sejnowski, L. Larsen, J. Hager, and P. Ekman. Classifying facial action. In D. Touretzky, M. Mozer, and M. Hasselmo, editors, *Advances in Neural Information Processing Systems* 8. MIT Press, Cambridge, MA, 1996

7. M. J. lyons, S. Akamatsu, M. Kamachi, and J. Gyoba, Coding facial expressions with Gabor wavelets. In Proc. *Third IEEE Int. Conf. Automatic Face and Gesture Recognition,* 200-205, 1998
8. M. J. Lyons, J. Budynek, and S. Akamatsu, Automatic Classification of single facial images, *IEEE Trans. Pattern Analysis and Machine Intelligence,* 21(12), 1357-1362, 1999
9. Z. Zhang, M. Lyons, M. Schuster, and S. Akamatsu, Comparison between geometry-based and Gabor-wavelets-based facial expression recognition using multi-layer perceptron, *Proc. Int'l Conf. Automatic Face and Gesture Recognition,* pp. 454-459, 1998
10. Z. Zhang, Feature-based facial expression recognition: Sensitivity analysis and experiments with a multi-layer perceptron, *Journal of Pattern Recognition and Artificial Intelligence,* 13(6): 893-911,1999
11. L. Wiskott, J. M. Fellous, N. Kruger, and C. Von der Malsburg. Face recognition by bunch graph matching. *IEEE Trans. Pattern Analysis and Machine Intelligence,* 19(7): 775-779,July 1997
12. B. S. Manjunath and W. Y. Ma, Texture features for browsing and retrieval of image data. *IEEE Trans. Pattern Analysis and Machine Intelligence,* 18(8), 837-842, 1996
13. G. D. Guo and C. R. Dyer, Simultaneous Feature Selection and Classifier Training via Linear Programming: A Case Study for Face Expression Recognition, *IEEE Conference on Computer Vision and Pattern Recognition,* I, 346-352, June, 2003
14. B. Bhanu and Y. Lin, Learning Composite Operators for Object Detection, *GECCO 2002,* pp. 1003-1010, 2002
15. X. Tan, B. Bhanu, and Y. Lin, Learning Features for Fingerprint Classification, *Audio- and Video-based Biometric Person Authentication 2003,* pp. 319-326, 2003

An Enhanced Genetic Algorithm for DNA Sequencing by Hybridization with Positive and Negative Errors

Thang N. Bui and Waleed A. Youssef

Department of Computer Science
The Pennsylvania State University at Harrisburg
Middletown, PA 17057
{tbui,wuy101}@psu.edu

Abstract. This paper describes a genetic algorithm for the DNA sequencing problem. The algorithm allows the input spectrum to contain both positive and negative errors as could be expected from a hybridization experiment. The main features of the algorithm include a preprocessing step that reduces the size of the input spectrum and an efficient local optimization. In experimental tests, the algorithm performed very well against existing algorithms. The algorithm also performed very well on a large data set generated in this paper from real genomes data.

1 Introduction

Determining the genome of living organisms has been a major research initiative world wide in the last few years. One of the principal steps in this endeavor is the sequencing of DNA. Informally, DNA sequencing is the process of determining the correct order of nucleotides in a DNA segment. Many techniques have been developed for DNA sequencing. DNA sequencing experiments are typically performed in two stages: shotgun sequencing and walking. In shotgun sequencing, many short, randomly selected fragments of a DNA segment are sequenced. Due to the stochastic nature of this process, there are parts of the DNA segment that are left unsequenced or insufficiently covered. These parts are then covered by a deterministic finishing process called walking [19].

The two most popular methods for DNA sequencing are the Sanger method and the Sequencing by Hybridization (SBH) method [20]. In this paper we consider only the SBH method. SBH follows the methodology known as "break, read and assemble". In this methodology a DNA sequence is partitioned into smaller size fragments. The fragments are then read using a fluorescent light. The assemble phase tries to retrieve the original sequence from the shorter length fragments, i.e., to determine the exact sequence of nucleotides of the DNA molecule.

From an algorithmic point of view the DNA sequencing problem is the problem of constructing a chromosome that most likely contain all the DNA fragments in a given input set, called the *spectrum.* The spectrum is usually obtained through some experiments such as a hybridization experiment. It should

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 908–919, 2004.

be noted that the fragments in a spectrum have overlaps and all fragments have the same length. If a spectrum contains all possible fragments of length l of a DNA sequence and there are no errors in the fragments then there exist efficient algorithms for reconstructing the original DNA sequence from the spectrum. In general, however, there are errors in the spectrum, e.g., missing fragments or erroneous fragments, making the problem of reconstructing the original DNA sequence an NP-hard problem [4].

In this paper we present a genetic algorithm for the DNA sequencing problem. Our algorithm differs from others in that it can efficiently handle different types of errors in the input. Additionally, our algorithm includes a preprocessing step that effectively reduces the size of the input, thereby helps reduce the running time of the algorithm. The idea of the preprocessing step can be extended to create a hierarchical structure that enables our algorithm to deal with much longer sequence. Experimental results show that our algorithm outperformed other algorithms from [1] and [7]. We also performed extensive test of our algorithm on data that we generated systematically from genomes obtained from the GenBank (www.ncbi.nlm.nih.gov/Genbank/). To determine the quality of these results we used the Smith-Waterman algorithm for sequence alignment [24]. The results of these experiments show that our algorithm is very robust against a large range of errors.

The rest of the paper is organized as follows. Section 2 describes some common terminologies, defines the problem formally and lists some current work on the problem. The algorithm is described in Section 3. Experimental results comparing the performance of our algorithm against others are given in Section 4. Section 4 also includes results showing the performance of our algorithm on a large data set that we generated. The conclusion and future directions are given in Section 5.

2 Preliminaries

In this section we describe some of the terminologies needed for the rest of the paper, give a formal description of the DNA sequencing problem and list some of the existing works that have been done on this problem. DNA (deoxyribonucleic acid) consists of two strands, each of which contains *nucleotides:* adenine (A), cytosine (C), guanine (G) and thymine (T). (Technically, there are other components in a DNA strand such as phosphates.) The nucleotides in each strand are connected together in series. The two strands of the DNA are twisted together into the famous double helix structure. Furthermore, each nucleotide in a strand is connected to a *complementary* nucleotide in the other strand, where A is paired with T and C is paired with G. Thus, each strand in a DNA completely determines the other.

A *fragment* is a short sequence of nucleotides. It is also known as an *oligonucleotide.* A *hybridization experiment* is an experiment that takes a DNA strand and produces copy of fragments of that strand. These fragments usually have overlap. The set of all fragments that result from a hybridization experiment is

known as a *spectrum.* All fragments in a spectrum have the same length. In this paper, we consider fragment lengths ranging from 10 to 50.

The DNA sequencing problem is the problem of determining a DNA strand based on a given spectrum. We can model this problem as follows. Let $\Sigma = \{A, C, T, G\}$ be an alphabet. Here we consider a spectrum as a set of distinct strings of length l over Σ. We refer to each string in a spectrum as a *fragment.* A spectrum is said to be *ideal* if the following condition is true for all but one fragment in the spectrum: the suffix of length $l-1$ in a fragment is a prefix of exactly one other fragment in the spectrum. The DNA sequencing problem can then be stated as the problem of constructing a string over Σ from a given spectrum (not necessarily an ideal spectrum), so that the resulting string is the shortest string that contains as many of the fragments in the spectrum as possible.

In general the input spectrum is not an ideal one. The errors appearing in a spectrum are usually due to errors in the hybridization experiment. Errors can be classified as positive or negative. The spectrum has *positive errors* when it contains fragments that are not part of the original sequence. It has *negative errors* when it fails to contain some oligonucleotides. Certain errors are *random,* meaning that they may disappear when the experiment is repeated. However, many hybridization errors are *systematic,* meaning that they are likely to repeat each time the experiment is run [22] [23].

If there are no errors, the problem of DNA sequencing is similar to the Shortest Superstring problem [20], which is defined as the problem of reconstructing a string given a collection of overlapped substrings. The Shortest Superstring problem is NP-hard, but it is known that greedy algorithms work well for this problem [12]. There exists an approximation algorithm with an approximation factor of three, i.e., the Superstring it produces is at most three times as long as the optimal shortest Superstring [8]. However, compared to the DNA Sequencing problem, the Shortest Superstring problem seems to be easier.

The existence of errors in the input spectrum makes the problem of reconstructing the original sequence an NP-hard problem [4]. Missing fragments from the experiment turn the problem into the problem of finding the Most-Likely sequence [4]. The most likely sequence is the shortest one containing almost each fragment as a substring. Some fragments might be excluded from the final result. Those excluded fragments are the ones that represent the positive errors in the experiment. Also, fragments might not be completely overlapped. Under normal situations, two fragments of length l intersect in $l-1$ positions. However, because of negative errors the longest overlap might not be of length $l-1$. Another source of difficulty exists when the spectrum contains repeated fragments. Most existing algorithms that allow for errors in the input spectrum put restrictions on the error model [5] [10] [11]. There are few algorithms that do not have any restriction on the input error model. Two such algorithms are in [1] and [7], and our algorithm is compared against them.

Algorithms solving the DNA Sequencing problem take as input the spectrum of all fragments. The output is a DNA sequence that is the most-likely one that

includes all fragments. In the case of an ideal spectrum, the result sequence would be of length $n+l-1$, where n is the number of fragments in the spectrum, l is the length of the fragments. However, because of errors this may not be always the case. The algorithm presented here deals with errors. So the output sequence would not necessarily be of length $n+l-1$. Negative errors may cause the output sequence to be shorter in length. Positive errors may cause it to be longer. In some other cases, those types of errors would mistakenly cause the algorithm to converge to a sequence that does not necessarily represent the optimal solution.

3 Algorithm

In this section we describe a genetic algorithm for solving the DNA sequencing problem when the input may have both positive and negative errors. We do not require that the starting fragment of the sequence be known as it is done in [6]. We use a steady state genetic algorithm together with a local optimization procedure to help improve the performance of the algorithm. Additionally, we have a preprocessing step that helps improve the algorithm even further. The overall algorithm is given in Figure 1. In the following subsections we give more details of the algorithm.

```
Sequence(S)   // S is a spectrum
  preprocess(S)
  generate a random initial population P
  for each a ∈ P
     LocalOptimize(a)
  endfor
  repeat
     Select two parents p1 and p2
     u ⟵ crossover(p1, p2)
     LocalOptimize(u)
     mutate(u)
     replace(u, p1, p2, P)
  until (there is no improvement)
  return the best member of P
  Align output sequence;
```

Fig. 1. The Enhanced Genetic Algorithm for DNA sequencing

Preprocessing. In general, the fewer fragments and longer fragments there are in the spectrum the easier the problem is. The idea of the preprocessing is to merge certain fragments together, thereby creating a new spectrum that has fewer and longer fragments. In this step we create long chains of fragments of the

form $F_1 \ldots F_k$, where F_i's are fragments, and the last $l-1$ elements of F_i match the first $l-1$ elements of F_{i+1}. Our objective is to make k as large as possible. The optimal case would be when $k = n$, where n is the cardinality of spectrum S. However, because of negative errors, that's not always the case. Each such chain of fragments is merged into one fragment in the new spectrum. The algorithm then works with this spectrum which has variable length fragments and a smaller number of fragments than the original spectrum.

The preprocessing algorithm creates a chain by selecting an unused fragment in the spectrum and adding it to the chain. The fragment is then marked used. The algorithm extends the chain by selecting an unused fragment that has an overlap of $l-1$ with the last fragment in the chain. If such a fragment exists, it is added to the chain and marked used, and the process is repeated. If there is no such fragment the chain is terminated. The algorithm then starts a new chain. The algorithm terminates when all fragments in the original spectrum have been used. The algorithm then merges the fragments in each chain to create a fragment for the new spectrum. This algorithm can be efficiently implemented using dynamic programming technique.

As an example, suppose we have a DNA sequence CTAGACGTTC of length 10. An ideal spectrum would consist of the following six fragments: CTAGA, TAGAC, AGACG, GACGT, ACGTT and CGTTC, where we have assumed that the fragment length is 5. However, because of errors from the hybridization experiment an input spectrum in this case may consists of the following six fragments: CTAGA, TAGAC, AGACG, TATCC, ACGTT, CGTTC. This spectrum differs from the ideal spectrum in that it does not contain the fragment GACGT (a negative error), instead it contains the fragment TATCC which is not a substring of the original DNA sequence (a positive error). Thus, this spectrum has one negative error and one positive error. Using this spectrum as the input, the preprocessing algorithm would produce the following chains [CTAGA, TAGAC, AGAC], [TATCC], and [ACGTT, CGTTC], which yields the spectrum consisting of the following three fragments: CTAGACG, TATCC, ACGTTC.

Encoding and Initialization. We assume that fragments in the spectrum obtained from the preprocessing step are indexed in some order. Each member of the population is a vector of fragment indexes representing a possible solution sequence to the problem. Given a vector $u[1 \ldots m]$ of fragment indexes, the corresponding sequence is obtained by merging the fragments $F_{u[1]}, F_{u[2]}, \ldots, F_{u[m]}$ in that order, where F_i is the ith fragment in the spectrum. Here, two adjacent fragments are put together by overlapping them as much as possible. We also maintain the constraint that each fragment index appears at most once in the encoding of a sequence. The vectors in the population can be of variable length. In what follows, we refer to each member of the population as a vector or sequence.

An initial population of size 120 is generated at random. The size of the population remains constant throughout the algorithm. The vectors in the initial population all have the same size, i.e., each vector has the same number of

fragment indexes. However, since the fragments are of variable length after the preprocessing step, the corresponding sequences have different lengths. Suppose the spectrum obtained after preprocessing contains the fragments CTAGACG, TATCC, ACGTT, and the fragments are indexed in that ordered from 1 to 3. Then, the vector (1,3) yields the sequence CTAGACGTT, and the vector (2,1) yields the sequence TATCCTAGACG.

Fitness. The fitness of each sequence is calculated based on two factors: (i) the amount of overlap between adjacent fragments in the sequence, and (ii) the length of the sequence. The idea here is that the more overlap there are between adjacent fragments the shorter the sequence is. Also, if the length of the sequence is equal to $n+l-1$, where n is the cardinality of the spectrum and l is the length of each fragment, a bonus value is added to the value of the fitness. Note that $n+l-1$ is the optimal length for a sequence that includes all fragments in the spectrum. More formally, let $u[1 \ldots k]$ be a vector representing a sequence U in the population. The fitness of U is defined as follows.

$$f(U) = \sum_{i=1}^{k-1} |F_{u[i]} \cap F_{u[i+1]}| + z$$

where z is the bonus value defined by

$$z = \begin{cases} s, & \text{if } |U| = n+l-1, \\ s/||U| - (n+l-1)|, & \text{otherwise.} \end{cases}$$

For our experiment, s was set to 100. The fitness can be computed efficiently using dynamic programming technique.

Parent Selection. The parents are selected using the standard proportional selection method where sequences that have higher fitness have a better chance of being selected. The standard roulette wheel scheme is used in our algorithm [13].

Crossover. We use a 3-point crossover in our algorithm. The offspring is constructed by selecting alternately from each parent after 3 cutpoints have been determined. Note that the members of the population are vectors of fragment indexes. This process may create offsprings that contain duplicated fragment indexes. A repair algorithm is used to get rid of any repeated fragments and to ensure that each fragment appears at most once within the offspring. The repair algorithm works by replacing the repeated fragments with fragments from the spectrum that are not currently in use by the sequence.

Mutation. The mutation is performed on each newly created offspring sequence as follows. With a probability of 10% each fragment index in the offspring is swapped with another randomly selected fragment index.

Local Optimization. The local optimization algorithm has two steps. The first step is to scan the sequence sequentially and identify a pair of adjacent fragments, say x and y, that has the smallest overlap. Find the fragment, say z, that has the highest overlap with x. Replace y with z. The vector is then repaired, if needed, to eliminate duplicated fragments.

The second step is to rearrange the fragments in the sequence in the hope of improving its fitness value. This is done by first finding the two pairs of adjacent fragments that have the two smallest overlaps. Let s, t be the first pair and x, y be the second pair. That is, assume that the vector $u = (a, \ldots, s, t, \ldots, x, y, \ldots z)$. We then construct a new vector u' by swapping the fragments between t and x with the fragments from y to the end of u. Thus, $u' = (a, \ldots, s, y, \ldots, z, t, \ldots, x)$. If the fitness of u' is better than that of u, we replace u by u'. Otherwise, we keep u and discard u'. By using dynamic programming technique the local optimization algorithm and the repair algorithm can be efficiently implemented.

Replacement Scheme. If the fitness of the new offspring is larger than the fitness of the worse of the two parents, then we replace that parent with the new offspring. Otherwise, we discard the new offspring.

Stopping Condition. The algorithm terminates if there is no improvement in the total fitness of the population in 400 consecutive generations, or if the number of generations exceeds 50,000.

4 Experimental Results

In this section we first describe the performance of our algorithm in comparison with some existing algorithms for the DNA sequencing problem. We then show the result of our algorithm on an extensive set of data generated by us using genome sequences from the GenBank (www.ncbi.nlm.nih.gov/Genbank/). Our algorithm was implemented in C++ and was run on a PC with Pentium IV 2.4GHz Intel processor with 512MB of RAM.

Our first set of test data is from [7] [15]. We used it to compare our algorithm against the Hybrid Genetic Algorithm of [7] and the Tabu Search algorithm in [1]. The data from this set consists of spectra having 100, 200, 300, 400 and 500 fragments. There are 40 spectra for each size, for a total of 200 instances. The fragment length in all of these instances is 10. In each instance there are 20% positive errors and 20% negative errors.

To determine the quality of the solution we follow [7] and use the classical pairwise Smith-Waterman sequence alignment algorithm to compare the solution output by the algorithm with the known sequence from which the spectrum was derived. We use two values from the output of the Smith-Waterman algorithm: the match percentage and the similarity score. In addition, as in [7], for each instance tested we include the number of times the algorithm finds the optimal answer. This number is called the optimum number. Table 1 and Figure 2 summarize the results of the comparison. It can be seen that our algorithm performs

significantly better than the other algorithms as the sequence length gets larger. More details can be seen in Figures 2 (a), (b) and (c).

Even though the running times are available for all algorithms, we cannot compare the running times, since different machines were used. For our algorithm, the average running time was from 0.6 second for the smallest problem size to 15.1 seconds for the largest problem size tested. The Hybrid GA and Tabu Search algorithms were run on a PC with a Pentium II 300MHz processor and 256MB of RAM. The average running times range from 13.5 seconds to 437.9 seconds for the Hybrid GA and from 14.1 seconds to 471.5 seconds for the Tabu Search algorithm. More details can be found in Table 1.

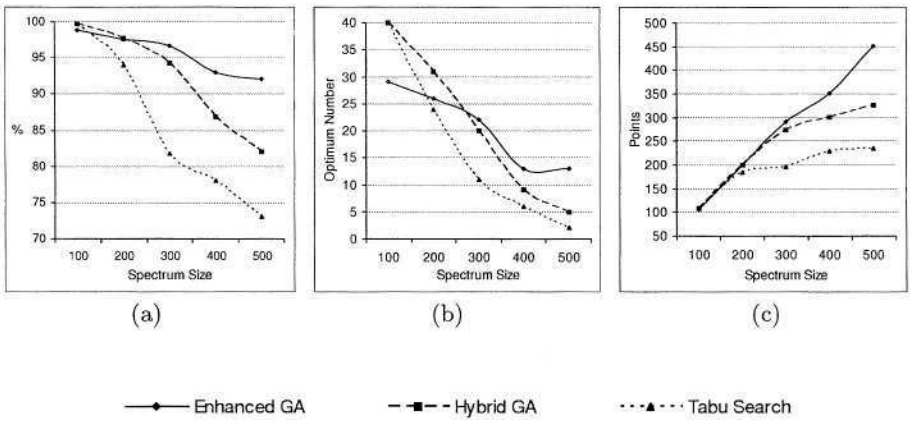


Fig. 2. Comparison between our algorithm Enhanced GA, Hybrid GA and Tabu Search: (a) Match Percentage, (b) Optimum Number, and (c) Similarity Score

The second set of test data we used was generated by us using three genomes obtained from the GenBank. Table 2 shows the details of the genomes that we used. For each of the three genomes we generated 10 sequences of each length in the set {100, 0200, 300, 400, 500, 1000, 2000}. That is, for each genome we generated 70 sequences. From each of these sequences we generated spectra with fragments of length 10 and 20. For each fragment length, we generated spectra with 13 different combinations of positive and negative errors, ranging from 0 to 20% errors. Hence, for all three genomes we generated a total of $3 \times 70 \times 2 \times 13 = 5{,}460$ spectra. For each input spectrum the algorithm was run 50 times. We used the Mersenne Twister random number generator of [17] in our algorithm.

We have also generated a similar set of spectra with fragments of length 50. The algorithm was tested on all spectra of three different lengths: 10, 20, and 50. We observe that the longer the fragments are, the better the results are. In fact, with fragment length of 50, our algorithm almost always found the optimal answers, and thus, we do not include the data for fragments of length 50 here. We used different fragment lengths since in practice different hybridization

Table 1. Summary of results by Enhanced GA, Hybrid GA and Tabu Search

Algorithm		100	200	300	400	500
Enhanced GA	Average Similarity Score (pt)	106.7	200.6	291.6	352.2	451.6
(Pentium IV,	Average Similarity Score (%)	98.9	97.6	96.7	92.9	92.0
2.4GHz,	Running time (sec)	0.6	1.5	3.6	8.6	15.1
512MB RAM)	Optimum no.	29	26	22	13	13
HGA	Average Similarity Score (pt)	108.4	199.3	274.1	301.7	326.0
(Pentium II,	Average Similarity Score (%)	99.7	97.7	94.3	86.9	82.0
300MHz,	Running time (sec)	13.5	63.4	154.9	263.4	437.9
256MB RAM)	Optimum no.	40	31	20	9	5
Tabu Search	Average Similarity Score (pt)	108.4	184.1	196.6	229.5	235.1
(Pentium II,	Average Similarity Score (%)	99.7	94.0	81.8	78.1	73.1
300MHz,	Running time (sec)	14.1	60.8	177.7	258.3	471.5
256MB RAM)	Optimum no.	40	24	11	6	2

Table 2. Genomes from the GenBank used in testing the Genetic Algorithm

Sequence	Length(bp)
Human immunodeficiency virus 2	10,359
Drosophila melanogaster DNA sequence of white locus	14,245
Canis familiaris clone RP81-60B6	165,116

techniques may require different fragment lengths. Normally, hybridization rate is better if the fragment length is larger. However, for *in situ* hybridization small fragment length is required [18].

As in the case of the first data set, we use the Smith-Waterman sequence alignment algorithm to determine the quality of the solutions returned by the algorithm. We used an implementation of the Smith-Waterman algorithm provided by Jie Li of the Iowa State University [16]. Figures 3 and 4 show the performance and running time of our algorithm on the second set of data. In Figure 3, the left graph shows the match percentage for spectra with fragment length 10, and the right graph is for spectra with fragment length 20. The x-axis shows the various error combinations in the input spectrum. The notation -a+b indicates spectra with a% negative error and %b positive error. For each error combination, the match percentage shown is the average of the match percentages taken from all spectra generated from the three genomes and with 50 runs for each such spectrum. In all cases, the match percentage is over 85%. It can be observed that spectra with longer fragment length seem to be easier to solve than ones with smaller fragment length. The same machine that we used to test the algorithm on the first data set was used for the second data set. Figure 4 shows that spectra with higher error percentage seem to take longer than spectra with smaller error percentage.

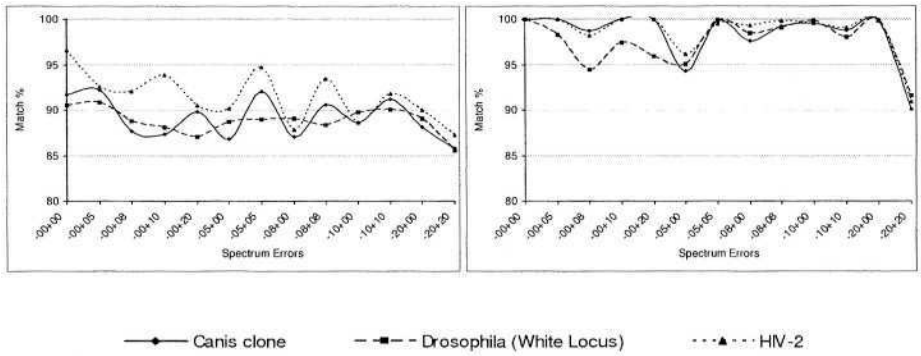


Fig. 3. Plot of match percentage of Enhanced GA against error combination for spectra with fragment length of 10 (left) and 20 (right). The label -a+b indicates spectra with $a\%$ negative error and $\%b$ positive error.

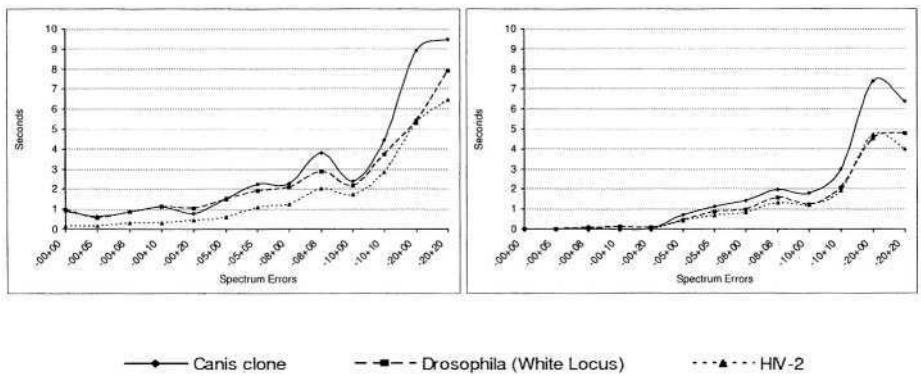


Fig. 4. Plot of running time of Enhanced GA against error combination for spectra with fragment length of 10 (left) and 20 (right). The label -a+b indicates spectra with $a\%$ negative error and $\%b$ positive error.

Experimental results from the two data sets suggest that our algorithm performs very well against existing algorithms. It is also very robust against different combinations of errors.

5 Conclusion

This paper introduced a new enhanced genetic algorithm for the DNA Sequencing problem. The results produced by the algorithm were very good and in many cases were optimal or close to optimal and were frequently better than existing algorithms. Taking into account the difference in speed of the machines on which the various algorithms were run, our algorithm seems to be comparable if not

faster than existing algorithms. This paper did not look at the case when the input spectrum contains repeated fragments. Repeated fragments can cause the algorithm not to be able to find optimal answers even when there are no other types of errors in the spectrum. This type of error diminishes if the fragment length increases. We plan to look into the problem of dealing with repeated fragments.

Acknowledgements. The authors would like to thank Jacek Blazewicz and Marta Kasprzak for providing them with the data used in [7], Steven S. Skiena for useful discussions, and Jie Li for providing them with an implementation of the Smith-Waterman algorithm. The authors would also like to thank Erick Cantu-Paz and the anonymous referees for helpful comments.

References

1. Blazewicz, J., P. Formanowicz, F. Glover, M. Kasprzak and J. Weglarz, "An Improved Tabu Search Algorithm for DNA Sequencing with Errors," Proceedings of the III Metaheuristics International Conference (MIC), 1999, pp. 69–75.
2. Blazewicz, J., P. Formanowicz, M. Kasprzak, W.T. Markiewicz and J.Weglarz, "DNA Sequencing with Positive and Negative Errors," Journal of Computational Biology 6, 1999, pp. 113–123.
3. Blazewicz, J., P. Formanowicz, M. Kasprzak, W. T. Markiewicz and J. Weglarz, "Tabu Search for DNA Sequencing with False Negatives and False Positives," European Journal of Operational Research 125, 2000, pp. 257–265.
4. Blazewicz, J. and M. Kasprzak, "Complexity of DNA sequencing by hybridization," Theoretical Computer Science, 290, 2003, pp. 1459-1473.
5. Blazewicz, J., A. Kaczmarek, M. Kasprzak, W. T. Markiewicz and J. Weglarz, "Sequential and Parallel Algorithms for DNA Sequencing," Computer Applications in the Biosciences 13, 1997, pp. 151–158.
6. Blazewicz, J., J. Kaczmarek, M. Kasprzak, J. Weglarz and W. T. Markiewicz, "Sequential Algorithms for DNA Sequencing," Computational Methods in Science and Technology, 1, 1996, pp. 31–42.
7. Blazewicz, J., M. Kasprzak and W. Kuroczycki, "Hybrid Genetic Algorithm for DNA Sequencing with Errors," Journal of Heuristics, 8, 2002, pp. 495–502.
8. Blum, A., T. Jiang, M. Li, J. Tromp and M. Yannakakis, "Linear Approximation of Shortest Superstrings," Journal of the ACM, 41(4), 1994, pp. 630–647.
9. Cummings, M. R. *Human Heredity: Principles and Issues,* West Publishing Company, 1991.
10. Fogel, G. B. and K. Chellapilla, "Simulated Sequencing by Hybridization Using Evolutionary Programming," Proc. of the IEEE Congress on Evolutionary Computation, CEC'99, 1999, pp. 445–452.
11. Fogel, G. B., K. Chellapilla and D. B. Fogel, "Reconstruction of DNA Sequence Information From a Simulated DNA Chip Using Evolutionary Programming," Lecture Notes in Computer Science, edited by V. W. Porto, N. Saravanan, D. Waagen and A. E. Eiben, Vol. 1447, 1998, pp. 429–436.
12. Frieze, A. and W. Szpankowski, "Greedy Algorithms for the Shortest Common Superstring That Are Asymptotically Optimal," Algorithmica, 21(1), 1998, pp. 21–36.

13. Goldberg, D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning,* Addison-Wesley, 1989.
14. Haan, N. M. and S. J. Godsill, "Sequential Methods For DNA Sequencing" Department of Engineering, University of Cambridge, U.K.
15. Kasprzak, M., Personal communications, August 2003.
16. Li, J. "Implementation of Smith-Water Alignment Algorithm," Iowa State University, Personal Communication.
17. Matsumoto, M. and T. Nishimura, "Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator," ACM Transactions on Modeling and Computer Simulation, 8(1), January 1998, pp. 3–30.
18. *Nonradioactive In Situ Hybridization Application Manual,* Technical Manual, Roche Applied Science.
19. Percus, A. G. and D. C. Torneyy, "Greedy Algorithms for Optimized DNA Sequencing," Los Alamos National Laboratory, Los Alamos, NM 87545.
20. Pevzner, P. A., *Computational Molecular Biology, An Algorithmic Approach,* The MIT Press, second printing 2001, Chapter 4, pp. 59-63.
21. Pevzner, P. A., H. Tang and M. S. Waterman, "An Eulerian Path Approach to DNA Fragment Assembly," Department of Computer Science and Engineering, University of California, San Diego, La Jolla, CA; and Departments of Mathematics and Biological Sciences, University of Southern California, Los Angeles, CA, June 7, 2001.
22. Phan, V. T. and S. Skiena, "Dealing with Errors in Interactive Sequencing by Hybridization," Oxford University Press, 17(10), 2002, pp. 1-9.
23. Skiena, S., Personal communications, September 2003.
24. Waterman, M.S., *Introduction to Computational Biology: Maps, Sequences and Genomes,* Chapman & Hall, London, 1995.

Unveiling Optimal Operating Conditions for an Epoxy Polymerization Process Using Multi-objective Evolutionary Computation

Kalyanmoy Deb[1], Kishalay Mitra[2], Rinku Dewri[3], and Saptarshi Majumdar[4]

[1] Mechanical Engineering Department, Indian Institute of Technology Kanpur, Kanpur-208016, deb@iitk.ac.in, http://www.iitk.ac.in/kangal/deb.htm
[2] Manufacturing Practice, Tata Consultancy Services, 54B Hadapsar Industrial Estate, Pune-411013, India, kmitra@pune.tcs.co.in
[3] Department of Mathematics, Indian Institute of Technology Kharagpur, Kharagpur-721302, rinku@webteam.iitkgp.ernet.in
[4] Tata Research Development and Design Centre, 54B Hadapsar Industrial Estate, Pune 411013, India, smajumdar@pune.tcs.co.in

Abstract. The optimization of the epoxy polymerization process involves a number of conflicting objectives and more than twenty decision parameters. In this paper, the problem is treated truly as a multi-objective optimization problem and near-Pareto-optimal solutions corresponding to two and three objectives are found using the elitist non-dominated sorting GA or NSGA-II. Objectives, such as the number average molecular weight, polydispersity index and reaction time, are considered. The first two objectives are related to the properties of a polymer, whereas the third objective is related to productivity of the polymerization process. The decision variables are discrete addition quantities of various reactants e.g. the amount of addition for bisphenol-A (a monomer), sodium hydroxide and epichlorohydrin at different time steps, whereas the satisfaction of all species balance equations is treated as constraints. This study brings out a salient aspect of using an evolutionary approach to multi-objective problem solving. Important and useful patterns of addition of reactants are unveiled for different optimal trade-off solutions. The systematic approach of multi-stage optimization adopted here for finding optimal operating conditions for the epoxy polymerization process should further such studies on other chemical process and real-world optimization problems.

Keywords: Multi-objective optimization, genetic algorithms, real-world optimization, Pareto-optimal solutions, chemical engineering process optimization.

1 Introduction

Real-world optimization problems often demand to cater the need of solving more than one objective simultaneously. Multi-objective optimization problems

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 920–931, 2004.

lead to a set of optimal solutions, known as *Pareto-optimal* solutions, as opposed to the single solution provided by any single-objective optimization task. Although only one solution must be chosen at the end of the optimization task and this often must be performed with the guidance of a decision-maker, it is a better practice to first find a set of Pareto-optimal solutions to have an idea of the extent of trade-offs possible among the underlying objectives before focusing on a particular solution [7]. Although the field of research and application on multi-objective optimization is not new, the use of evolutionary multi-objective optimization (EMO) techniques in various engineering and business applications is a recent phenomenon. Polymerization processes, being quite complex in nature, offer themselves as an extremely challenging candidate for multi-objective optimization studies. In modeling the polymerization system, several molecular parameters, such as the number or weight average molecular weights (M_n or M_w, respectively), the polydispersity index (PDI), concentration of different functional groups etc., can all be predicted quite accurately using various experimentally measured indices such as strength and stiffness of the final product. Moreover, the desired objectives in a polymerization process often exhibit conflicting relationships and therefore become an ideal problem for multi-objective optimization studies. In this paper, multi-objective optimization of a semibatch epoxy polymerization system which is often used to manufacture high-strength composites, reinforced plastics, adhesives, protective coatings in appliances, etc. is thoroughly investigated.

A recent review [2] reveals that several studies are carried out on multi-objective optimization of polymerization reactors. A number of studies [20,10,11, 3,5] considered multi-objective optimization of copolymerization reactors. Wajge and Gupta [21] studied multi-objective optimization of the nylon-6 batch reactor and obtained different optimal temperature histories corresponding to different solutions on the Pareto-optimal set using the same technique. Sareen and Gupta [18] extended that work and studied the nylon-6 semibatch reactor and obtained different optimal pressure histories and optimal jacket fluid temperature corresponding to different solutions on the Pareto-optimal set. A number of studies using an EMO approach are carried out on the multi-objective optimization of nylon-6 and polymethyl methacrylate (PMMA) reactors [4,16, 12,13]. These studies are mainly based on adapted version of the non-dominated sorting genetic algorithm (NSGA) developed by Srinivas and Deb [19].

In the Taffy process [14], the most popular industrial process for preparing epoxy polymers, bisphenol-A (monomer) and epichlorohydrin, in excess, are reacted in presence of sodium hydroxide (NaOH). Although it is well established that alkali plays a key role in the epoxy polymerization process, the role of addition of other reactants (bisphenol-A and epichlorohydrin) is not well established. Experimental and theoretical studies are very few in open literatures for the epoxy polymerization process. Raha and Gupta (1998) used species balance and equation of moments approach to study the process. They gave special importance to build the entire modeling framework as well as the effect of kinetic parameters and reactant's effect over the performance of the reaction process. However, based on some earlier studies [17] which showed the importance of all

three reactants, we launch the present detail study for a better understanding of the true nature of interactions among three conflicting objectives associated in an epoxy polymerization process.

2 Problem Formulation

The complete kinetic scheme for the above mentioned polymerization system can also be found elsewhere [1]. Raha, Majumdar and Gupta [17] have validated the model with the available experimental data. Using the species balance approach, ODEs corresponding to the initial value problem (IVP) are derived for various species and their moments. The "state" of the reaction scheme can be well described by a set of 48 state variables ($\mathbf{x} = (x_1, x_2, \ldots, x_{48})^T$), including all species balance and moment balance equations:

$$\frac{\mathrm{d}x_i}{\mathrm{d}t} = f_i(\mathbf{x}, \mathbf{U}), \quad i = 1, 2, \ldots, 48, \tag{1}$$

where $\mathbf{x}$ and $\mathbf{U}$ are vectors of the state and manipulated variables (such as the amount of intermediate additions for different reactants at different times). Manipulated variable vector consists of three discrete histories, namely, discrete history for NaOH addition (described here with $U_1(t_j)$), discrete history for epichlorohydrin addition ($U_2(t_j)$) and discrete history for bisphenol-A addition ($U_3(t_j)$) (where t_j is the j-th time of addition of reactants). Given three discrete profiles ($\mathbf{U}$ at time zero and at other time steps) and the initial conditions of all state variables ($\mathbf{x}$ at time zero), the reaction scheme model can be solved by using an explicit integrator (RK-type method) to solve all 48 differential equations. This simulation procedure can then be combined with the NSGA-II [9] optimization code for performing a multi-objective optimization.

2.1 Defining the Optimization Problems

Three different multi-objective optimization problems are studied here. The first problem (Problem 1) is related to the quality of polymer produced, whereas the second problem (Problem 2) addresses the productivity issue also:

$$\textbf{Problem 1:} \begin{cases} \text{Maximize } M_n, \\ \text{Minimize PDI}, \\ \text{subject to satisfying mass and moment balance equations}, \\ \qquad u_i^{\min} \le u_i \le u_i^{\max} \quad i = 1, \ldots, 21. \end{cases} \tag{2}$$

One computer simulation runs from zero (initial condition, $t = 0$) to $t = t_{\mathrm{sim}}$ (7 hr used here). Each of the three profiles ($U_1(t)$, $U_2(t)$, $U_3(t)$) are, therefore, discretized into seven equally spaced points. Each of these variables is forced to lie between a lower bound ($u_i^{\min}$) and a upper ($u_i^{\max}$) bound. No restriction on the M_n and PDI values are used as constraints, as the satisfaction of variable bounds will ensure limiting values on M_n and PDI.

The objective for Problem 2 is a vector of two objective functions: maximization of M_n and minimization of the overall reaction time, t_{sim}, subject to satisfying mass and moment balance equations. Here, the reaction time t_{sim} is also a decision variable, thereby making the total number of variables to 22.

The objective for Problem 3 is a vector of three objective functions: maximization of M_n, minimization of t_{sim}, and minimization of PDI, subject to mass and moment balance equations.

3 NSGA-II for the Epoxy Polymerization Problem

Each solution is represented as a real-valued vector of 21 (or 22 for problems 2 and 3) values indicating the addition of NaOH, EP, and AA_0. For the real-coded NSGA-II, we use the simulated binary crossover (SBX) and the polynomial mutation operators [8]. When a pre-specified maximum iteration count ($N = N_{\max}$) is reached, NSGA-II is terminated and the non-dominated solutions of the final population are declared as the obtained Pareto-optimal solutions. In problems 1 and 2, $N_{\max} = 200$ and a population size of $N_{\text{pop}} = 250$ are used. Since Problem 3 deals with a three-dimensional Pareto-optimal front, we have chosen a larger population size and run NSGA-II for more iterations: $N_{\text{pop}} = 1,000$ and $N_{\max} = 500$. The crossover and mutation probabilities of $p_c = 0.9$ and $p_m = 0.1$ are used for the real-coded NSGA-II. NSGA-II can handle mixed type of optimization problems. Thus, we have used the objectives as they are mentioned in the previous section.

Based on some earlier studies, following variable bounds are set here to allow the optimizer (NSGA-II) a substantial search space to look for the optimized solutions: (i) the addition of NaOH varies between 0.2 and 1.0 kmol/m^3 at $t = 0$ hr and between zero and 1.0 kmol/m^3 for $t > 0$ hr., (ii) the addition of EP varies between 0.2 and 2.0 kmol/m^3 at $t = 0$ hr and between zero and 2.0 kmol/m^3 for $t > 0$ hr., and (iii) the addition of AA_0 varies between 0.2 and 1.0 kmol/m^3 for $t = 0$ hr and between zero and 1.0 kmol/m^3 for $t > 0$ hr.

4 Discussion on Problem 1

A simulation result corresponding to the initial condition of Batzer and Zahir [1] (with M_n=633.2 kg/kg-mole and PDI=1.61) is considered as the benchmark performance data with which our optimized solutions will be compared. The solutions termed as "Initial" in Figure 1 denote the objective vectors with which the NSGA-II search process is started. The figure indicates that the random solutions (within the chosen variable bounds) are far from being close to the optimized front (marked as 'NSGA-II'), particularly producing solutions with large M_n values. The figure clearly shows that a wide range of distribution in both M_n and in PDI values are obtained. The trade-off obtained between the two objectives is also clear from the figure. The following observations can be made from the obtained results: (i) The resulting M_n-PDI trade-off is *non-convex,* a phenomenon which is rare in real-world multi-objective optimization

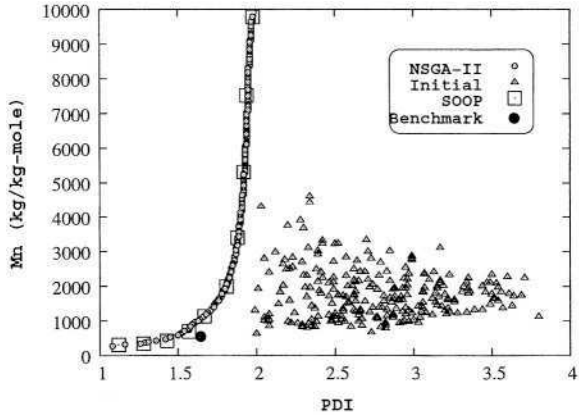


Fig. 1. Obtained NSGA-II solutions for the M_n and PDI optimization are compared with single-objective optimization (SOOP) solutions. Initial solutions of NSGA-II and the benchmark solution are also shown.

problems, (ii) Investigating the solutions of Figure 1 further, it can be inferred that compared to the benchmark solution (marked as 'Benchmark' in the figure) there exist better solutions which can produce larger M_n (ranging from 645 to 966 kg/kg-mole) and less PDI (ranging from 1.52 to 1.61, respectively) value, leading to polymers with better properties as compared to benchmark solution.

In order to verify whether the obtained NSGA-II solutions are actually close to the true Pareto-optimal front of this problem, we use a single-objective preference based method next. Since, the above observation indicates that the Pareto-optimal front is non-convex, the commonly-used weighted-sum approach [15] may not be the right approach for this problem, as the weighted-sum approach is known to be inadequate in finding the Pareto-optimal solutions in the non-convex region [6]. Thus, we use the the ϵ-constraint method [15] here. In this case, we convert the second objective (Minimization of PDI) into an additional constraint as $\text{PDI} \leq \text{PDI}_\epsilon$ and maximize only the first objective. Other constraints and variable bounds, as given in Problem 1, are kept the same. To obtain different Pareto-optimal solutions, we simply choose a different value for PDI_ϵ and optimize the resulting single-objective optimization problem using a single-objective GA. The constraints are handled using a penalty parameter less procedure [6]. The GA parameters, such as the population size, operator parameters, etc., are kept the same as those used in the above NSGA-II study. Figure 1 marks these solutions as 'SOOP' solutions obtained by 10 independent GA runs, each performed with adifferent PDI_ϵ value. Since these solutions are found to lie on or near the Pareto-optimal front obtained by the NSGA-II, it can be stated that the non-dominated front found by NSGA-II is probably the true Pareto-optimal front.

4.1 Searching for Salient Properties of Pareto-Optimal Solutions

Each of the solutions on Pareto-optimal front carries information about 21 decision variables. The Fritz-John necessary condition for Pareto-optimality conditions are [6,15] are presented below:

Definition 1. *(Fritz–John necessary condition). A necessary condition for* $\mathbf{x}^*$ *to be Pareto-optimal is that there exist vectors* $\boldsymbol{\gamma} \geq \mathbf{0}$ *and* $\mathbf{v} \geq \mathbf{0}$ *(where* $\boldsymbol{\gamma} \in \mathbb{R}^M$, $\mathbf{v} \in \mathbb{R}^J$ *and* $\boldsymbol{\gamma}, \mathbf{v} \neq \mathbf{0}$*) such that the following conditions are true:*

1) $\sum_{m=1}^{M} \gamma_m \nabla f_m(\mathbf{x}^*) - \sum_{j=1}^{J} v_j \nabla g_j(\mathbf{x}^*) = \mathbf{0}$, *and*
2) $v_j g_j(\mathbf{x}^*) = 0$ *for all* $j = 1, 2, \ldots, J$,

where the underlying multi-objective optimization problem (with M objectives f_m to be minimized) is assumed to have J inequality constraints: $g_j(\mathbf{x}) \geq 0$ for $j = 1, \ldots, J$. The above conditions, although cannot be applied to our problem directly due to the inability to compute the gradients of the objective functions and constraints, suggest certain common properties which all Pareto-solutions must satisfy. This leads us to believe that the obtained solutions, if close to the Pareto-optimal solutions, will share some common properties among them and, of course, will have some differences in order to have trade-offs among them. If such properties exist, they would be worth searching for in a real-world application problem, as the sheer knowledge of them will provide important and useful information about the optimal trade-off among objectives [7]. An investigation is performed next to identify whether the obtained M_n-PDI solutions bear any similarity in terms of the associated decision variables. Interesting trends are discovered and shown in Figure 2.

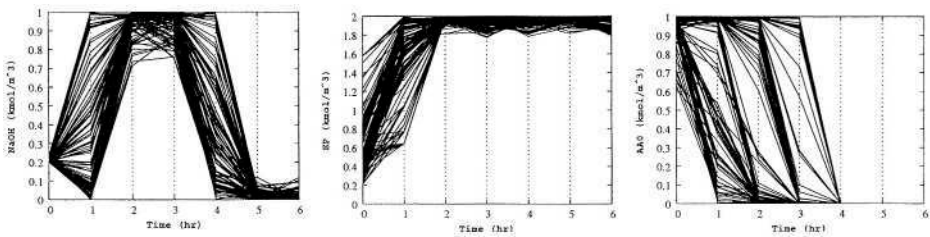


Fig. 2. Time-variant additions of NaOH (top-left), EP (top-right), and AA_0 (bottom) show common patterns for real-coded NSGA-II solutions on Problem 1.

Though the decision variables are discrete additions at various time steps, they are joined with straight lines to show the trends. A casual look at the plots will reveal some interesting patterns followed in all obtained solutions. Although the additions at every hour could have been anywhere on the vertical axis at each time step, all obtained solutions seem to follow some patterns. These patterns reveal important insights about the optimal working characteristics of the epoxy

polymerization problem, some of which we have deciphered and are discussed in the following:

1) The general trend captured in NaOH addition is to start from the lower bound, reduce or increase the addition at the first hour, increase close to the upper bound in the next hour, then continue with the same amount for some more time and finally reduce close to the lower bound. A high amount of NaOH addition is required in the first phase of reaction process for a better initiation of polymerization and the amount of NaOH can be kept low in the later part of the process as mainly the growth of chain length occurs at that part of time and NaOH is produced as a by-product. The reason for NaOH to come down at the lower bound after the addition at the zero-th hour for some solutions is due to the fact that the initiation steps require NaOH as a reactant and also produce as a by-product. In case of M_n value less than 7.0×10^3 kg/kg-mole, NaOH is found to be added to the system in a controlled fashion. For M_n value greater than 7.0×10^3 kg/kg-mole, NaOH, instead of decreasing at time step of the first hour after the first addition, goes up and rest of the trend remains same as it is stated earlier. In these cases, NaOH takes part not only in the initiation but also in growth of chain length by helping to form certain species of polymer (BE_n) which contributes significantly towards the high value of M_n.
2) The trend found for epichlorohydrin is straightforward. It must be started at different levels depending on the required M_n value, but must be quickly increased close to the upper limit and must be continued at that amount till the end. The reason for small required value of epichlorohydrin initially is due to it having a less contribution in polymer chain initiation. However, epichlorohydrin should be supplied maximally in the later stages due to its major contribution in chain growth mechanisms.
3) Bisphenol-A must be started in large amount and then must be reduced with time. This is because bisphenol-A takes part actively in polymerization initiation step and its requirement is reduced at the later part. The quantity of addition is observed to prolong for longer time steps for higher values of M_n. This happens because a large M_n value is achieved by adding more amount of bisphenol-A in the system. It has been seen that the added bisphenol-A is consumed almost completely (i.e. not added more than the amount required) in most of the cases.

It is clear from these observations that different polymers (with different M_n and PDI values) must be produced optimally with a different addition pattern of reactants. Although some of these observations can be explained from the chemistry of the process, Figure 2 shows the optimal operating conditions.

To better understand the characteristics of solutions at different parts of the optimized front, we divide the entire front into three groups. As the solutions on optimized front spans M_n from 0.4×10^3 kg/kg-mole to almost 9.5×10^3 kg/kg-mole, each region roughly spans 3.0×10^3 kg/kg-mole in M_n axis: 0.0 – 3.0×10^3 kg/kg-mole M_n (Group 1), $3.0 - 6.0 \times 10^3$ kg/kg-mole M_n (Group 2), and $6.0 - 9.0 \times 10^3$ kg/kg-mole M_n (Group 3). The representative solution in each group are shown in Table 1 and the regions for each group is also marked in Figure 1.

Table 1. Three representative solutions picked from the optimized front obtained by real-coded NSGA-II in each of the three problems are shown.

Source	PDI (kg/kg)	M_n (kg/kg-mole)	Reaction time (hr)
Benchmark	1.61	633.2	7.00
Problem 1			
Group 1	1.59	926.5	7.00
Group 2	1.87	3186.1	7.00
Group 3	1.97	9507.8	7.00
Problem 2			
Group 1	1.70	946.6	2.57
Group 2	1.90	3219.5	4.00
Group 3	1.99	9507.6	6.60
Problem 3			
Group 1	1.61	943.71	3.80
Group 2	1.88	3218.02	5.17
Group 3	1.97	9508.45	6.98

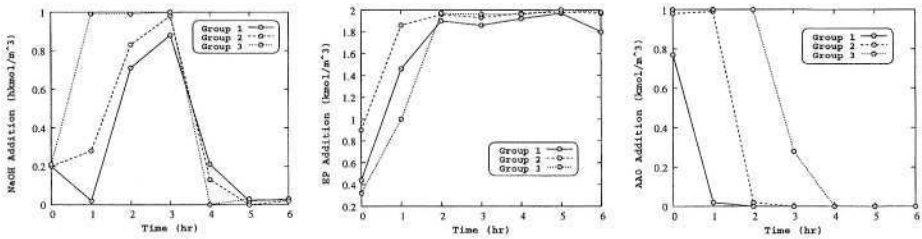


Fig. 3. Time-variant additions of NaOH (left), EP (middle), and AA_0 (right) are shown for three representative solutions from each group on the optimized front obtained using the real-coded NSGA-II for Problem 1.

The trend in addition of reactants for each group is shown in Figure 3.

Each group has a distinct pattern of adding the three constituents. The information depicted in these figures conveys that there lies a relationship between the solutions in the optimized front and the system under consideration. The relationship can be regarded as the 'blue-print' of the system. Given a set of objectives, certain properties emerge from the system, not arbitrarily, but following some basic properties of the system. This relationship between the property of the system and the solutions of the optimized front would be of tremendous importance to practitioners.

5 Discussion on Problems 2 and 3

For brevity, we do not show the obtained front for Problem 2 separately. Instead, we show the front along with the results of Problem 3 obtained using three objec-

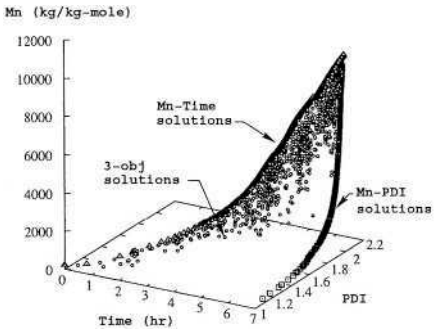


Fig. 4. NSGA-II solutions obtained from the three-objective optimization are shown. The fronts obtained in the previous two problems are found to lie on two edges of the obtained three-dimensional front.

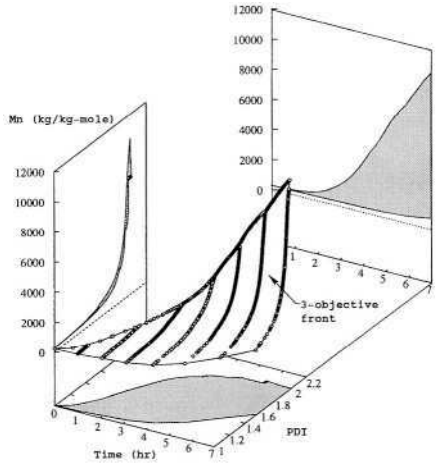


Fig. 5. Several optimizations of Problem 1 with different t_{sim} values help find the third boundary of true three-dimensional non-dominated front.

tives in Figure 4. It is interesting that the limiting fronts (from problems 1 and 2) lie on the two edges of the three-dimensional front. We deduce the following conclusions from the three-dimensional front:

1) The three-dimensional non-dominated front is a non-convex front.
2) For low values of M_n, polymers can be prepared in much less than 7 hours. Since the reaction time is also minimized in Problem 3, there exists almost no solution requiring as large as 7 hr to produce a polymer having a low M_n.
3) For high values of M_n, more reaction time is needed. As in the polymerization process, the repeat units get added with the monomer to form a longer chain-length, a polymer having a high value of M_n requires more time to form than a polymer having a low value of M_n. But for a desired M_n and PDI combination, one can find a solution requiring smaller than 7 hours to do the job, but the occurrence of such a quick operation gets reduced with the requirement of higher and higher M_n values.
4) Finally, for the maximum M_n requirement, there exists only one solution (with M_n=10,402.12 kg/kg-mole), requiring 7 hours (the maximum allowed) of reaction time, but also producing the largest PDI value (1.985). Such is the trade-off often observed in a multi-objective problem and Figure 4 shows many such trade-off solutions producing different values of M_n and PDI and requiring different amount of reaction time.
5) Another interesting aspect is that for any fixed reaction time, the M_n-time optimization (Problem 2) solutions produced the maximum M_n value and there exists no other solution producing a better (smaller) PDI value and an identical

M_n value. But, the three-dimensional optimized front provides more information about the trade-off than both two-objective optimized fronts, discussed earlier.

On the three-dimensional optimized front, there seems to be a larger concentration of solutions towards the M_n-time front (in other words, there are more solutions requiring a smaller processing time). To understand this trend better, we repeat Problem 1 for different reaction times. We force the reaction time to end at $t_{\text{sim}} = 1$ hr, 2 hr and so on, and collect all the obtained optimized solutions. Thereafter, we perform a non-domination check considering all three objectives (maximization of M_n, minimization of PDI, and minimization of t_{sim}) and the resulting solutions are plotted in Figure 5. An interesting aspect is revealed. For identical PDI values, there exists a smaller reaction time solution outperforming a larger time solution. This feature of the problem produces a *third* boundary on this three-dimensional non-dominated front, thereby showing the complete bounded trade-off surface of interactions. The figure also shows the projection of the three-dimensional trade-off surface on the three two-objective planes. It is clear that although there are some rooms for trade-offs among M_n-time and PDI-time combinations, in terms of M_n-PDI combinations, there is almost a straightforward trade-off. This fact also indicates that the consideration of minimization of the reaction time is important in this problem to reveal interesting time-saving trade-off solutions. To understand the three-objective interactions further, we consider the PDI-time and M_n-time projections (as shown shaded in Figure 5) and plot them again in Figure 6 in the left and right plots, respectively.

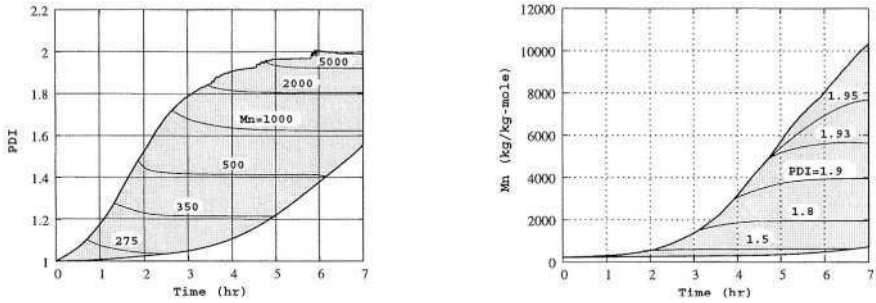


Fig. 6. PDI and reaction time interaction (left) and M_n and reaction time interaction (right) reveal interesting properties about the optimal operating conditions of the polymerization problem. In the left figure, M_n is shown in kg/kg-mole.

On the PDI-time plot, we show contours of fixed M_n values. For each contour line, the trade-off between PDI and reaction time is clear. However, what is more interesting is that the trade-off becomes marginal away from the M_n-time optimized front (the upper boundary). To achieve a small advantage in the PDI value, a large reaction time is necessary. Since the slope of the three-objective optimized front is quite small at these regions, the NSGA-II has found very few

solutions away from the M_n-time boundary. It can then be concluded that for a fixed M_n requirement, it is better to consider the trade-off solutions close to the M_n-time optimized boundary. Another aspect is the rate at which the 'region of optimality' reduces for an increasing M_n value. For example, if a polymer with M_n greater than 5,000 kg/kg-mole is desired, there does not exist too many options in terms of the reaction time. The right plot in Figure 6 shows contour lines for fixed values of PDI on the three-objective optimized front. Once again the trade-off between M_n and reaction time is clear from the plot. Although a similar conclusion (about choosing solutions close to the M_n-time optimal boundary) can be made for smaller PDI values, for large PDI values (such as PDI=1.95) the trade-off between M_n and reaction time is quite substantial. These two plots can be used to determine a suitable operating condition for a particular application of the epoxy polymerization process.

6 Conclusions and Extensions

A well-validated model consisting of a large number of moment-based ordinary differential equations has been utilized for the multi-objective optimization of epoxy polymerization process using an evolutionary algorithm (NSGA-II). The aim of the study has been to extract the discrete addition patterns of the reactants for optimizing various objectives simultaneously (e.g. Problem 1: Maximization of M_n with minimization of PDI, Problem 2: Maximization of M_n with minimization of reaction time and Problem 3: Maximization of M_n, minimization of PDI, and minimization of reaction time). All three problems considered here have displayed a non-convex non-dominated front. NSGA-II has been able to find solutions on or near the true non-dominated front of the problems. This has been validated by solving the multi-objective problems using a single-objective preference based optimization method (ϵ-constraint method). The advantage of using the NSGA-II is that it has found multiple (as many as 250) optimized solutions in a single simulation run.

Importantly, the multi-objective optimization of the epoxy polymerization process has led to the discovery of certain operating principles (addition time-pattern of three reactants (NaOH, Epichlorohydrin, and Bisphenol-A)) for all high-performance solutions. The trade-off between the objectives have been clearly characterized by showing and contrasting a representative additive pattern of all three reactants. Such information brings out the 'blue-print' of the optimal operating conditions of a chemical process and are often important in real application in an industry.

References

1. H. Batzer and S. S. Zahir. Studies in the molecular weight distribution of epoxide resins. *Journal of Applied Polymer Science,* 21:1843, 1977.
2. V. Bhaskar, S. K. Gupta, and A. K. Ray. Applications of multi-objective optimization in chemical engineering. *Reviews in Chemical Engineering,* 16(1):1, 2000.

3. D. Butala, K. Y. Choi, and M. K. H. Fan. Multiobjective dynamic optimization of a semibatch free-radical copolymerization process with interactive cad tools. *Comput. Cham. Eng.,* 12:1115, 1988.
4. S. S. S. Chakravarthy, D. N. Saraf, and S. K. Gupta. Use of genetic algorithms in the optimization of free radical polymerizations exhibiting the trommsdorff effect. *J. Appl. Poly. Sci.,* 63:529, 1997.
5. K. Y. Choi and D. N. Butala. An experimental-study of multi-objective dynamic optimization of a semibatch copolymerization process. *Poly. Eng. Sci.,* 31:353, 1991.
6. K. Deb. *Multi-objective optimization using evolutionary algorithms.* Chichester, UK: Wiley, 2001.
7. K. Deb. Unveiling innovative design principles by means of multiple conflicting objectives. *Engineering Optimization,* 35(5):445–470, 2003.
8. K. Deb and R. B. Agrawal. Simulated binary crossover for continuous search space. *Complex Systems,* 9(2):115–148, 1995.
9. K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast and elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation,* 6(2):182–197, 2002.
10. L. T. Fan, C. S. Landis, and S. A. Patel. *Frontiers in Chemical Reaction Engineering,* page 609. New Delhi: Wiley Eastern, 1984.
11. J. N. Farber. Steady state multiobjective optimization of continuous copolymerization reactors. *Poly. Eng. Sci.,* 26:499, 1986.
12. S. Garg and S. K. Gupta. Multiobjective optimization of a free radical bulk polymerization reactor using genetic algorithm. *Macromol. Theor. Simul.,* 8:46, 1999.
13. R. R. Gupta and S. K. Gupta. Multi-objective optimization of an industrial nylon-6 semibatch reactor system using genetic algorithm. *J. Appl. Poly. Sci.,* 73:729, 1999.
14. A. Kumar and S. K. Gupta. *Reaction Engineering of Step Growth Polymerization.* New York: Plenum, 1987.
15. K. Miettinen. *Nonlinear Multiobjective Optimization.* Boston, MA: Kluwer, 1999.
16. K. Mitra, K. Deb, and S. K. Gupta. Multi-objective dynamic optimization of an industrial nylon 6 semibatch reactor using genetic algorithm. *J. App. Poly. Sci.,* 69:69, 1998.
17. S. Raha, S. Majumdar, and K. Mitra. Effect of caustic addition in epoxy polymerization process: A single and multiobjective evolutionary approach. *Macromolecular Theory and Simulations,* in press.
18. R. Sareen and S. K. Gupta. Multiobjective optimization of an industrial semibatch nylon 6 reactor. *J. Appl. Poly. Sci.,* 58:2357, 1995.
19. N. Srinivas and K. Deb. Multi-objective function optimization using non-dominated sorting genetic algorithms. *Evolutionary Computation Journal,* 2(3):221–248, 1994.
20. A. Tsoukas, M. V. Tirrell, and G. Stephanopoulos. Multi-objective dynamic optimization of semibatch copolymerization reactors. *Chem. Eng. Sci.,* 37:1785, 1982.
21. R. M. Wajge and S. K. Gupta. Multiobjective dynamic optimization for a nonvaporizing nylon-6 batch reactor. *Poly. Eng. Sci.,* 34:1161, 1994.

Efficient Clustering-Based Genetic Algorithms in Chemical Kinetic Modelling

Lionel Elliott[1], Derek B. Ingham[1], Adrian G. Kyne[2], Nicolae S. Mera[2], Mohamed Pourkashanian[3], and Sean Whittaker[1]

[1] Department of Applied Mathematics, University of Leeds, Leeds, LS2 9JT, UK
{lionel, amt6dbi}@amsta.leeds.ac.uk, che9sw@leeds.ac.uk
[2] Centre for Computational Fluid Dynamics, Energy and Resources Research Institute, University of Leeds, Leeds, LS2 9JT, UK
{fueagk, fuensm}@sun.leeds.ac.uk
[3] Energy and Resources Research Institute, University of Leeds, Leeds, LS2 9JT, UK
fue6lib@sun.leeds.ac.uk

Abstract. Two efficient clustering-based genetic algorithms are developed for the optimisation of reaction rate parameters in chemical kinetic modelling. The genetic algorithms employed are used to determine new reaction rate coefficients for the combustion of four different fuel/air mixtures in a perfectly stirred reactor (PSR). The incorporation of clustering into the genetic algorithm allows for a considerable reduction in the number of computationally expensive fitness evaluations to be realised without any loss in performance. At each generation, the individuals are clustered into several groups and then only the individual that represents the cluster is evaluated using the expensive fitness function. The fitness values of the other individuals in the same cluster are estimated from the representative individual based on a distance measure in a process called *fitness imitation.*

1 Introduction

Many combustion phenomena are kinetically controlled; these may include the formation of pollutants in an exhaust stack, the burning velocity of a premixed flame, or the conversion of NO to NO_2 in a gas turbine combuster. In order to fully understand the chemical processes occurring in such phenomena, it is essential that a detailed chemical kinetic approach be undertaken.

One popular way to model the chemistry of combustion is to use a system of chemical reactions for which the rates of each reaction are known. Reaction rate data for simple fuels, such as hydrogen, is known with a high degree of confidence, see [1], and thus the combustion of hydrogen at temperatures in excess of 1000K may be modeled using a simple eight-step reaction mechanism. However, considerably more complex fuels, such as kerosene, require more than 1000 reaction steps with over 200 species, see [2].

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 932–944, 2004.

Genetic algorithms (GAs) can be used to optimise the reaction rate data of a chemical kinetics problem. A method utilising GAs is proposed in [3] for the rapid extraction of the chemical kinetic rate coefficients for a simplified combustion mechanism from a given set of (detailed) chemical data. In this method the matching of heat release and species production rates of the simplified mechanism to those of an underlying detailed chemical mechanism allows for speedy identification of rate coefficients for a particular operating condition.

In [4], a powerful inversion technique that is based on a binary-coded GA, is developed. Here, new Arrehenius reaction rate coefficients for the combustion of a hydrogen/air mixture in a perfectly stirred reactor are determined. In this technique, the output species profiles obtained from an original set of rate constants are reproduced by a new different set determined using a GA inversion process. A kinetic scheme defined by decoding the genetic data generated by the GA can be used as the basis for running CHEMKIN's Perfectly Stirred Reactor (PSR) computer library, see [5] and [6], and the resulting net species concentration outputs compared to known data. The PSR code can be used to establish the net species concentrations of each of the products based upon different reactor conditions for given estimates to the set of Arrhenius reaction rate coefficients. The overall goal of the inversion process is to determine the unknown reaction rate coefficients which both match the given net species concentrations at different reactor conditions and will, furthermore, correctly predict the net species concentrations at other reactor conditions.

Many engineering or scientific GA-based optimisation problems, involve the use of a simulator or analysis code linked to the GA. It is this code that is used to generate fitness values for the population of possible solutions produced by the GA. One main difficulty in applying GAs to such problems is that a large number of fitness evaluations, i.e. executions of the simulator or analysis code, are usually required before a satisfactory result can be obtained. In general, the computational expense for a single execution of this code is very high compared to the expense for generating and manipulating the population of possible solutions. The main focus of this study is to the incorporation of clustering techniques into the GA in order to speed up the optimisation process without any loss in the algorithms performance. The hybrid clustering-based GAs, are tested on four different fuel/air mixture PSR problems, and the results are compared with a standard GA. The four test problems are a hydrogen/air mixture, a formyl-radical/air mixture, a methane/air mixture, and a kerosene/air mixture. The computational expense increases from the hydrogen/air mixture through to the kerosene/air mixture.

In this study, a floating point version of the GA has replaced the binary-coded GA used previously as they have been shown to be especially suited to numerical optimisation on continuous domains, see [7].

2 The Perfectly Stirred Reactor (PSR) Code

2.1 Introduction

The Perfectly Stirred Reactor (PSR) code is one of three codes available from Sandia National Laboratories that has been used in conjunction with the CHEMKIN II suite of software. The PSR code is a FORTRAN computer program that predicts the steady-state temperature and species composition in a perfectly stirred reactor, whilst the CHEMKIN suite handles the chemical reaction mechanism and the thermodynamic properties.

The stirred reactor consists of a small thermally insulated chamber that has both inlet and outlet ducts. The reactor is characterised by a reactor volume, residence time or mass flow rate, heat loss or a gas temperature, and an inlet temperature and mixing composition. High-intensity turbulent mixing of the steady inlet flow of fuel and oxidiser produces a nearly spatially uniform distribution of contents within the reactor. Because it is assumed that the mixing process is infinitely fast, the conversion of reactants to products is controlled solely by chemical reaction rates.

2.2 Reaction Rate Parameters

In general, Y_k is used to denote the mole fraction of the k^{th} species, where $k = 1,..,K$ and K represents the total number of species. The mole fractions of the k^{th} species at the inlet are denoted by Y^*_k, for $k = 1,..,K$, and the inlet temperature by T^*, whilst the temperature and composition which exit the reactor are assumed to be the same as those in the reactor since the mixing in the reactor chamber is intense.

The net chemical production rate of each species results from a competition between all the chemical reactions involving that species. It is assumed that each reaction proceeds according to the law of mass action and the forward rate coefficients are in modified Arrhenius form

$$k_{f_i} = A_i T^{\beta_i} \exp\left(-\frac{E_i}{RT}\right) \tag{1}$$

for $i = 1,..,N_R$, where T is the temperature, $R = 1.9860$ cal mol^{-1} K^{-1} is the universal gas constant, there are N_R competing reactions occurring simultaneously, and the rate equations (1) contain the three parameters A_i, β_i, and E_i for the i^{th} reaction. A_i is the pre-exponential collision frequency factor, β_i is the non-Arrhenius index, and E_i is the activation energy. It is the incorporation of clustering techniques into the GA for the determination of these parameters for each reaction, based upon outlet species mass fractions alone, which is investigated in this study.

2.3 Problem Formulation

During this study, results obtained using the clustering-based GAs are compared with those obtained from a standard GA for four test problems, namely

(i).	Hydrogen/air	9 species and 19 reactions.
(ii).	Formyl-radical/air	22 species and 67 reactions.
(iii).	Methane/air	24 species and 103 reactions.
(iv).	Kerosene/air	97 species and 460 reactions.

For each of the four test problems, an inverse solution procedure is set up in an attempt to recover the species concentration profiles predicted by a previously validated reaction mechanism over a limited range of reactor conditions. A total of $N_s = 11$ different PSR conditions are considered for the first three problems, corresponding to various changes to the inlet temperature (T = 1000K,..,2000K, with ΔT = 100K), whilst $N_s = 5$ different PSR conditions are considered for the kerosene mixture corresponding to T = 1000K, 1040K, 1080K, 1110K, and 1150K.

The inversion process aims to determine the unknown Arrhenius reaction rate coefficients (A_i, β_i, and E_i in (1)) by searching for the set of reaction rates that gives the best fit to a given set of data, i.e. the validated reaction mechanism. A ± 25% variation from the values given in the validated reaction mechanism is used to define the boundaries within which the GA searches for solutions. This small variation in the values of the reaction rate coefficients is sufficient to produce significantly larger variations in the net species concentrations.

A set of output species concentration profiles is obtained by simulation using a previously validated reaction mechanism. If data is simulated for N_s different reactor conditions, and all K species are measured for each condition, then the data will consist of $K{\cdot}N$ species concentration measurements. A set of Arrhenius reaction rate coefficients is chosen which gives a best fit to the species concentration measurements. Such a fitting procedure is accomplished using an optimisation technique that looks for the maximum of the following function using a GA

$$f_{\mathrm{PSR}}\left(\left(A_i, \beta_i, E_i\right)_{i=1,..,N_R}\right) = \left\{10^{-8} + \sum_{j=1}^{N_s}\sum_{k=1}^{K}\left(Y_{jk}^{\mathrm{calc}} - Y_{jk}^{\mathrm{orig}}\right)^2\right\}^{-1} \tag{2}$$

where

- Y_{jk}^{calc} represents the mole concentration of the k^{th} species in the j^{th} set of reactor conditions using the set of Arrhenius rate coefficients.
- Y_{jk}^{orig} represents the corresponding original value for the same mole concentration obtained by simulation using a validated reaction rate mechanism.

3 Optimisation Using the Genetic Algorithm

3.1 The Genetic Algorithm

A floating point number encoded generational GA is used in this study in place of a previously used binary encoded GA as they have been shown to be better suited to numerical optimisation on continuous domains. Previous work incorporating a standard GA into an optimisation procedure for the recovery of reaction rate parameters highlighted the following genetic operators as most appropriate for this study, roulette selection, BLX-α crossover, crossover probability $p_c = 0.6$, and non-uniform mutation, mutation probability $p_m = 0.6$. A value of $\alpha = 0.5$ was chosen for the crossover operator as this strikes an equal compromise between exploration and exploitation of the search domain. The maximum number of generations performed by the GA was set to 1000. The population size and the size of the offspring pool will be discussed in section 4. An elitist strategy is incorporated into the GA with the two fittest individuals from the previous generation surviving into the next generation.

3.2 An Introduction to Fitness Approximation

A large proportion of real-world engineering or scientific GA-based optimisation problems involve the use of a simulator or analysis code to compute the population fitness values. In many cases, each application of the simulator code takes a non-negligible amount of time to return a fitness value, and thus for problems that require a high number of fitness evaluations in order to return a satisfactory solution, the computational expense becomes a major issue.

Several methods exist that involve the use of approximation models to replace some or all of the computationally expensive evaluations. The most popular ones include polynomial modelling whereby the fitness landscape is approximated using a polynomial function; with quadratic approximation functions being favoured, see [8], the Kriging model, see [9], neural networks including multi-layer perceptrons and radial basis function networks, see [10] and Support Vector Machines (SVM), see [11].

There are two major concerns regarding the use of these models for fitness evaluation. First, it should be ensured that the algorithm converges to the global optimum of the original fitness function. Second, the computational cost should be reduced as much as possible. The advantage of such models is that they are in, general, significantly less computationally expensive than the original fitness function. A comprehensive review of fitness approximation techniques in evolutionary computation can be found in [12].

The application of many of the more common approximation models such as those listed in [8] to [11] to problems of a high dimension is not practical as the computational expense of constructing and executing the model is in many cases comparable to the original fitness function. The remaining part of this paper concerns the use of two clustering algorithms each separately embedded into a standard GA, and each

used to form an approximate model from which it is possible to significantly reduce the total number of original fitness function evaluations without loss in performance for four PSR combustion problems of high dimension.

3.3 Fitness Approximation by Clustering

Clustering is an exploratory data analysis method applied to data samples in order to discover structures or certain groupings in a data set. The data samples are grouped so that they exhibit some degree of similarity within each group. These groups are called clusters. There are three general categories of clustering, hierarchical clustering, partitional clustering, and overlapping clustering.

Perhaps the most important aspect of clustering is the choice of similarity measure. Most common clustering methods use a distance measure to assign similarity; these include the city block distance, the Euclidean distance, and the Minkowski distance. In general, the distance d_{ij} between the i^{th} and j^{th} vector data samples $\underline{x}_i$ and $\underline{x}_j$ whose dimension is n is computed as

$$d_{ij} = d\left(\underline{x}_i, \underline{x}_j\right) = \sqrt[m]{\sum_{k=1}^{n} \left| x_{ik} - x_{jk} \right|^m} \tag{3}$$

where

- $m = 1$ City block distance
- $m = 2$ Euclidean distance
- $m = 3$ Minkowski distance

Hierarchical clustering algorithms construct a structure of clusters and provide a view of the data at different levels of granularity. The approach usually takes one of two approaches, the first is the agglomerative approach, and the second is the divisive approach, see [13]. There are several commonly used hierarchical clustering algorithms and these include the single-linkage algorithm, the complete-linkage algorithm, the average-linkage algorithm, and Ward's method.

Partitional clustering algorithms create an unstructured set of clusters whereby the original data set is partitioned into similar groups based on proximity to one another, with the view that samples close together are similar. Each cluster is completely separate from the other clusters and no overlapping occurs. There are several commonly used partitional clustering algorithms and these include the *k*-means algorithm, the hard c-means algorithm, and Forgy's algorithm, see [13].

Overlapping clustering algorithms create a clustering pattern similar to that of partitional clustering with exception that the clusters are allowed to partially overlap on another. Commonly used examples of this type of clustering include the fuzzy *c*-means algorithm, and the *b*-clump algorithm.

In this study the *k*-means partitional clustering algorithm and Ward's hierarchical method (also known as the minimum-variance method) are each separately incorporated into a standard GA and are used to cluster the population at each and every

generation after initialisation. After clustering, the representative for each cluster is chosen and its fitness is evaluated using the original fitness function (2). The fitnesses of the remaining individuals in the population are then estimated based on the Euclidean distance from their cluster representative.

The *k*-means algorithm clusters the GA population in the following way,

a. At each and every generation after initialisation the GA generates the n_{child} individuals in the usual way with the first n_{cluster} individuals forming the initial cluster sites (this is a random choice due to the individuals being generated randomly by the GA).

b. Consider the first of the $n_{\text{child}} - n_{\text{cluster}}$ remaining individuals and place it in the cluster whose centroid is closest.

c. Re-compute the centroid of the altered cluster.

d. Repeat steps 2 and 3 until the whole population is clustered.

e. The whole population is now clustered in such a way that each individual is closer to their respective cluster centroid than to any other.

Ward's method clusters the GA population in the following way

a. Each of the n_{child} individuals generated by the GA at each generation forms an initial cluster site.

b. The number of cluster sites is reduced one at a time until n_{cluster} clusters are remaining.

c. At each cluster reduction, the method merges the two clusters resulting in the smallest increase in the total sum of squares of the distances of each individual to its respective cluster centroid. This total sum increases monotonically as the number of clusters decreases.

d. Sites clustered at a previous clustering step are never unmerged.

In problems where the components of the vector data sample differ by several orders of magnitude, as is the case with PSR or PREMIX modelling it is necessary to scale the components such that they all belong to [0,1]. This is accomplished in the following way:

$$\tilde{x}_k = \frac{x_k - x_k^{\text{lower}}}{x_k^{\text{upper}} - x_k^{\text{lower}}} \tag{4}$$

where, x_k^{upper} and x_k^{lower} are the upper and lower bounds respectively of the search space for the k^{th} component of the vector data sample.

There are many ways to choose the cluster representatives, but the easiest is to randomly select an individual from those present in each of the clusters. The fitness of the cluster representative is then evaluated using the original fitness function (2).

The fitness of the remaining individuals is estimated from the their cluster representative in proportion to the Euclidean distance from the representative. Using the Euclidean distance metric, the distance from the cluster representative of the p^{th} cluster to the q^{th} individual in the p^{th} cluster is computed as follows:

$$d\left(\underline{x}_p^{\text{rep}}, \underline{x}_p^q\right) = \sqrt[2]{\sum_{k=1}^{n} \left| \tilde{x}_{pk}^{\text{rep}} - \tilde{x}_{pk}^q \right|^2} \tag{5}$$

In order to make the clustering model independent of the dimension of the problem, and thus ensuring its effectiveness on the four test problems, which vary from a dimension of 57 for hydrogen through to 1380 for kerosene, the Euclidean distances need to scaled such that they all belong to [0,1]. This is accomplished by dividing the Euclidean distance given in (5) by the maximum theoretical distance as follows:

$$\tilde{d}\left(\underline{x}_p^{\text{rep}}, \underline{x}_p^q\right) = \frac{d\left(\underline{x}_p^{\text{rep}}, \underline{x}_p^q\right)}{\sqrt[2]{n}} \tag{6}$$

The final stage in the application of the clustering algorithm is to compute the indirect fitness of a non-representative cluster individual based on its distance from the cluster representative. A simple but effective form for this indirect evaluation is as follows:

$$\frac{\text{Fit}\left(\underline{x}_p^q\right)_{\text{indirect}}}{\text{Fit}\left(\underline{x}_p^{\text{rep}}\right)_{\text{direct}}} = \left(1 - \tilde{d}\left(\underline{x}_p^{\text{rep}}, \underline{x}_p^q\right)\right) \tag{7}$$

4 Numerical Results

As outlined in section 2.3 the two clustering-based GAs will each be tested on four different mixture problems and their results compared to those obtained with a standard GA (s1GA) that uses population sizes of $n_{\text{pop}} = 50$ and $n_{\text{child}} = 60$. The *k*-means clustering GA (kmGA) and the Ward's method GA (WmGA) both use population sizes equal to those of the s1GA, and a value of $n_{\text{cluster}} = 10$ is chosen in each case. This value for n_{cluster} strikes a good compromise between the quality of solution obtained and the overall computational requirements of the GA. The computational expense of clustering the population and evaluating the indirect fitnesses at each generation is negligible for both the kmGA and the WmGA when compared to the cost of evaluations involving the original fitness function (2). Thus, since only $n_{\text{cluster}} = 10$ evaluations of the original fitness function (2) are made at each generation for both the kmGA and the WmGA, an approximately six-fold decrease in operational runtime is observed over the s1GA for an equivalent number of generations performed. It is worth noting that the clustering of the population in the WmGA is significantly more computationally demanding than that in the kmGA but is still negligible when

compared with original fitness function evaluations, with the extent of this negligibility becoming more pronounced as the number of species increases. The results from the clustering-based GAs are also compared to those obtained from a second standard GA (s2GA) that uses population sizes of $n_{pop} = 8$ and $n_{child} = 10$. The computational requirements of the s2GA are approximately comparable to both the kmGA and the WmGA as all three perform an equal number of evaluations of the original fitness function (2) per generation.

Figures 1(a), 1(b), 1(c), and 1(d), corresponding to the hydrogen-PSR problem, the formyl-PSR problem, the methane-PSR problem, and the kerosene-PSR problem respectively, present evolutions of the average fitness value of the fittest individual at each generation as a function of generation number for the four different GAs used. In Figure 1(a), the kmGA clearly outperforms the other algorithms attaining a significantly higher fitness value after completing 1000 generations. For this problem the overall performance of the WmGA and the s1GA are comparable as they both attain a similar fitness value after 1000 generations. The overall performance of the s2GA is significantly worse than any of the other three algorithms due in most part to the reduced population sizing used in this algorithm. With only $n_{child} = 10$ individuals created at each generation in comparison to $n_{child} = 60$ for the other algorithms; the s2GA is at a significant disadvantage when it comes to effectively searching the solution domain. This performance trend for the s2GA can also clearly be observed in Figures 1(b), 1(c), and 1(d).

Similar trends to those observed in Figure 1(a) can also be observed in Figure 1(b) with the exception that now both clustering-based GAs exhibit a marked performance gain over the standard GAs. It is worth noting that when moving from the hydrogen problem (Figure 1(a)) to the formyl problem (Figure 1(b)), corresponding to an increase in the dimension of the solution domain, the relative performance of the kmGA with respect to the WmGA decreases. Figures 1(c) and 1(d) show that the WmGA now outperforms the kmGA as well as the standard GAs, demonstrating that this is a more suitable clustering algorithm when dealing with vector data sets of very high dimension.

Figures 2(a), 2(b), 2(c), and 2(d), corresponding to the same sequence of test problems as those in Figure 1, present the average percentage error in predicting the mole fractions of selected output species at a given temperature based on reaction mechanisms generated by the various GAs. The prediction errors indicate the extent to which each of the four GA-optimised mechanisms can recover the output species mole fractions that are produced from a simulation experiment involving a previously validated original reaction mechanism. All four GA-optimised mechanisms are in very good agreement with the original validated mechanism for the first three test problems at temperature $T = 1500$K as shown in Figures 2(a), 2(b), and 2(c). The agreement in the case of the kerosene problem for the same output species is in general, not as good; see Figure 2(d). This could be due in part to the difficulty of simultaneously recovering the mole fractions of 97 output species as is required for the kerosene mechanism.

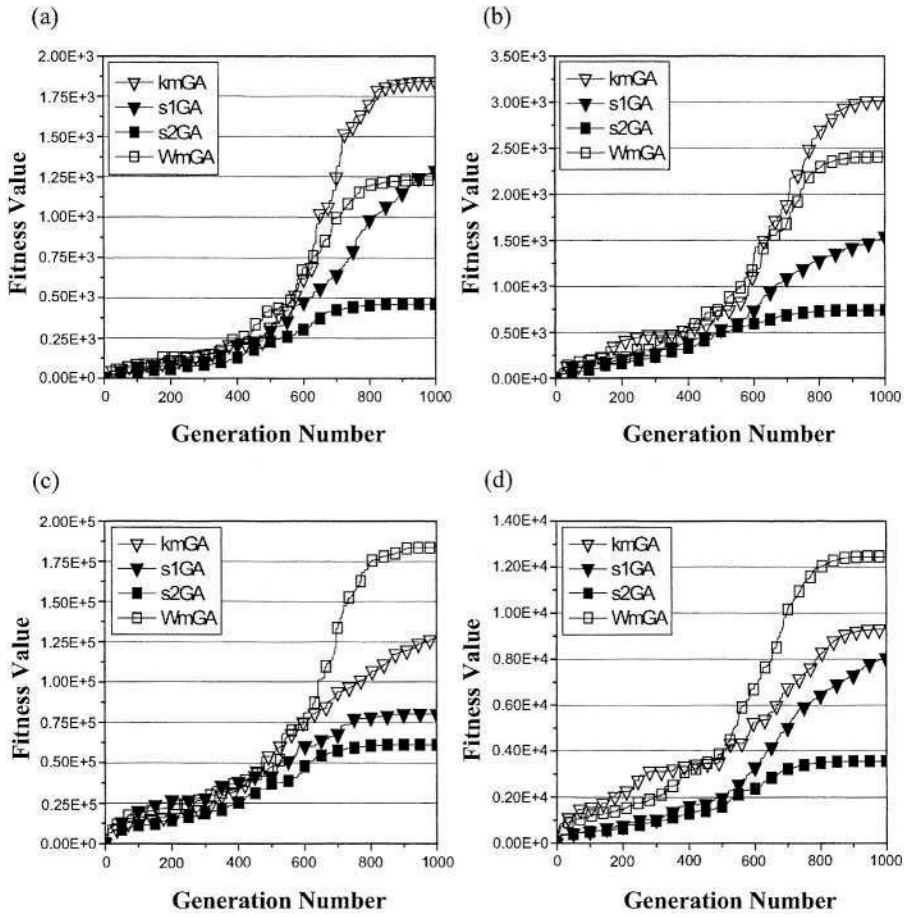


Fig. 1. The average fitness value of the fittest individual at each generation based on six differently seeded GA runs as a function of generation number for the *k*-means genetic algorithm (kmGA), the Ward's method genetic algorithm (WmGA), the first standard genetic algorithm (s1GA), and the second standard genetic algorithm (s2GA). Results are displayed for four different fuel/air mixtures, (a) hydrogen/air, (b) formyl-radical/air, (c) methane/air, and (d) kerosene/air.

The ability of the s1GA-optimised mechanism to recover the output species mole fractions for each of the four problems is in general not as good as that of the other three mechanisms, and this accounts for its relative lack of performance as shown in Figure 1. Similar validation results, although not presented here, are observed at the other PSR operating conditions.

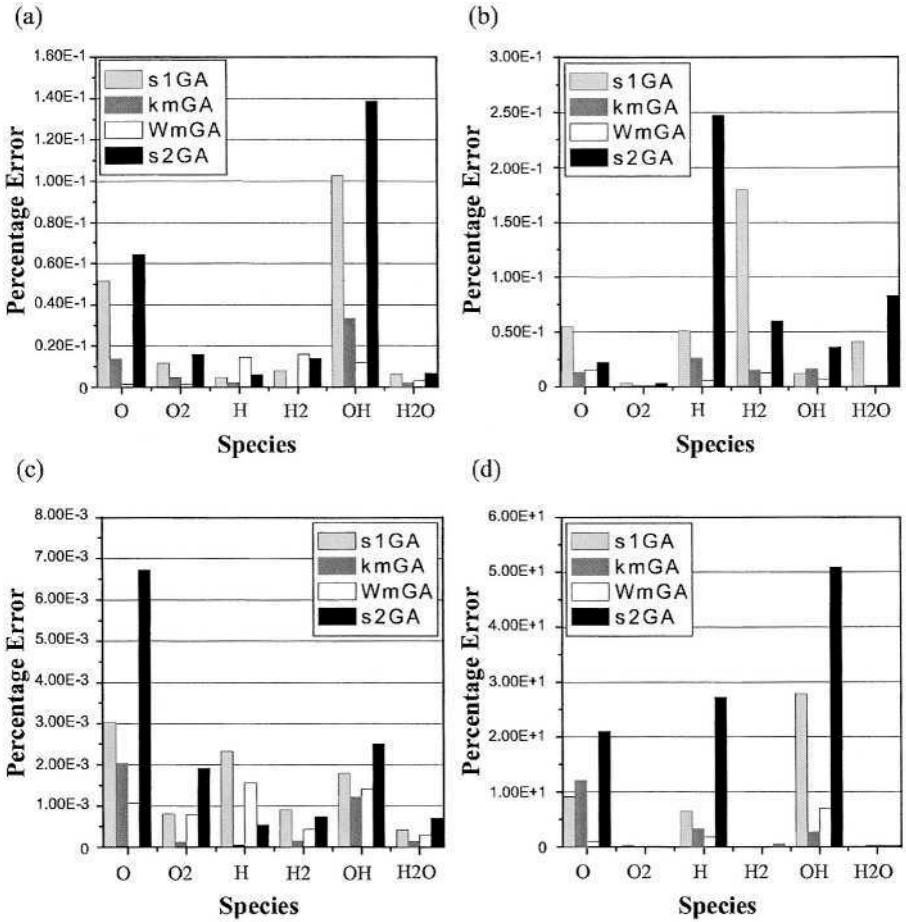


Fig. 2. The average percentage error in predicting the mole fractions of various output species based on reaction mechanisms generated by the *k*-means genetic algorithm (kmGA), the Ward's method genetic algorithm (WmGA), the first standard genetic algorithm (s1GA), and the second standard genetic algorithm (s2GA). Results are displayed for four different fuel/air mixtures, (a) hydrogen/air at temperature T = 1500K, (b) formyl-radical/air at temperature T = 1500K, (c) methane/air at temperature T = 1500K, and (d) kerosene/air at temperature T = 1080K.

5 Conclusion

In this study two efficient clustering-based genetic algorithms (GAs) have been applied to the problem of optimising reaction rate parameters for the combustion of four different fuel/air mixtures in a perfectly stirred reactor (PSR) over various operating

conditions. The clustering algorithms divide the entire population at each generation into groups based on a distance similarity measure. A representative is chosen for each group and its fitness is calculated directly using the computationally expensive original PSR fitness function. The fitness values of the remaining individuals in each cluster are calculated indirectly from their cluster representative, thus significantly reducing the total number of original PSR fitness function evaluations.

The clustering-based GAs are shown to outperform a standard GA in simulated experiments over the four test problems, whilst at the same time yielding an approximate six-fold reduction in the computational time required to complete an equal number of generations.

There remains the potential for future research applying clustering-based GAs to PSR combustion. One key area of consideration is the choice of similarity measure. At present, a Euclidean distance measure is used to assign similarity amongst individuals, but perhaps a more suitable choice involves using the rates of production associated with each species. It is these rates of production that are inexpensively calculated within the PSR program and then used to expensively determine the output species mole fractions. It is hoped that such a change of similarity measure will bring the clustering closer to the physical nature of PSR combustion.

Acknowledgements. The author would like to thank the EPSRC for their funding towards this study.

References

1. Dixon-Lewis, G., Goldsworthy, F.A., and Greenberg, J.B.: Proceedings of the Royal Society, London, A346, pp. 261-275, (1975).
2. Dagaut, P., Reuillon, M., Boetner, J.C., and Cathonnet, M.: Kerosene Combustion at Pressures up to 40atm: Experimental Study and Detailed Chemical Kinetic Modelling. Proceedings of the Combustion Institute, Vol. 25, pp. 919-926, (1994).
3. Polifke, W., Geng, W., and Döbbeling, K.D.: Optimisation of Rate Coefficients for Simplified Reaction Mechanisms with Genetic Algorithms. Combustion and Flame, Vol. 113, pp. 119-135, (1998).
4. Harris, S.D., Elliott, L., Ingham, D.B., Pourkashanian, M., and Wilson, C.W.: The Optimisation of Reaction Rate Parameters for Chemical Kinetic Modelling of Combustion Using Genetic Algorithms. Computer Methods in Applied Mechanics and Engineering, Vol. 190, pp. 1065-1083, (2000).
5. Kee, R.J., Miller, J.A., and Jefferson, T.H.: CHEMKIN: A General-Purpose, Problem-Independent, Transport Table, FORTRAN Chemical Kinetics Code Package. Sandia Report SAND80-8003, Sandia National Laboratories, (1980).
6. Glarborg, P., Kee, R.J., Grcar, J.F., and Miller, J.A.: PSR: A FORTRAN Program for Modelling Well-Stirred Reactors. Sandia Report SAND86-8209, Sandia National Laboratories, (1988).
7. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. 3rd edn. Springer-Verlag, Berlin Heidelberg New York, (1996).

8. Rasheed, K.: An Incremental-Approximate Clustering Approach for Developing Dynamic Reduced Models for Design Optimisation. Proceedings of the 2000 Congress on Evolutionary Computation, pp. 986-993, (2000).
9. Ratle, A.: Accelerating the Convergence of Evolutionary Algorithms by Fitness Landscape Approximation. Parallel Problem Solving from Nature - PPSN V, Springer-Verlag, pp. 87—96, (1998).
10. Rasheed, K., Vattam, S., and Ni, X.: Comparison of Methods for Using Reduced Models to Speed Up Design Optimisation. Proceedings of the Genetic and Evolutionary Computation Conference 2002, pp. 1180-187, (2002).
11. Gunn, S.R.: Support Vector Machines for Classification and Regression. Technical Report, University of Southampton, Faculty of Engineering and Applied Science, Department of Electronics and Computer Science, (1998).
12. Jin, Y.: Fitness Approximation in Evolutionary Computation – A Survey. Proceedings of the 2002 Genetic and Evolutionary Computation Conference 2002, pp. 1105-1112, (2002).
13. Gose, E., Johnsonbaugh, R., and Jost, S.: Pattern Recognition and Image Analysis. Prentice Hall PTR, (1996).

An Informed Operator Based Genetic Algorithm for Tuning the Reaction Rate Parameters of Chemical Kinetics Mechanisms

Lionel Elliott[1], Derek B. Ingham[1], Adrian G. Kyne[2], Nicolae S. Mera[2], Mohamed Pourkashanian[2], and Christopher W. Wilson[3]

[1] Department of Applied Mathematics, University of Leeds, Leeds, LS2 9JT, UK, {lionel,amt6dbi}@amsta.leeds.ac.uk

[2] Centre for Computational Fluid Dynamics, Energy and Resources Research Institute, University of Leeds, Leeds, LS2 9JT, UK, {fueagk,fuensm,fue6lib}@sun.leeds.ac.uk

[3] Department of Mechanical Engineering, Mappin Street, University of Sheffield, Sheffield, S1 3JD, UK, c.w.wilson@sheffield.ac.uk

Abstract. A reduced model technique based on a reduced number of numerical simulations at a subset of operating conditions for a perfectly stirred reactor is developed in order to increase the rate of convergence of a genetic algorithm (GA) used for determining new reaction rate parameters of chemical kinetics mechanisms. The genetic algorithm employed uses perfectly stirred reactor, laminar premixed flame and ignition delay time data in the inversion process in order to produce efficient reaction mechanisms that are valid for a wide range of combustion processes and various operating conditions.

1 Introduction

Reduction of engine development costs, improved predictions of component life or the environmental impact of combustion processes, as well as the improved assessment of any industrial process making use of chemical kinetics, is only possible with an accurate chemical kinetics model. However, the lack of accurate combustion schemes for high-order hydrocarbon fuels, e.g. kerosene, precludes such techniques from being utilised in many important design processes. The process of optimising the reaction rates for complex hydrocarbon fuels is a computationally intensive procedure. Therefore it is required to devise fast methods for tuning the reaction rate parameters of various types of fuels.

We note that various methods have been proposed in order to find a set of reaction rate parameters that gives the best fit to a given set of experimental data. However, in the case of complex hydrocarbon fuels, the objective function is usually highly structured, having multiple ridges and valleys and exhibiting multiple local optima. For objective functions with such a complex structure, traditional gradient based algorithms are likely to fail. Optimisation methods based upon linearisation of the objective function fall in the same category, see

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 945–956, 2004.

[1]. On the other hand, genetic algorithms are particularly suitable for optimising objective functions with complex, highly structured landscapes.

A powerful inversion technique, based on a genetic algorithm, was developed in [2] in order to determine the reaction rate parameters for the combustion of a kerosene/air mixture in a perfectly stirred reactor (PSR) and in premixed laminar flames (PREMIX). In order to obtain a reaction mechanism which can be used for a wide range of practical problems we also extend the work of [2] to include data from ignition delay time measurements. It is important to use this approach since in many practical situations the amount of experimental data available is limited and it is important to use all the available experimental data, which may come from various types of measurements. Ignition delay times are easier to measure experimentally and therefore there is a larger amount of experimental data available for various fuels.

A real coded genetic algorithm, is employed in this paper since it was found that algorithms of this type accurately model the continuous search space of numerical optimisation problems. The rate of convergence of the genetic algorithm employed is accelerated by using a reduced model implemented within the frame of the informed operators technique proposed in [3]. However, rather than using polynomial approximations, neural networks or Kriging models to build the reduced model for the informed operators, in this paper we use a reduced number of PSR numerical simulations at a subset of operating conditions to provide useful information about the fitness function landscape to the informed operators. The method used in our study is a variation of the injection island genetic algorithm (iiGA), see [4], used for problems involving finite element analysis models.

2 Description of the Problem

2.1 Reaction Rate Parameters

In a combustion process, the net chemical production or destruction rate of each species results from a competition between all the chemical reactions involving those species. In this study it is assumed that each reaction proceeds according to the law of mass action and the forward rate coefficients are of three parameter functional Arrhenius form, namely

$$k_{f_i} = A_i T^{b_i} \exp\left(-\frac{E_{a_i}}{RT}\right) \tag{1}$$

for $i = 1, \ldots, N_R$, where R is the universal gas constant and there are N_R competing reactions occurring simultaneously. The rate equations (1) contains the three parameters A_i, b_i and E_{a_i} for the i^{th} reaction. These parameters are of paramount importance in modeling the combustion processes since small changes in the reaction rates may produce large deviations in the output species concentrations. It is the possibility of the fast and efficient determination of these parameters for each reaction, based upon outlet species concentrations and ignition delay times, which is investigated in this paper.

2.2 The Perfectly Stirred Reactor (PSR) Calculations, the Laminar Premixed Flames (PREMIX) Calculations and the Ignition Delay Time (SENKIN) Calculations

Various software packages may be used for the direct calculations to determine the output species concentrations if the reaction rates are known. In this study, the perfectly stirred rector calculations are performed using the PSR computer program that predicts the steady-state temperature and composition of the species in a perfectly stirred reactor, see [5], while the laminar premixed flame structure calculations were performed using the PREMIX code, see [6] and the ignition delay time calculations are performed using the SENKIN code, see [7].

If PSR calculations are undertaken for N_S^{PSR} sets of reactor conditions then the output data which is used in the GA search procedure will consist of a set of mole concentrations, $(X_{jk},\ j = 1, ..., N_S^{PSR},\ k = 1, ..., K)$, where X_{jk} represents the mole concentration of the k^{th} species in the j^{th} set of reactor conditions.

The laminar premixed flame structure calculations were performed using the PREMIX code for burner stabilized flames with a known mass flow rate. If PREMIX calculations are performed for N_S^{PREMIX} different sets of operating conditions then the output data of the code which is used by the GA in the matching process consists of a set of species concentration profiles $(Y_{jk}(x)$, $j = 1, ..., N_S^{PREMIX}$, $k = 1, ..., K)$ and a set of burning velocity profiles $(V_j(x)$, $j = 1, ..., N_S^{PREMIX})$, where Y_{jk} is the profile of the mole concentration along the burner for the k^{th} species in the j^{th} set of operating conditions and V_j is the burning velocity profile for the j^{th} set of operating conditions.

In calculating the ignition delay times, the SENKIN code of Sandia National Laboratories [7] was used to predict the time dependent chemical kinetic behaviour of a homogeneous gas mixture in a closed system. If all the physical operating conditions are specified then for a given set of reaction rates (1) the ignition delay times $t_j,\ j = 1, ..., N_S^{SENKIN}$ are calculated together with the first-order sensitivity coefficients.

2.3 Reformulation of the Problem as an Optimisation Problem

An inverse solution procedure attempts, by calculating new reaction rate parameters that lie between predefined boundaries, to recover the profiles of the species (to within any preassigned experimental uncertainty) resulting from numerous sets of operating conditions. The inversion process aims to determine the unknown reaction rate parameters $((A_i,\ b_i, E_{a_i}),\ i = 1, ..., N_R)$ that provide the best fit to a set of given data.

The GA inversion procedure proposed seeks for the set of reaction rate parameters that gives the best fit to these measurements. In order to do so we consider three objective functions which compare predicted and measured species concentrations, burning velocities and ignition delay times:

$$f_{PSR}((A_i, b_i, E_{a_i})_{i=1,N_R}) = \left\{ 10^{-3} + \sum_{j=1}^{N_S} \sum_{k=1}^{K} W_k \frac{\left| X_{jk}^{calc} - X_{jk}^{orig} \right|}{X_{jk}^{orig}} \right\}^{-1} \qquad (2)$$

$$f_{PREMIX}((A_i, b_i, E_{a_i})_{i=1,N_R}) = \left\{ 10^{-3} + \sum_{j=1}^{N_S} \left(\frac{\|V_j^{calc} - V_j^{orig}\|_{L^2}}{\|V_j^{orig}\|_{L^2}} + \sum_{k=1}^{K} W_k \frac{\|Y_{jk}^{calc} - Y_{jk}^{orig}\|_{L^2}}{\|Y_{jk}^{orig}\|_{L^2}} \right) \right\}^{-1} \quad (3)$$

$$f_{SENKIN}((A_i, b_i, E_{a_i})_{i=1,N_R}) = \left\{ 10^{-3} + \sum_{j=1}^{N_S} \frac{\left| t_j^{calc} - t_j^{orig} \right|}{t_j^{orig}} \right\}^{-1} \quad (4)$$

where

- X_{jk}^{calc}, Y_{jk}^{calc} and V_j^{calc} represent the mole concentrations of the k^{th} species in the j^{th} set of operating conditions and the burning velocity in the j^{th} set of operating conditions using the set of reaction rate parameters$((A_i, b_i, E_{a_i})$, $i = 1, ..., N_R)$,
- X_{jk}^{orig}, Y_{jk}^{orig} and V_j^{orig} are the corresponding original values which were measured or simulated using the exact values of the reaction rate parameters,
- $\| \cdot \|_{L^2}$represents the L^2 norm of a function which is numerically calculated using a trapezoidal rule,
- W_k are different weights that can be applied to each species depending on the importance of the species.
- t_j^{calc} and t_j^{orig} represent the calculated and the measured or simulated ignition delay times in the j^{th} set of operating conditions.

It should be noted that the fitness function (2) is a measure of the accuracy of species concentrations predictions obtained by a given reaction mechanism for PSR simulations. The second fitness function (3) is a measure of the accuracy in predicting species concentrations and burning velocity profiles in the case of laminar premixed flames while the last fitness function (4) is a measure of the accuracy in predicting ignition delay times.

3 The Genetic Algorithm Optimisation Technique

3.1 A Standard Real Coded Genetic Algorithm

In this study we use a constrained genetic algorithm similar to the one proposed in [2]. The genetic operators and the parameters used for this genetic algorithm were taken to be population size $n_{pop} = 20$, number of offspring $n_{child} = 30$, non-uniform arithmetic crossover, crossover probability $p_c = 0.65$, tournament selection, tournament size $k = 2$, tournament probability $p_t = 0.8$, non-uniform mutation, mutation probability $p_m = 0.5$.

It is worth noting that when working with accurate experimental data, or numerically simulated data, the three fitness functions (2), (3) and (4) have a common global optimum given by the real values of the reaction rate coefficients or the values that were used to generate the data, respectively. Therefore the three objective functions are not conflicting and we can optimise them simultaneously using a product objective function of the form:

$$f = f_{PSR} \cdot f_{PREMIX} \cdot f_{SENKIN} \quad (5)$$

although it searches for a single common optima rather than a set of non-dominated solutions. If the experimental data is corrupted with a large level of noise then the three objective functions (2), (3) and (4) may be conflicting and then Pareto oriented techniques have to be used in order to construct the front of non-dominated solutions, i.e. a set of possible optima. In this situation other considerations have to be used to make a decision on what is the best location of the optimum.

3.2 Informed Operators Using a Subset of Operating Conditions

It is the purpose of this section to present an informed operators based genetic algorithm (IOGA) that uses a reduced subset of PSR or SENKIN operating conditions for guiding the GA search. In some optimisation tasks, the functions to optimise are analytical expressions that take a negligible amount of time to compute. However, in the case of realistic engineering design problems the optimisation function is a computationally expensive code which can take up to several hours of CPU time. Consequently it is desirable to use reduced models which are usually much faster than the actual objective function. Reduced models can be mathematical approximations, such as response surface induced using some evaluations of the original expensive model, can be simpler models relying on simpler equations like one-dimensional approximations, etc., or can be approximations based on a different level of discretisation, such as the injection island model.

Often the reaction rates retrieval problem described in section 2 requires a large number of operating conditions to be considered in the optimisation process in order to generate a widely applicable reaction mechanism. This leads to large computational times spent in evaluating one possible solution for the reaction rates. Often within the GA search these expensive evaluations are performed for an individual which is far from being an acceptable solution, and in this way much computational time is wasted on evaluations which do not provide any further information on the solution of the problem. In particular, PREMIX evaluations require a large CPU time. For complex hydrocarbon fuels the CPU time can build up to several months even if only one PREMIX operating condition is used, see [2].

In order to avoid expensive PREMIX evaluations for reaction mechanisms that are far from being an accurate solution, it is the purpose of this paper to implement a reduced model based on a subset of PSR or SENKIN operating conditions in order to pre-evaluate the GA generated solutions, and then to decide which individuals should be evaluated with the original expensive model. Thus, if the problem considered requires the evaluation of the reaction mechanism at N_S^{PREMIX} PREMIX operating conditions, N_S^{PSR} PSR operating conditions and N_S^{SENKIN} SENKIN operating conditions to obtain accurate approximations, the idea of reduced models is based on extracting information from evaluations at only $N'_{PSR} < N_S^{PSR}$ PSR operating conditions, information which is used to guide the search and to decide which solutions are evaluated with the full fitness function (5). The reduced model is implemented within the frame of informed

operators technique, see [3]. This approach has the advantage that it does not require assumptions about the accuracy of the reduced model. The assumption that computations at a subset of PSR operating conditions can accurately predict computations for a wider range of operating conditions, for PSR, PREMIX and SENKIN, does not hold in many practical problems. However in the informed operators approach the only requirement is that the reduced model is a better than random approximation of the accurate one. This approach is also simple and easy to implement into a genetic algorithm. The only difference between GA described in section 3, which will be referred to as the "standard GA" and the informed operators based GA (IOGA) is the replacements of the mutation and crossover operators with the following operators:

- **Informed mutation** To do mutation several random mutations are generated of the base point. Each mutation is generated by randomly selecting the parameters for the original mutation method. The off-springs created are evaluated using the reduced model based on N'_{PSR} PSR operating conditions. The mutation that appears best according to the reduced model is returned as the result of the mutation operator.
- **Informed crossover** To do crossover two parents are selected at random according to the original selection strategy. These two parents will not change in the course of the informed crossover operation. Several crossover are conducted by randomly selecting the internal parameters of the crossover operator. Each individual generated is evaluated with the reduced model and the best is considered to be the outcome of the crossover operator.

The number k of random mutations and crossovers applied for generating mutated or recombined children is an internal parameter of the IOGA and various values are investigated in this paper.

It should be noted that the informed operator technique employed in this study resembles the local search memetic algorithms discussed in [8]. However, such real coded memetic algorithms require many evaluation of the full fitness function to perform the local search and this is not feasible for computationally expensive problems such as the one illustrated here. Instead, in this study we employ in the search process a reduced model that incorporates problem specific knowledge. Other reduced models using response surfaces constructed from polynomial approximations, neural networks, Kriging models or support vector machines can be employed and they are very efficient for problems with reduced numbers of parameters, see [3]. However, for the problem of retrieving reaction rate coefficients, the number of unknown parameters is very large. For the combustion of hydrogen/air mixture there are 57 parameters to be identified, for $N_R = 19$ reactions, but for complex aviation fuels the number of unknown parameters can exceed 1000, which prohibits the application of such reduced models. However, the reduced model using subsets of operating conditions as described in this study can be applied also for complex reaction mechanisms with large numbers of reactions. Moreover, other response surface models which do not incorporate problem specific knowledge may lead the algorithms toward false optima. The reduced model employed in this study extracts information

from a reduced set of operating conditions, thus enabling a higher degree of confidence in its predictive capabilities.

4 Numerical Results

In order to test the informed operators based genetic algorithm inversion procedure the test problem considered is the recovery of the species concentration profiles predicted by a previously validated hydrogen/air reaction mechanism for a wide range of operating conditions. The reaction scheme used to generate the data involves $K = 9$ species across $N_R = 19$ reactions.

The test problem considered is the combustion of a hydrogen/air mixture at constant atmospheric pressures, $p = 1\,atm$ and $p = 10\,atm$. A set of $N_S^{PSR} = 10$, $N_S^{PREMIX} = 10$ and $N_S^{SENKIN} = 10$ different operating PSR, PREMIX and SENKIN operating conditions have been considered corresponding to various changes to the temperature from $1000K$ to $2000K$. The inlet composition is defined by a specified fuel/air equivalence ratio, $\Phi = 1.0$.

Several different mechanisms are generated using a GA optimisation process as follows

- **STANDARD** is a chemical reaction mechanism obtained by optimising the fitness function (5) by the standard GA described in section 3 without using informed operators.
- **IOGA-PSR** is a chemical reaction mechanism obtained by optimising the fitness function (5) by the IOGA described in section 3.2 using informed operators based on a subset of $N'_{PSR} = 4$ PSR operating condition evaluations.
- **IOGA-SENKIN** is a chemical reaction mechanism obtained by optimising the fitness function (5) by the IOGA described in section 3.2 using informed operators based on a subset of $N'_{SENKIN} = 4$ SENKIN operating conditions evaluations.

The operating conditions used for the informed operators are taken to be the bounds of the range used as operating conditions in the full optimisation process, i.e. $p = 1\,atm$ and $p = 10\,atm$ and $T = 1000K$ and $T = 2000K$. The GA generated mechanisms are used to produce estimations of the output species concentrations and these estimations are compared with the predictions given by the original mechanism (ORIGINAL), i.e. the mechanism which was used to simulate the data numerically.

Figure 1 presents the fitness functions obtained by the IOGA using PSR or SENKIN information with $k = 3$ in comparison with the fitness function obtained by the standard GA. It should be mentioned that all the results presented in this paper are obtained as the average of five different GA runs for five different sequences of random numbers. The graphs have been scaled such that they represent the results obtained in similar CPU times. The time unit that appears of the x-axis of Figure 1 is equivalent to the CPU time required to perform one whole generation in the standard GA and and is approximately 318 seconds of CPU time.

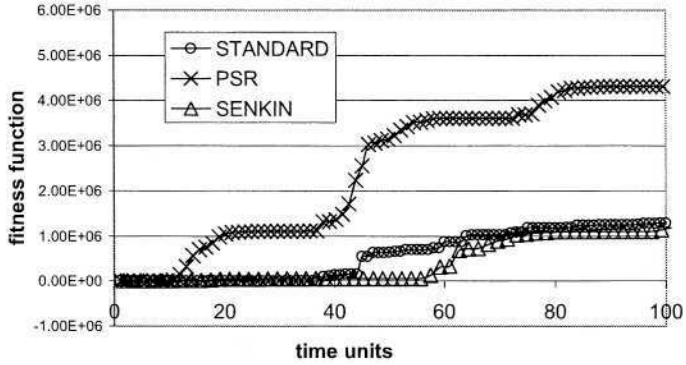


Fig. 1. The fitness functions for the standard GA and the informed operators based GA using perfectly stirred reactor evaluations (PSR) and ignition delay time evaluations (SENKIN).

It can be seen in Figure 1 that the IOGA-PSR clearly outperforms the standard GA. The PSR informed GA reaches the maximum value attained by the standard GA in about 1/5 of the CPU time required in the standard GA. However, the informed GA using SENKIN operating conditions is performing even worse than the standard GA. This can be explained by the fact that the SENKIN simulations only provide information about ignition delay times, while both PSR and SENKIN use information about the species concentration. The SENKIN object function (4) has multiple global optima since there are several sets of parameters that give exact predictions of ignition delay times but not all of these sets are accurate in predicting the species concentration. If a GA is employed to optimise only the SENKIN fitness function (4) then the reaction mechanism generated, GA-SENK, is very accurate in predicting the ignition delay time but is poor in predicting species concentrations. Figure 2 presents the O mole fractions in a PSR predicted by the GA-SENK mechanism in comparison with the original mechanism and the GA-ALL mechanism obtained by optimising the global fitness function (5) for PSR, PREMIX and SENKIN. It can be seen that if a mechanism is only optimised for SENKIN then it does not produce good results for PSR and PREMIX. However, mechanisms optimised for PSR and/or PREMIX, produce reasonably accurate results for SENKIN simulations as well, see [2]. Similarly, if an objective function based only on SENKIN evaluations is used in the informed operators then the search is possibly guided toward some false optima which correspond to reaction mechanisms efficient for ignition delay time predictions but not for species concentrations. This explains why the IOGA-PSR clearly outperforms the standard GA while the SENKIN informed operators do not improve the convergence rate of the GA. Therefore we will focus in the remainder of this study on PSR informed operators.

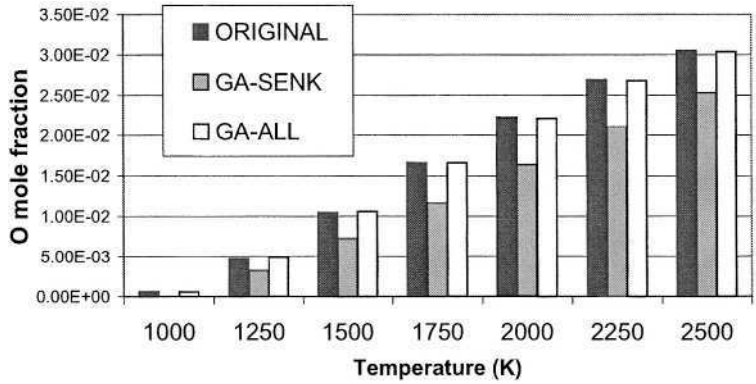


Fig. 2. The mole fraction of O in a perfectly stirred reactor at pressure $p = 1\,atm$ and air fuel ration $\Phi = 1.0$ for various temperatures as predicted by reaction mechanisms obtained by GA optimisations against ignitions delay time measurements (GA-SENK), or against ignition delay time measurements and species concentrations (GA-ALL) compared with the mole fractions predicted by the original mechanism.

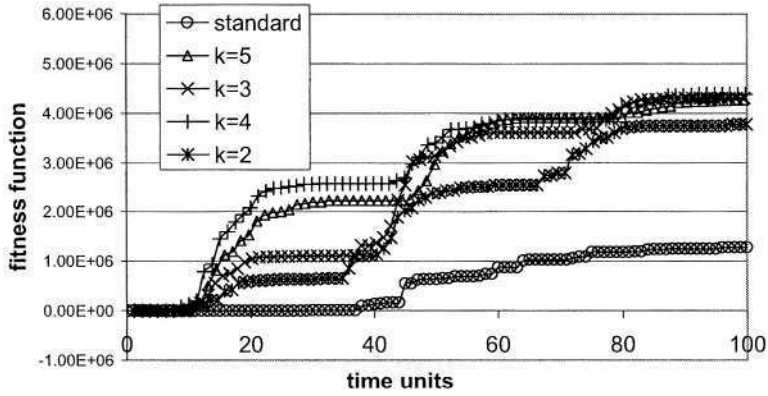


Fig. 3. The fitness functions for standard GA and the informed operators based GA using various numbers of mutations and crossovers, k, for informed operators based on $N'_{PSR} = 4$ PSR evaluations.

Figure 3 presents the evolution of the fitness function (5) values for the IOGA-PSR algorithm for various values of k, the number of trial mutations and crossovers that are generated and evaluated using the reduced model. It can be seen for all the values of k, IOGA outperforms the standard GA, reducing the CPU time required to obtain the same maximum value by a factor of up to 5. Similar results are obtained for larger values of k, see Figure 4 which presents the corresponding fitness functions for $k \in \{10, 15, 20\}$. Various values of the

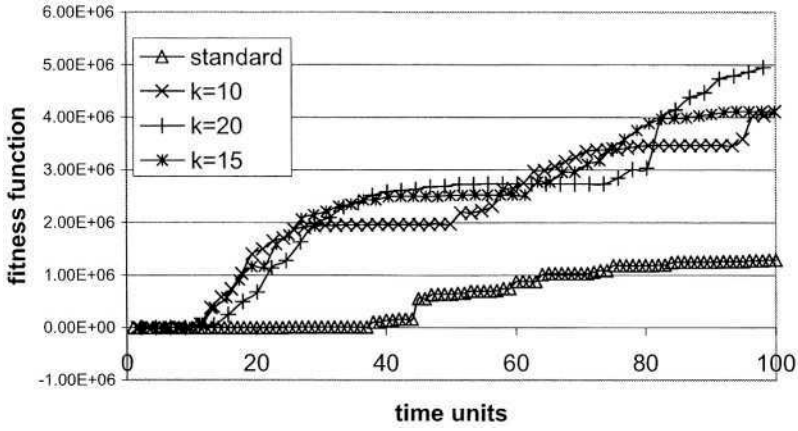


Fig. 4. The fitness functions for the informed operators based GA using large numbers of mutations and crossovers, $k \in \{10, 15, 20\}$, for informed operators based on $N'_{PSR} = 4$ PSR evaluations, in comparison with the fitness obtained for the standard GA.

parameter k have been investigated and the best results were observed if the time spent in evaluations using the fitness function (5) is balanced with the time spent in reduced model evaluation, i.e. if k is chosen such that within a GA generation, the time spent on reduced model evaluations is from one to four times the CPU time spent on evaluations with the fitness function (5).

4.1 Validation of the IOGA Generated Mechanisms for Various Combustion Processes

It is the purpose of this section to validate the mechanisms generated by the IOGA by comparing their predictions for PSR, PREMIX and SENKIN simulations with those obtained by the original mechanism. Figure 5 presents the O mole fractions obtained in PSR simulations as predicted by the original mechanism and the two mechanisms generated by the standard GA and the IOGA-PSR for $k = 5$, for various operating conditions, namely pressure $p = 1atm$, air fuel ration $\Phi = 1.0$ and various temperatures T = 1000*K*, ..., 2000*K*. It can be seen that similar species concentrations are predicted by all three mechanisms. To provide additional validation of the results obtained by the inversion procedures presented in this paper, the predicted ignition delay times are computed using all the reaction mechanisms generated in this study. Figure 6 compares the ignition delay times predicted by the reaction mechanisms generated in this study with the ignition delay times generated by the original mechanism. We note that both the the standard GA and the IOGA produced mechanisms reproduce the same ignition delay times as the original mechanism. Although not presented here it is reported that the GA generated mechanisms produce accurate results also for PREMIX numerical simulations. The IOGA generated mechanism was tested at various other operating conditions and it was found to produce accurate pre-

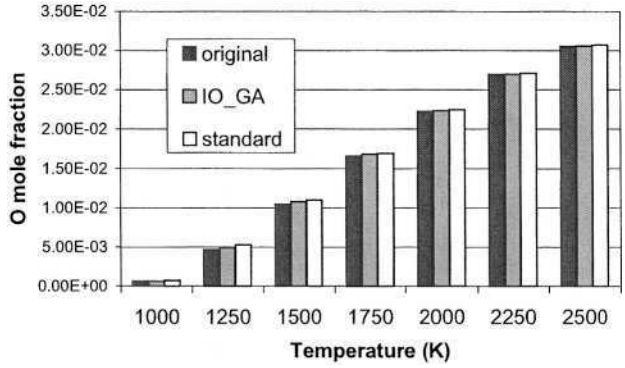


Fig. 5. The mole fraction of O in a perfectly stirred reactor at pressure $p = 1\ atm$ and air fuel ration $\Phi = 1.0$ for various temperatures as predicted by reaction mechanisms obtained by the standard GA and the informed operators based GA, compared with the mole fractions predicted by the original mechanism.

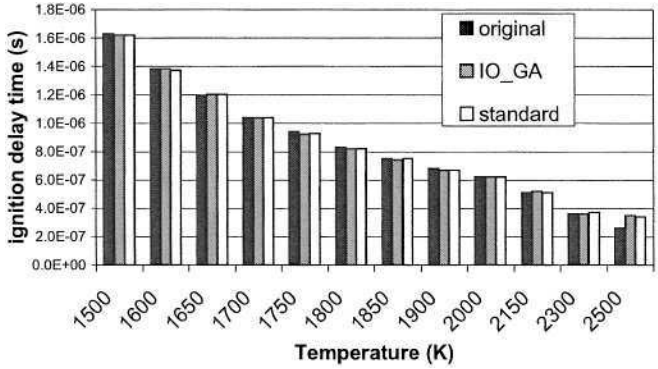


Fig. 6. The ignition delay times at pressure $p = 10\,atm$ and air fuel ration $\Phi = 1.0$ for various temperatures $T \in \{1500, ..., 2500\}$ as predicted by reaction mechanisms obtained by the standard GA and the informed operators based GA, compared with the mole fractions predicted by the original mechanism.

dictions for a wide range of operating conditions and various type of numerical simulations of combustion processes.

5 Conclusions

In this paper the problem of retrieving reaction rate coefficients for the combustion hydrogen/air mixtures under various operating conditions has been investi-

gated using a reduced model based GA. Informed GA operators using a subset of PSR simulations were used to speed-up the optimisation process. The reaction mechanism developed was tested for several test examples. The new reaction rate coefficients generated by the IOGA were found to successfully match mole fraction values in PSR simulations and burning velocity and species profiles in PREMIX simulations, as well as ignition delay time predicted by the original starting mechanism over a wide range of operating conditions.

It was found that informed operators technique using subsets of PSR simulations is very efficient in guiding the GA operators toward more promising regions of the search space. The numerical results presented have shown that the computational time required to achieve an efficient reaction mechanism in the GA can be reduced up to five times by employing the informed operators technique.

References

1. Frenklach M., Wang H, Rabinowitz J. (1992) Optimisation and analysis of large chemical kinetic mechanisms using the solution mapping method - combustion of methane, Prog. Energy Combust. Sci., 18: 47- 73.
2. Elliott, L., Ingham, D.B., Kyne, A.G., Mera, N.S., Pourkashanian, M. and Wilson, C.W., (2003), Optimisation of reaction mechanisms for aviation fuels using a multi-objective genetic algorithm, in Cantú-Paz et al. (Eds.), Genetic and Evolutionary Computation Conference - GECCO 2003, Springer Verlag, Berlin, 2046-2057.
3. Rasheed, K, Vattam, S., Ni, X., (2002), Comparison of Methods for Using Reduced Models to Speed Up Design optimisation, in (eds) E. Cantu-Paz et al., Genetic and Evolutionary Computation Conference 2002 (GECOO2002), 1180-1187.
4. Eby, D., Averill, R. C., Gelfand, B., Punch, W., Mathews, O. and Goodman, E.D. (1997), An Injection Island GA for Flywheel Design optimisation, in (eds) Zimmermann, J.H. et al. Eufit '97 – 5th European Congress on Intelligent Techniques and Soft Computing, Verlag Mainz, Wissenschaftsverlag, 687–691.
5. Glarborg, P., Kee, R.J., Grcar, J.F., Miller, J.A.: PSR, (1988), A FORTRAN program for modelling well-stirred reactors, Sandia National Laboratories Report SAND86-8209.
6. Kee, R.J., Grcar, J.F., Smooke, M.D., Miller, J.A., (1985), A FORTRAN program for modelling steady laminar one-dimensional premixed flames, Sandia National Laboratories Report SAND85-8240.
7. Lutz, A.E., Kee, R.J. and Miller, J.A., SENKIN: A FORTRAN program for predicting homogeneous gas phase chemical kinetics with sensitivity analysis, Sandia National Laboratories Report SAND87-8248, (1987).
8. Lozano, M., Herrera, F., Krasnogor, N. and Molina, D., (2004), Real-coded Memetic Algorithms with Crossover Hill-Climbing, Evolutionary Computation, in print.

Transfer of Neuroevolved Controllers in Unstable Domains

Faustino J. Gomez and Risto Miikkulainen

Department of Computer Sciences, University of Texas, Austin, TX, 78712 USA

Abstract. In recent years, the evolution of artificial neural networks or *neuroevolution* has brought promising results in solving difficult reinforcement learning problems. But, like standard RL methods, it requires that solutions be discovered in simulation and then be transferred to the real world. To date, transfer has been studied primarily in mobile robot problems that exhibit stability. This paper presents the first study of transfer in an unstable environment. Using the double pole balancing task, we simulate the process of transferring controllers from simulation to the real-world, and show that the appropriate use of noise during evolution can improve transfer significantly by compensating for inaccuracy in the simulator.

1 Introduction

Neuroevolution (i.e. methods that evolve artificial neural networks) can potentially be used to automatically design controllers for domains such as autonomous robotics and aerospace where designing controllers by more conventional means can be very difficult. Unfortunately, evaluating populations of candidate controllers through direct interaction with real world systems is too time consuming and often dangerous. Instead, controllers must first be evolved in a *simulation environment* that approximates the real-world *target environment,* and then be transferred to that the target environment. In order for transfer to be successful, the controllers must be robust enough to tolerate discrepancies that inevitably exist between the two environments. Otherwise, a controller that performs well in simulation may perform poorly, or not at all, when placed in the real world.

While there is a substantial body of work addressing transfer in the mobile robot domain [1,2,3,4,5,6], almost all transfer experiments have been conducted with robots (e.g. Khepera) and environments that are relatively "transfer friendly." Most significantly, robots like the Khepera are stable: in the absence of a control signal the robot will either stay in its current state or quickly converge to a nearby state. Consequently, a robot can often perform competently in the real world as long as its behavior is preserved qualitatively after transfer. This is not the case with a great many systems of interest such as rockets, aircraft, and chemical plants that are inherently unstable. In such environments, the controller must continuously output a precise control signal to maintain system equilibrium and avoid failure. Therefore, controllers for unstable systems may be less amenable to techniques that have worked for transfer in robots.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 957–968, 2004.

In this paper, we study the problem of transferring controllers for unstable, non-linear systems with the aim of providing a more general understanding of the transfer process. This paper represents the first attempt to systematically study the transfer of neuroevolved controllers outside of the mobile robot domain.

The next section describes the methodology used to study transfer, section 3 presents our experimental results, section 4 analyzes the robustness of transferred solutions, and in section 5 we provide some analysis of our results. Sections 6 and 7 contain a broader discussion of transfer and future directions, and our conclusions, respectively.

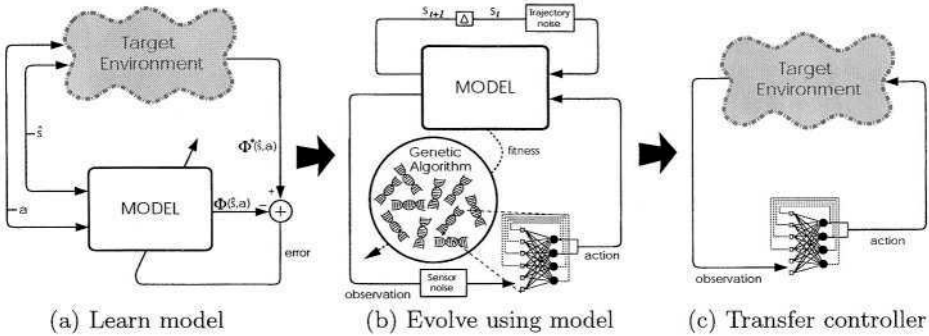


Fig. 1. The model-based neuroevolution approach, (a) Step 1: Learn a model of the target environment; in this case, the double pole balancing system. The model (e.g. a supervised neural network) receives the a state $\hat{s}$ and an action a as its input and produces a prediction of the next state $\Phi(s,a)$ as its output. The error between the prediction and the correct next state $\Phi^*(s,a)$ is used to improve the prediction, (b) Step 2: Evolve a controller using the model instead of the target environment. The boxes labeled "trajectory noise" and "sensory noise" add uniform noise to the signal passing through them. These noise sources make controllers more robust and facilitate transfer, (c) Step 3: Test the controller on the target environment.

2 Transfer Methodology

For the purpose of studying transfer, we have chosen double pole balancing as the test domain because it embodies the essential elements of unstable systems while being simple enough to study in depth, and it has been studied extensively, but in simulation only. The problem consists of balancing two poles that are hinged to a cart, riding on a finite stretch of track, by applying a force to the cart at regular intervals. This problem is challenging because the interaction between the poles makes the system highly non-linear and requires very precise control [7].

Figure 1 shows the three steps that constitute the approach used in our experiments. First, the target environment is modeled with a supervised neural network to produce a simulation environment (step 1). Second, the controller is evolved in the simulation environment (step 2). Third, this controller is transferred to the target environment (step 3). In the usual application of this method,

the target environment is the physical system, and nothing needs to be known about it as long as its interaction with a controller can be observed. However, in this study, we treat the analytical system (i.e. the differential equations [7] normally used in the literature as a reinforcement learning benchmark) as if it were a real two-pole system (i.e. the target environment), and the simulation environment is an approximation of it. This way it is possible to systematically study conditions for successful transfer in a controlled setting and gain insights that would be difficult to obtain from an actual transfer to a physical system.

Sections 2.1, 2.2, describe in detail steps 1 and 2 of the approach, respectively. Step 3 is presented in section 3.

2.1 Learning the Simulation Model

While it is possible to develop a tractable mathematical model for some systems, most real world environments are too complex or poorly understood to be captured analytically. A more general approach is to learn a *forward model* using observations of target environment behavior. The model, Φ, approximates the discrete-time state-transition mapping, Φ^*, of the target environment:

$$\hat{s}_{t+1} = \Phi^*(\hat{s}_t, a_t), \quad \forall \hat{s} \in \hat{S}, a \in A, \tag{1}$$

where $\hat{S}$ is the set of possible states in the target environment, and $\hat{s}_{t+1}$ is the state of the system some time in the future if action a is taken in state $\hat{s}_t$. The function Φ^* is in general unknown and can only be sampled. Using Φ, interaction with the target environment can be simulated by iterating

$$s_{t+1} = \Phi(s_t, \pi(s_t)), \tag{2}$$

where π is the control policy and $s \in S$ are the states of the simulator.

In modeling the two-pole system we followed a standard neural network system identification approach [8]. Figure 1a illustrates the basic procedure for learning the model. The model is represented by a feed-forward neural network (MLP) that is trained on a set of state transition examples obtained from Φ^*. State-action pairs are presented to the model which outputs a prediction $\Phi(\hat{s}, a)$ of the next state $\Phi^*(\hat{s}, a)$. The error between the correct next state and the model's prediction is used to improve future predictions using backpropagation.

The MLP allows us to construct a model quickly, using few assumptions about Φ^*, and relatively few examples of correct behavior. Furthermore, when modeling a real system these examples can be obtained from an existing controller (i.e. the controller upon which we are trying to improve).

A training set of 500 example state transitions was generated by randomly sampling the state space, $\hat{S} \subset \Re^6$, which consists of the cart position and velocity $(x, \dot{x})$, each pole's angle from vertical (θ_1, θ_2), and its angular velocity $(\dot{\theta}_1, \dot{\theta}_2)$, and the action space, $A \subset [-10, 10]$ Newtons, of the two-pole system. For each point $(\hat{s}^i, a^i)$, $\hat{s} \in \hat{S}, a \in A$, the next state $\Phi^*(\hat{s}^i, a^i)$ was generated 0.02 seconds (i.e. one time-step). The training set is then $\{ ((\hat{s}^i, a^i), \Phi^*_\delta(\hat{s}^i, a^i)) \}, i = 1..500$.

The accuracy of the model was measured in terms of the average squared error in the model's prediction across a separate test set of 500 examples.

Our model is global in the sense that a single function approximator is used to represent Φ for the entire state space, but it is local in terms of its temporal predictions. If the model is used to make predictions one time-step into the future, then the predictions will be very accurate. However, if the model is iterated (i.e. the output is fed back into the input) it will quickly diverge from the target environment.

In order to study the effect of model error on transfer, the weights of the model were saved at five points during training to obtain a set of models with different levels of error: $M_{0.002}, M_{0.005}, M_{0.008}, M_{0.010}, M_{0.025}$, where M_e is a model with error e. The models were tested extensively using trajectories from the target environment to verify that the model error (i.e. the error based on the test set), was a reliable indicator of the relative amount of prediction error that is actually encountered by controllers during evolution. That is, for two models M_x and M_y, $x > y$ implies that the prediction errors of M_x will generally be greater than those of M_y. It is important that this be true so that error measure e can be used to rank the models correctly for the experiments that follow.

2.2 Evolving with the Model

If the simulation environment perfectly replicates the dynamics of the target environment, then a model-evolved controller will behave identically in both settings, and successful transfer is guaranteed. Unfortunately, since such an ideal model is unattainable, the relevant questions are: (1) how do inaccuracies in the model affect transfer and (2) how can successful transfer be made more likely given an imperfect model? To answer these questions we evolved controllers at different levels of model error to study the relationship between model error and transfer, and to find ways in which transfer can be improved.

The controllers were evolved using the Enforced SubPopulations[1](ESP; [9]) neuroevolution method under three conditions:

No noise. The controllers interact with the model according to equation 2, and a controller is evaluated by having it control the double pole system starting in the center of a 4.8 meter track with the short pole (0.1m) at vertical and the long pole (1m) leaning at 4.5 degrees from vertical. Every 0.02 seconds the controller receives the state of the environment and outputs a control force, [-10,10]N, that is applied to the cart. The fitness used to evolve the controllers is the number of time steps before either pole exceeds ±36 degrees or the cart goes off the edge of the track. The task is considered to be solved when a controller can balance both poles for 100,000 time-steps.

This experiment provides a baseline for measuring how well controllers transfer from simulation to the target environment. Each of the five models

[1] The particular neuroevolution algorithm used is not important in this study and is only mentioned for completeness.

$(M_{0.002}, M_{0.005}, M_{0.008}, M_{0.010}, M_{0.025})$ was used in 20 simulations for a total of 100 simulations.

Sensor noise. The controllers are evolved as above except that their inputs are perturbed by noise. The agent-environment interaction is defined by

$$s_{t+1} = \Phi(s_t, \pi(s_t + v)), \tag{3}$$

where $v \in \Re^6$ is a random vector with components $v(i)$ distributed uniformly over the interval $[-\alpha, \alpha]$. Sensor noise can be interpreted as perturbations in the physical quantities being measured, imperfect measurement of those quantities, or, more generally, a combination of both. This kind of noise has been shown to produce more general and robust evolved controllers in the mobile robot domain [3,10].

In figure 1b, the box labeled "sensor noise" represents this noise source. As in the "no noise" case, 20 simulations were run for each of the five models, this time with three sensor noise levels: 5%, 7%, and 10%, for a total of 300 simulations.

Trajectory noise. In this case, instead of distorting the controllers' sensory input the noise is applied to the dynamics of the simulation model. The agent-environment interaction is defined by

$$s_{t+1} = \Phi(s_t, \pi(s_t)) + w, \tag{4}$$

where $w \in \Re^6$ is a uniformly distributed random vector with $w(i) \in [-\beta, \beta]$. At each state transition the next state is generated by the model and is then perturbed by noise before it is fed back in for the next iteration. Although, a similar effect could be produced by adding noise to the actuators, $s_{t+1} = \Phi(s_t, \pi(s_t) + w)$, equation 4 ensures that the the trajectory of the model remains stochastic even in the absence of a control signal, $\pi(s) = 0$.

Eight different levels of trajectory noise $\{0.5\%, 0.6\%, \ldots, 1.2\%\}$ were used. As with the sensor noise simulations, 20 simulations were run for each of the five models at each trajectory noise level, for a total of 800 simulations.

3 Transfer Results

After evolving controllers with the model, the solution found by each simulation was submitted to a single trial in the target environment. Our criteria for successful transfer was whether a controller could perform the same task in the target environment that it performed in simulation. That is, balance the poles for 100,000 time steps using the same setup (i.e. initial state, pole lengths, etc.) used during evolution, but with no noise. Later, in section 4, we apply a more rigorous standard.

Sensor Noise

Figure 2a shows the transfer results for controllers evolved with sensor noise. The plot shows the average number of time-steps that the networks at each level of model error could control the target environment. Successful transfers were

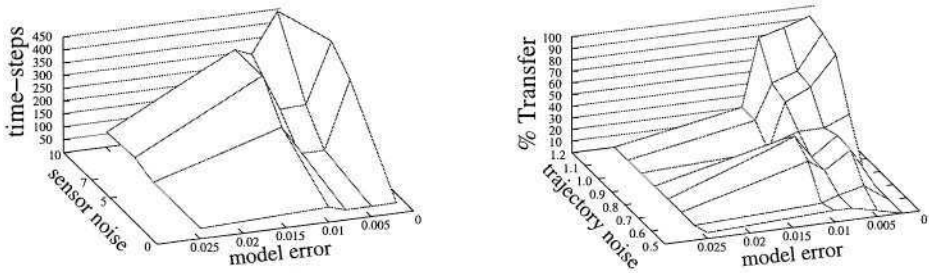


Fig. 2. Transfer results. Plot (a) shows the number of time steps the controllers evolved at different levels of model error and sensor noise could balance the poles after transfer to the target environment. Each curve corresponds to a different level of noise. Sensor noise improves performance slightly but does not produce successful transfers (a transfer was considered successful if the network could control the system for 100,000 time steps). Plot (b) shows the percentage of controllers that successfully transferred to the target environment for different levels of model error and trajectory noise. Lower error (i.e. more accurate local model) and higher trajectory noise produces controllers more likely to transfer.

very rare in this case (occurring only twice in all 400 simulations); in effect, the surface plot shows the performance of networks that did not transfer.

Without noise, all of the transferred controllers failed within 50 time steps i.e. almost immediately. As sensor noise was added, performance improved significantly, especially when model error was low, but controllers were still far from being able to stabilize the target environment. Therefore sensor noise, even at high levels, is not useful for transfer in this kind of domain.

Trajectory Noise

On the other hand, trajectory noise had a much more favorable effect on transfer. Because successes were frequent, a different plot is used. Figure 2b shows the percentage of networks that transferred successfully for different levels of model error and trajectory noise. Each curve corresponds to a different level of trajectory noise, with the "0" curve again indicating transfer without noise. The plot shows that trajectory noise compensates for model error and improves transfer. To achieve very reliable transfer, low model error needs to be combined with higher levels of trajectory noise. The best results were achieved with 1.2% trajectory noise and a model error of 0.002, yielding a successful transfer rate of 91%.

4 Evaluating Controller Robustness

During evolution a controller can only be exposed to a subset of the conditions that can exist in the real world. Therefore, how will it perform under conditions that differ from those encountered during evolution? In this section, we analyze the quality of transferred controllers in three respects: (1) generalization to

novel starting states, (2) resistance to external disturbances, and (3) resistance to changes in the environment. Since only the controllers that were evolved with trajectory noise transferred successfully, this analysis pertains only to those controllers. Also, because different levels of trajectory noise had different transfer rates (figure 2b), additional controllers were evolved at each noise level until a minimum of 20 successfully transferred controllers were obtained for each level.

Generalization

To measure how trajectory noise affects the performance of transferred controllers in across a broad range of initial states, 625 test cases were generated by allowing the state variables $x, \dot{x}, \theta_1$, and $\dot{\theta}_1$ to take on the values: 0.05, 0.25, 0.5, 0.75, 0.95, scaled to the appropriate range for each variable ($5^4 = 625$). These ranges were ±2.16m for x, ±1.35m/s for $\dot{x}$, ±3.6 deg for θ_1, and ±8.6deg/s for $\dot{\theta}_1$. A controller is awarded a point for each of the 625 different initial states from which it is able to control the system for 1000 time steps. This test, first introduced in [11], has become a standard for evaluating the generality of solutions in pole balancing. A high score indicates that a solution has competence in a wide area of the state space.

Figure 3a is a visualization of a controller's performance on this test. Each dot in the upper half of the graph identifies the number of time steps the poles could be balanced for a particular start state, i.e. test case. Each test case is denoted by a unique setting of the four state variables $x, \dot{x}, \theta_1, \dot{\theta}_1$ in the lower half of the graph (θ_2 and $\dot{\theta}_2$ were always set to zero). Drawing a vertical line through the graph at a given dot gives the state variable values for that case. For example, the balance time for case 245 ($x = -1.080$, $\dot{x} = 1.215$, $\theta_1 = 0.031$, $\dot{\theta}_1 = 0.135$; the labeled point in the figure) is 129 time steps for this particular controller, which solved 353 of the 625 cases.

Figure 4a shows the quantitative results for the generalization test. Solutions evolved with more trajectory noise generalize to a larger number of novel initial states. Recall that the fitness function used here simply measures the number of time steps the poles stay balanced. It is therefore almost devoid of domain knowledge, and places no restriction (bias) on the control strategy. Still, the use of trajectory noise produces solutions that generalize to a large number of cases in the target environment that were not experienced in the simulation environment.

External Disturbances

The generalization test measures how well networks behave across a large region of the state space. Another important question is: how well will these solutions operate in "unprotected" environments where external disturbances, not modeled in simulation, are present? To answer this question, the networks were subjected to external forces that simulate the effect of wind gusts buffeting the cart-pole system. Each network was started in the center of the track with the long pole at 4.5 degrees (the small pole was vertical). After every 5,000 time steps, a force was applied to the cart for 2 time steps (0.04 sec). The magnitude of this pulse was started at 1 Newton and increased by 1 Newton on each pulse.

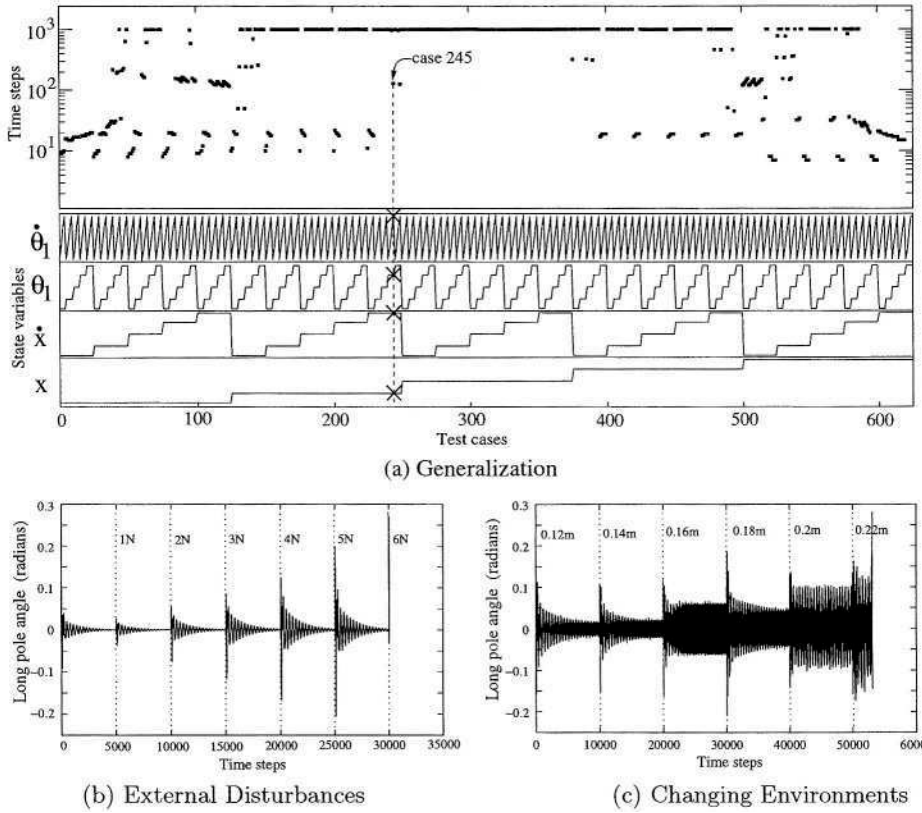
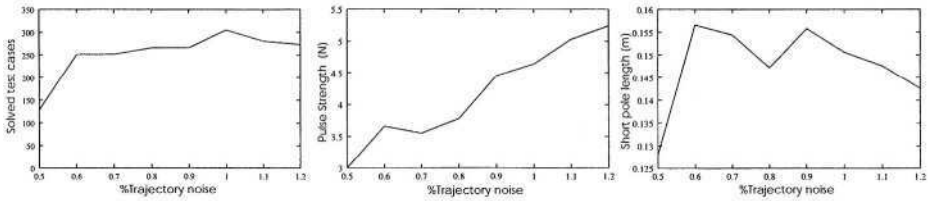


Fig. 3. Examples of controller behavior on the robustness tests. The plots quantify the performance of a particular controller evolved with 1.0% trajectory noise on the three robustness tests. In plot (a) the lower half of the plot shows the values of $x, \dot{x}, \theta_1$, and $\dot{\theta}_1$ for each of the 625 cases. The upper half shows the number of time steps the poles were balanced starting from each case. This controller solved 353 of the cases. Plot (b) shows the trajectory of the long pole angle for a disturbance test. Each vertical dotted line marks the onset of an external pulse of the shown intensity (in Newtons). The controller is able to recover from disturbances of up to 5 Newtons, but fails when the force reaches 6 Newtons. Plot (c) shows the trajectory of the long pole angle for a changing environment test. Each vertical dotted line marks each lengthening of the short pole. The controller is able to balance the poles using increasingly wide oscillations as the short pole is lengthened. However, at 0.22 meters (i.e. almost double the original length) the system becomes unstable.

Figure 3b shows the angle of the long pole for a typical disturbance test. In the first 5,000 time steps the controller stabilizes the system from its unstable initial state. After the first pulse hits, the system recovers rapidly. As the pulse is strengthened, the controller takes longer to recover, until the force becomes too large causing the system to enter into an unrecoverable state. Figure 4b shows the average maximum force that could be handled for each level of trajectory noise. Above 1.1% noise controllers could withstand an average pulse of over 5



(a) Generalization (b) External Disturbances (c) Changing Environments

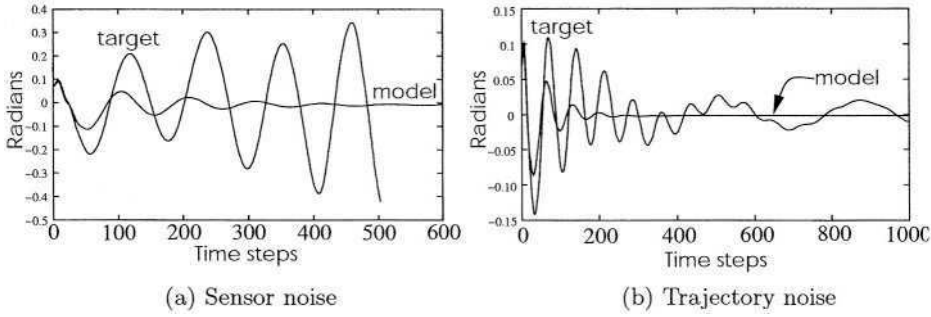
Fig. 4. Robustness results. The plots show the performance of successfully transferred controllers evolved at different levels of trajectory noise on the three different tests for robustness. Plot (a) shows the number of generalization test cases, on average, solved by the controllers. In plot (b), the y-axis is the average force in Newtons that a network could withstand. In plot (c), the y-axis is the average short pole length that could be balanced for 10,000 time steps. High levels of trajectory noise generally yield more robust behavior in transferred controllers.

Newtons. This magnitude of disturbance is very large: it is more than half of the maximum force that can be exerted by the controller. Higher levels of trajectory noise lead to networks that are less affected by external forces.

Changes to the Environment

An interesting and convenient property of the double pole system is that it is more difficult to control as the poles assume similar lengths. By lengthening the short pole during testing, we can measure how well the controllers can cope with changes to the dynamics of the environment such as those that occur in real systems through mechanical wear, damage, and modifications. For this test, each network was started controlling the system with a short pole length of 0.12 meters, 0.02 meters longer than the length used during evolution. If after 10,000 time steps the trial had not ended in failure, the cart-pole system was restarted from the same initial state but with the short pole elongated by another 0.02 meters. This cycle was repeated until the network failed to control the system for 10,000 time steps. A network's score was the short pole length that it could successfully control for 10,000 time steps. Figure 3c shows the behavior of the long pole during one of these tests.

Figure 4c shows the average short pole length versus trajectory noise. For low noise levels ($\approx$0.5%), the networks adapt only to relatively small changes (0.03m, or 30%). At and above 0.6%, networks tolerate up to as much as a 57% increase (to 0.157m) on average. These are very large changes because not only do the dynamics change, but the system also becomes harder and harder to control with each extension of the short pole. Trajectory noise prepares controllers for conditions that can deviate significantly from those to which they were exposed during evolution, and, consequently, produces high-performance solutions that can better endure the rigors of operating in the real world.



(a) Sensor noise (b) Trajectory noise

Fig. 5. Comparison of controller behavior before and after transfer. The graphs show typical behavior (long pole angle) of a network evolved with sensor noise (a) versus a network evolved with trajectory noise (b), when controlling either the model or the target environment. With sensor noise (a), the trajectories coincide initially, but after about 50 time steps the target environment becomes unstable, although the model is quickly damped into equilibrium. With trajectory noise (b), the trajectories also do not coincide after about 50 time steps but the network is still able to control the target environment because it has been evolved to cope with a range of deviations from the model trajectory.

5 Analysis of Transfer Results

Whenever a behavior is learned in one environment some adaptation is necessary to preserve an acceptable level of performance in another, different environment. In a sense, using noise pre-adapts the agent to a range of possible deviations from the simulation model. It is not surprising that controllers evolved without noise did not transfer, even when model error was low. But why is there such a disparity between the sensor and trajectory noise results?

Let us take a typical solution evolved with sensor noise and use it to control the simulation environment *without* sensor noise. The resulting behavior is shown by the "model" curve in figure 5a. Sensor noise forces evolution to select against strategies (such as swinging the poles back and forth) that are too precarious when the state of the environment is uncertain. Consequently, high-performance solutions quickly stabilize the environment by dampening oscillations. This behavior, however, does not help networks control the target environment. Because the target environment reacts differently from the model, a policy that would stabilize the simulation environment can soon cause the target environment to diverge and become unstable (the "target" curve in figure 5a).

This result differs from the experience of many researchers (e.g. [3,10]) that have used sensor noise to make robots more robust and facilitate transfer. Unlike robot navigation tasks, where small inaccuracies in actuator values do not affect the transferred behavior qualitatively, the pole balancing domain requires very precise control and is much less forgiving because it is inherently unstable; a sequence of poor or even imprecise actions can quickly lead to failure.

The success of controllers evolved with trajectory noise can be explained by looking at the space of state trajectories that are possible by making noisy state

transitions. If we let $\{u_i\}$ and $\{l_i\}$ be the sequence of states that form the upper and lower bound on the possible state trajectories for a given policy π, such that

$$u_{i+1} = \max_{s_i \in [l_i, u_i]} \|\Phi(s_i, \pi(s_i)) + \overline{w}\|, \tag{5}$$

$$l_{i+1} = \min_{s_i \in [l_i, u_i]} \|\Phi(s_i, \pi(s_i)) - \overline{w}\|, \tag{6}$$

where $\overline{w} = \text{argmax}_w \|w\|$, $s_1 = l_1 = u_1$, then every state $\|l_i\| \leq \|s\| \leq \|u_i\|$ can be visited by some sequence of state transitions generated by equation 4. State sequences $\{u_i\}$ and $\{l_i\}$ form a trajectory envelope for a particular controller.

All of the trajectories that can be followed from an initial state s_1 will fall inside this envelope. Although each network evaluation involves only a single trajectory of finite length, the number of state transitions in a successful trial (100,000) is large enough that it represents a relatively dense sampling of the trajectory space. So the controller is effectively trained to respond to a range of possible transitions at each state, much like the principle underlying robust control theory where a controller is designed not for a single system but for a whole family of systems that is likely to contain the actual system of interest.

6 Discussion and Future Work

The transfer experiments revealed that a relatively accurate model alone may not be sufficient to ensure transfer when the target environment is inherently unstable. Transforming the deterministic model into a stochastic one by injecting trajectory noise, however, improved the transferred controllers significantly. They were able to cope with a wide variety of conditions that where not present during evolution. These experiments demonstrate how such transfer can be achieved in principle. For a large class of problems involving unstable systems this technique should prove useful.

Although these experiments have focused on direct transfer, in domains where unsuccessful controllers can be tested a simple post-transfer search can be used to find a successful controller. We found that many of the controllers that did not transfer in our trajectory noise experiments were "near-misses" that could be adapted to the target environment quite easily through local random search in the weight space. In domains where testing is not feasible, it may be possible to determine the stability of the controller through analytical tools such as those developed by Suykens [12] and Kretchmar [13] for robust neurocontrol. Only controllers that pass a stability test would then be allowed to interact with the environment, thereby reducing the chance of failure.

The pole balancing problem studied in this paper is a good surrogate for challenging problems in the real world. In ongoing work, we are applying neuroevolution to the active guidance of finless (i.e. unstable) sounding rockets, a problem with dynamics similar to that of the inverted pendulum. The techniques described in this paper will be used to transfer controllers to the actual rocket for launch. Similar applications should be possible in other domains, leading to more accurate and robust non-linear control in the real world.

7 Conclusion

The transfer experiments demonstrated that combining an empirical model with trajectory noise can make transfer possible even in unstable environments. Our results will hopefully provide a better understanding of the conditions necessary to make neuroevolved controllers sufficiently robust, and ultimately allow neuroevolution to be a viable controller design method for real-world systems.

References

1. Nolfi, S., Floreano, D.: Evolutionary Robotics. MIT Press, Cambridge (2000)
2. Ficici, S.G., Watson, R.A., Pollack, J.B.: Embodied evolution: A response to challenges in evolutionary robotics. In Wyatt, J.L., Demiris, J., eds.: Eighth European Workshop on Learning Robots. (1999) 14–22
3. Jakobi, N., Husbands, P., Harvey, I.: Noise and the reality gap: The use of simulation in evolutionary robotics. In: Proceedings of the Third European Conference on Artificial Life, Springer-Verlag (1995)
4. Miglino, O., Lund, H.H., Nolfi, S.: Evolving mobile robots in simulated and real environments. Artificial Life **2** (1995) 417–434
5. Chavas, J., Corne, C., Horvai, P., Kodjabachian, J., Meyer, J.A.: Incremental evolution of neural controllers for robust obstacle-avoidance in khepera. In: EvoRobots. (1998) 227–247
6. Smith, T.M.C.: Blurred vision: Simulation-reality transfer of a visually guided robot. In: EvoRobots. (1998) 152–164
7. Wieland, A.P.: Evolving controls for unstable systems. In Touretzky, D.S., Elman, J.L., Sejnowski, T.J., Hinton, G.E., eds.: Connectionist Models: Proceedings of the 1990 Summer School. San Francisco, CA: Morgan Kaufmann (1990) 91–102
8. Barto, A.G.: Connectionist learning for control. In 3rd, W.T.M., Sutton, R.S., Werbos, P.J., eds.: Neural Networks for Control. MIT Press, Cambridge, MA (1990) 5–58
9. Gomez, F.J.: Robust Nonlinear Control through Neuroevolution. PhD thesis, Department of Computer Sciences, The University of Texas at Austin (2003)
10. Reynolds, C.W.: Evolution of obstacle avoidance behaviour: using noise to promote robust solutions. In Kenneth E. Kinnear, J., ed.: Advances in Genetic Programming. MIT Press (1994)
11. Dominic, S., Das, R., Whitley, D., Anderson, C.: Genetic reinforcement learning for neural networks. In: Proceedings of the International Joint Conference on Neural Networks (Seattle, WA), Piscataway, NJ: IEEE (1991) 71–76
12. Suykens, J., Moor, B.D., Vandewalle, J.: Stabilizing neural controllers: a case study for swinging up a double inverted pendulum. In: International Symposium on Nonlinear Theory and its Application (NOLTA'93). (1993) 411–414
13. Kretchmar, R.M.: A Synthesis of Reinforcement Learning and Robust Control Theory. PhD thesis, Department of Computer Science, Colorado State University, Fort Collins, Colorado (2000)

Evolving Wavelets Using a Coevolutionary Genetic Algorithm and Lifting

Uli Grasemann and Risto Miikkulainen

Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712, USA
{uli, risto}@cs.utexas.edu

Abstract. Finding a good wavelet for a particular application and type of input data is a difficult problem. Traditional methods of wavelet design focus on abstract properties of the wavelet that can be optimized analytically but whose influence on its real-world performance are not entirely understood. In this paper, a coevolutionary genetic algorithm is developed that searches the space of biorthogonal wavelets. The lifting technique, which defines a wavelet as a sequence of digital filters, provides a compact representation and an efficient way of handling necessary constraints. The algorithm is applied to a signal compression task with good results.

1 Introduction

Since their introduction over two decades ago, wavelets have proven useful in an extremely broad range of applications, from data compression to numerical simulation, from signal de-noising to seismology.

Most of these applications, however, share the same problem: Although using wavelets is relatively straightforward in most cases, finding a good wavelet for a particular job or type of input data is not. Often, a good wavelet can make all the difference. The choice of a wavelet is usually made on a somewhat ad-hoc basis, by trying out several well-known wavelets and keeping the best one. Traditional methods of wavelet design rely on abstract properties like filter length or order of approximation that can be analytically optimized, but whose influence on the performance of a wavelet is not entirely understood.

In this paper, a method for adapting biorthogonal wavelet bases to a specific application and class of input data is presented. The method is based on a coevolutionary genetic algorithm closely related to the *Enforced Sub-Populations (ESP)* neuroevolution method introduced in [1]. The algorithm encodes wavelets as a sequence of *lifting steps,* i.e. finite digital filters that can be combined to form a wavelet. Lifting provides a compact representation as well as an efficient way of handling necessary constraints.

The next section gives a brief tour of wavelets, lifting and wavelet-based data compression. Section 3 discusses related work, and section 4 describes the algorithm and the evaluation function used. In section 5, the algorithm is applied to the task of compressing cubic spline curves with good results.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 969–980, 2004.

2 Background

Both wavelet theory and the application of wavelets in data compression are complex and evolving subjects. This section gives a brief high-level overview of these topics. For more details on classical wavelet theory, see [2]; [3] contains an introduction to lifting, and [4] covers the basics of wavelet-based data compression.

2.1 Wavelets

Wavelets are a mathematical tool for representing and approximating functions hierarchically. At the heart of wavelet theory, there is a single function ψ, called the *mother wavelet.* Any function can be represented by superimposing translated and dilated versions of ψ.

The translates and dilates of ψ are denoted by $\psi_{j,i}$, where i and j are the translation and dilation parameter. We are focusing on the discrete case where i and j only take on integer values. The $\psi_{j,i}$ can be computed from the mother wavelet as

$$\psi_{j,i}(x) = 2^{\frac{j}{2}}\ \psi(2^j x - i). \tag{1}$$

Figure 1 shows an example wavelet and a translated and dilated version of that wavelet.

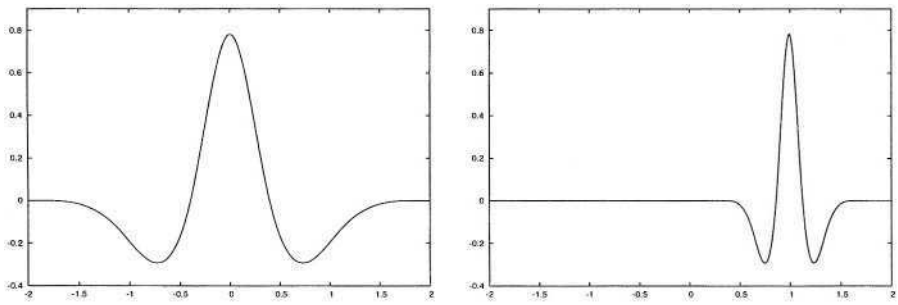


Fig. 1. A wavelet ψ and the translated and dilated version $\psi_{1,2}$. The wavelet shown is a cubic spline wavelet.

All the translates of ψ for a specific dilation j span a function space W_j:

$$W_j = span\{\ \psi_{j,i} \mid i \in \mathbb{Z}\}. \tag{2}$$

The W_j are called *wavelet spaces* or *detail spaces,* because each of them adds a level of detail to the wavelet representation of a function. All of the detail spaces combined form a basis in which any function can be expressed.

The process of decomposing a function into *wavelet coefficients* (a scaling factor for each of the $\psi_{j,i}$) is called *wavelet transform.* If the parameters i and j

take on dicrete values, we have a *discrete wavelet transform* or *DWT,* essentially leading to a finite number of coefficients.

In order to compute the DWT of a function f, we need to find one wavelet coefficient $\gamma_{j,i}$ for each $\psi_{j,i}$, such that

$$f = \sum_{j,i} \gamma_{j,i} \psi_{j,i}. \tag{3}$$

If a wavelet basis (i.e. the set of all $\psi_{j,i}$) is *orthogonal,* then the $\gamma_{j,i}$ are given by

$$\gamma_{j,i} = \langle f, \psi_{j,i} \rangle = \int_{-\infty}^{\infty} f(x) \overline{\psi_{j,i}}(x) dx, \tag{4}$$

where the bar denotes the complex conjugate. Otherwise, a *dual* wavelet $\widetilde{\psi}$ is necessary such that ψ and $\widetilde{\psi}$ together are *biorthogonal,* which basically means that the transform must be invertible. We can then use $\widetilde{\psi}$ for determining the wavelet coefficients (eqn. 4), and the original wavelet for the inverse DWT (eqn. 3). Note that an orthogonal wavelet is just a special case of a biorthogonal one where $\widetilde{\psi} = \psi$.

2.2 Filters and the Fast Wavelet Transform

Computing a wavelet transform in the way just described is expensive and cumbersome. However, an algorithm called the *Fast Wavelet Transform* or *FWT* allows computing the wavelet coefficients by recursively applying a pair of digital filters to the data, much like the Fast Fourier Transform reduces a DFT to computing a few finite sums.

A *digital filter* can be defined by giving a sequence of real numbers called *filter coefficients.* It is applied by convolution with an input sequence. A filter is said to have *finite impulse response (FIR),* if its coefficients are non-zero only on a finite range. A FIR filter can be represented by a finite number of coefficients and the index of the leftmost non-zero coefficient.

It turns out that the filter pair used in the FWT uniquely determines the mother wavelet ψ and also (in the biorthogonal case) the dual wavelet $\widetilde{\psi}$. In order to define a valid wavelet transform, a filter pair must be *complementary,* which is the same as saying that the associated wavelet must be biorthogonal.

Ensuring that a filter pair is complementary and that the individual filters are finite are the basic contraints in wavelet design.

2.3 Lifting

The Lifting scheme, introduced by Sweldens [5] in 1996, offers an easy way to construct complementary filter pairs. A finite filter, called a *lifting step,* is used to generate a new filter pair from an existing pair. Multiple lifting steps can be applied consecutively. In [3], Sweldens and Daubechies proved two important properties of lifting:

- Lifting preserves biorthogonality, i.e. if the original filter pair is complementary, then so is the new pair, no matter what lifting step is applied.
- Any wavelet with finite filters can be expressed as a sequence of lifting steps. Starting with the trivial wavelet transform (called the *Lazy Wavelet*), all possible wavelets can be reached by applying a finite number of finite-length lifting steps.

These two properties make lifting a powerful tool for constructing new wavelets.

2.4 Wavelets and Signal Compression

In signal compression applications, wavelets are used as the transformation part of a transform coder. Figure 2 shows the general structure of a transform coder.

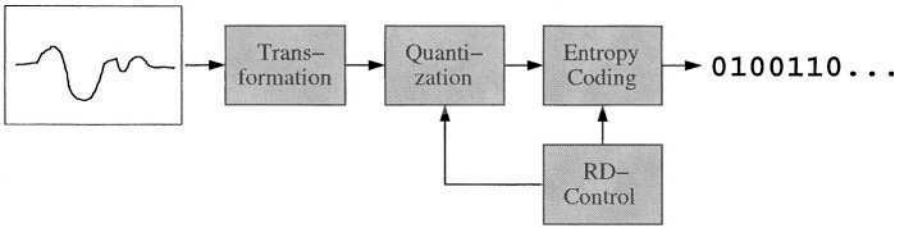


Fig. 2. The structure of a transform coder. The signal is first decorrelated using an invertible transform, and then quantized and entropy coded. The rate-distortion (RD) unit controls the quantization to minimize the distortion within the available bit rate.

The first step is to apply an invertible transform to the data in order to decorrelate it. Examples of such transforms are the discrete cosine transform (DCT) and the discrete wavelet transform (DWT). The performance of a transform coder depends largely on how well the transform decorrelates the signal. A well decorrelated signal consists mainly of values close to zero.

The transform coefficients are then quantized, i.e. expressed using symbols from a finite alphabet, and entropy coded, using as little space or bandwidth as possible. The rate-distortion (RD) unit controls the quantization in order to achieve minimal distortion within the available bit rate.

Examples for transform coders are the DCT-based JPEG standard and the wavelet-based JPEG2000 standard.

3 Related Work

The idea of adaptive wavelet bases is not new. Traditionally, this has been done by so-called *dictionary* methods, where a basis is selected from an overcomplete set of predefined functions called *atoms.* Examples of such methods are the *best*

basis algorithm [6] and *wavelet packets* [7]. [8] and [9] use evolutionary algorithms for adaptive dictionary methods. Note that dictionary methods do not come up with new wavelets. Instead, they simply try to select the best combination of atoms to form a basis.

The lifting technique has provided new ways of adapting wavelets. For example, Claypoole et al. [10] use lifting to adapt a wavelet transform to a given signal by optimizing data-based prediction error criteria.

In [11] and [12], genetic algorithms are used for the design of digital filters.

Several stochastic optimization techniques have been applied to the design of wavelets. Monro and Sherlock [13] use simulated annealing to find wavelets with balanced uncertainty in space and frequency. Hill et al. [14] use a genetic algorithm to find windowed trigonometric functions that can be used in a continuous wavelet transform.

To our knowledge, the combination of lifting and genetic algorithms has not been explored previously.

4 Evolving Wavelets

In section 2.3, two interesting properties of lifting were mentioned:

- It preserves biorthogonality, and
- any wavelet can be expressed as a sequence of lifting steps.

These two properties make sequences of lifting steps an effective representation for wavelets in a genetic algorithm, because (1) any random sequence of lifting steps will encode a valid (i.e. biorthogonal) wavelet, and (2) any wavelet can be represented using the genetic code. In this section, a coevolutionary genetic algorithm that evolves wavelets encoded as lifting steps will be described.

Algorithm. The coevolutionary GA used is closely related to the *Enforced Sub-Populations (ESP)* neuroevolution algorithm introduced by Gomez and Miikkulainen [1]. ESP evolves a number of populations of individual neurons in parallel. In the evaluation phase, ESP repeatedly selects one neuron from each sub-population to form candidate networks. The fitness of a particular neuron is the average fitness of all networks in which it participated.

This concept can be easily applied to wavelet evolution: Several populations of lifting steps are evolved in parallel, and are randomly combined to form wavelets, which are then evaluated. No migration or crossover occurs between sub-populations. Figure 3 describes the algorithm in more detail.

Representation. A lifting step is represented as a fixed-length sequence of floating point numbers for the filter coefficients, and a single integer for the leftmost index of the filter. Using a fixed number of fixed-length steps limits the number of wavelets that can be represented. However, it also limits the length of the wavelet filters, which is a desirable effect. Also, most wavelets used in practice can be factored into a small number of short lifting steps [3], so this limitation is unlikely to interfere with finding good solutions.

WAVELET-ESP

input: N, the number of sub-populations
L, the lengths of the lifting filters
M, the size of each sub-population
P, the mutation rate

1. INITIALIZE
Create M filters of length L for each of the N sub-populations, and randomize them.

2. EVALUATE
Select N lifting steps, one from each sub-population, and evaluate the resulting wavelet. Add the fitness to the cumulative fitness of all participating steps. Repeat until each step has been evaluated 10 times on average.

3. RECOMBINE
Rank the lifting steps in each sub-population by their average fitness. Each step in the top quartile is recombined with a higher-ranking step. The offspring is mutated with probability P and replaces the lowest-ranking half of each sub-population.

4. REPEAT
Repeat the EVALUATE-RECOMBINE cycle for a fixed number of generations.

Fig. 3. The ESP algorithm applied to wavelets. Several populations of lifting steps are evolved in parallel, and are combined in the evaluation phase to form wavelets.

Initialization. Each chromosome is initialized by setting the values of the coefficients to random values from a gaussian distribution with mean 0 and variance 0.5, and setting the leftmost index of each filter to a random integer between -2 and 2. This reflects the values commonly found in lifting steps.

Crossover. The crossover operator performs simple one-point crossover on the coefficients. The integers representing the leftmost indices of the parent filters are randomly assigned to the children.

Mutation. A chromosome is mutated by adding low-variance gaussian noise to a random filter coefficient and/or adding ±1 to the integer representing the leftmost index.

Fitness Evaluation. In signal compression, the ideal measure of fitness would be the performance in an actual transform coder as described in section 2.4. However, there are two problems with this approach. First, evaluating a wavelet using a transform coder is almost prohibitively expensive. Second, in order to make a fair comparison between two wavelets, either the available number of

bits needs to be fixed and the resulting distortion used as a fitness measure, or vice versa. Both options are inexact and expensive for actual transform coders.

Figure 4 shows a definition of the evaluation function. It is an idealized version of a transform coder: Instead of quantizing and entropy-coding the wavelet coefficients, it uses only a certain percentage of the coefficients for reconstruction and sets the rest to zero. This is much less expensive and allows choosing the compression ratio exactly, which means that the resulting distortion can be used directly as a fitness measure. Villasenor et al. [15] have used a similar but even simpler method to evaluate wavelets with good results.

EVALUATION FUNCTION

input: D, the input data
W, a candidate wavelet
R, the compression ratio

return: The fitness of W.

1. TRANSFORM
Transform D using the wavelet W.

2. COMPRESS
Sort the resulting wavelet coefficients. Keep only the largest $R \times |D|$. Set the rest to zero.

3. RECONSTRUCT
Perform an inverse transform using W and the altered wavelet coefficients.

4. MEASURE THE ERROR
Measure the resulting distortion (L_2 error) E and return 1/E.

Fig. 4. The evaluation function is an idealized version of a transform coder: Instead of quantizing and entropy-coding the wavelet coefficients, it uses only part of the coefficients for reconstruction and sets the rest to zero.

5 Experiments

The algorithm described in the previous section was evaluated on the task of compressing cubic splines, i.e. 1D-sequences of data sampled from cubic spline curves. Input sequences of length 256 were generated from 16 random control points from the interval [–1, 1]. Figure 5 shows an example input curve. This type of input data has several advantages:

1. The optimal compression ratio is known, because the number of random control points is known.

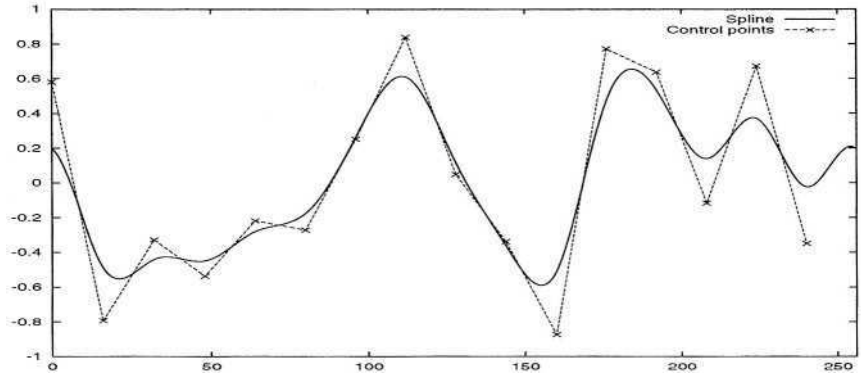


Fig. 5. An example for the input data used in the experiment. A spline curve is created from 16 random control points.

2. Optimal wavelets for the task are known (the class of cubic spline wavelets, like the one shown in figure 1).
3. Cubic Splines are known to be an acceptable model for a wide range of input data, including images. Some of the best known wavelets for image compression are spline wavelets [4].

5.1 Methodology

The algorithm was run 30 times for 40 generations. After every generation, the best wavelet found so far was evaluated on a test set of 50 spline curves. Fresh test and training data were used for every run.

Preliminary experiments were conducted to determine the best parameter settings. The algorithm turned out to be very robust; similar results were obtained for a wide range of parameters.

The following parameters were used for the reported results: The population size was 400 for each sub-population, which means that 4000 evaluations took place each generation. The algorithm evolved 6 sub-populations in parallel, each of which contained lifting steps of length 2. The mutation rate was 0.4. The evaluation function used 26 of the 256 wavelet coefficients for reconstruction of the signal, i.e. the compression ratio was roughly 10:1.

5.2 Results

Figure 6 shows the learning curve (the average performance on the test set) averaged over 30 runs. The bars are 95% confidence intervals for the expected performance.

The horizontal lines show the performance of two well-known orthogonal wavelets [16], and the biorthogonal 9/7 wavelet introduced by Antonini [17], probably the most popular wavelet for image compression. These comparisons

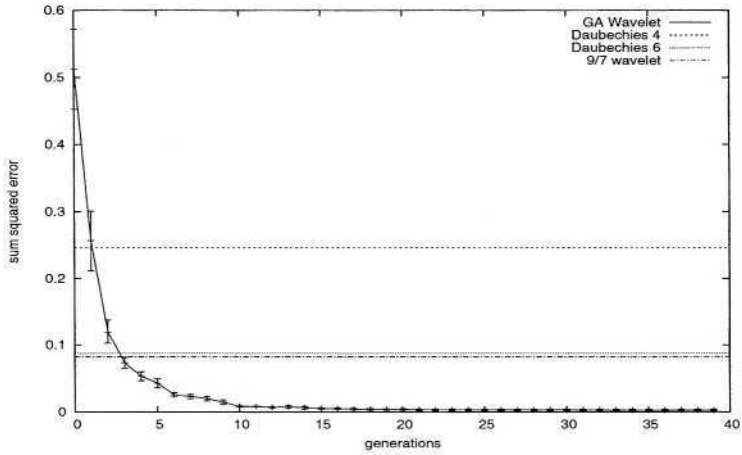


Fig. 6. The learning curve (average performance on the test set) averaged over 30 runs. The bars are 95% confidence intervals. The horizontal lines show the performance of some well-known wavelets.

are intended to put the performance of the wavelets into perspective. As mentioned before, the optimal wavelets for this kind of input data are cubic spline wavelets like the one in figure 1. Using such a wavelet would result in zero compression error. Note, however, that for real-world applications, an optimal wavelet is generally not known.

Figure 7 shows some of the best wavelets found by the GA after 40 generations. Note that these wavelets are locally very similar to the wavelet in figure 1, but also that all of them differ in overall shape. These are characteristics shared with actual cubic spline wavelets.

Figure 8 illustrates how the compression performance develops during a typical run. The wavelets used are the winners of generations 1, 5, 10, 20 and 40. The plots on the left compare an example spline curve with the reconstructed version after 10:1 compression. The plots on the right show a rate-distortion curve for the same wavelet, i.e. the sum squared compression error as a function of the percentage of wavelet coefficients used for reconstruction.

The best wavelet of the first generation performs very poorly. Keep in mind, however, that no evolution has taken place yet; we are simply looking at the best wavelet from a random population.

After five generations, the compressed curve is already much closer to the original. The rate-distortion curve shows that the error has been reduced by over an order of magnitude. By generation 10, the compressed curve is even closer to the original curve, and by generation 20, the left plot no longer shows a difference between the two. Between generations 20 and 40, the error decreases by another order of magnitude.

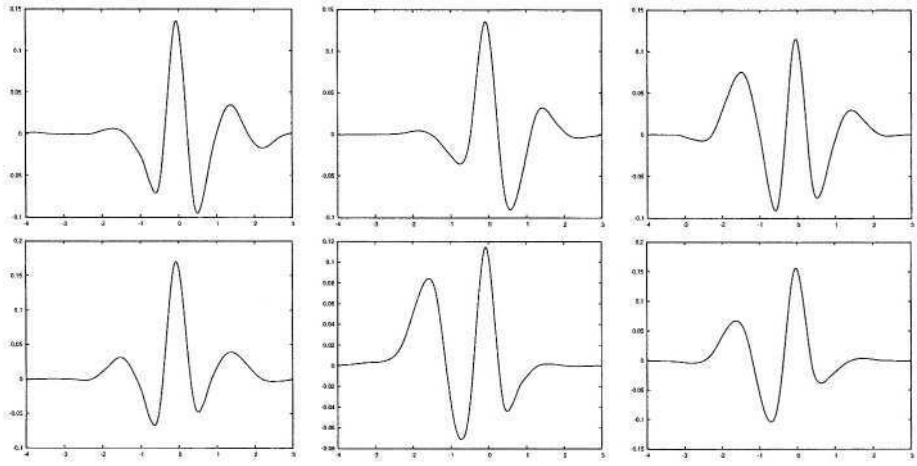


Fig. 7. Some of the best wavelets found in different runs of the GA. All 30 runs produced near-optimal wavelets.

The GA has clearly found a near-optimal wavelet for the compression of cubic splines. The winner wavelets from all 30 runs show similar performance.

6 Future Work

The next step will be to test the algorithm on real-world data. Current work focuses on the evaluation of the algorithm on a compression application using natural images.

The design of non-separable two-dimensional wavelets has received much attention in the literature (see e.g. [18]). The algorithm presented in this paper could be adapted to this case without major changes.

The most interesting possibility for future research, however, is the use of non-linear lifting predictors. Evolutionary neural networks, for example, could be used instead of the FIR fiters used in this paper, which might lead to wavelets with more power to express end exploit regularities in the input data.

7 Conclusions

In this paper, we described a coevolutionary genetic algorithm that evolves biorthogonal wavelets encoded as a series of lifting steps. Applied to the task of compressing cubic spline curves, the algorithm consistently found near-optimal wavelets that outperformed some well-known wavelet bases.

The experiments reported in this paper show that the combination of lifting and genetic algorithms provides a powerful framework for adaptive wavelet design. Evolution of sequences of lifting steps seems to be a friendly problem for genetic algorithms, especially for the coevolutionary approach used.

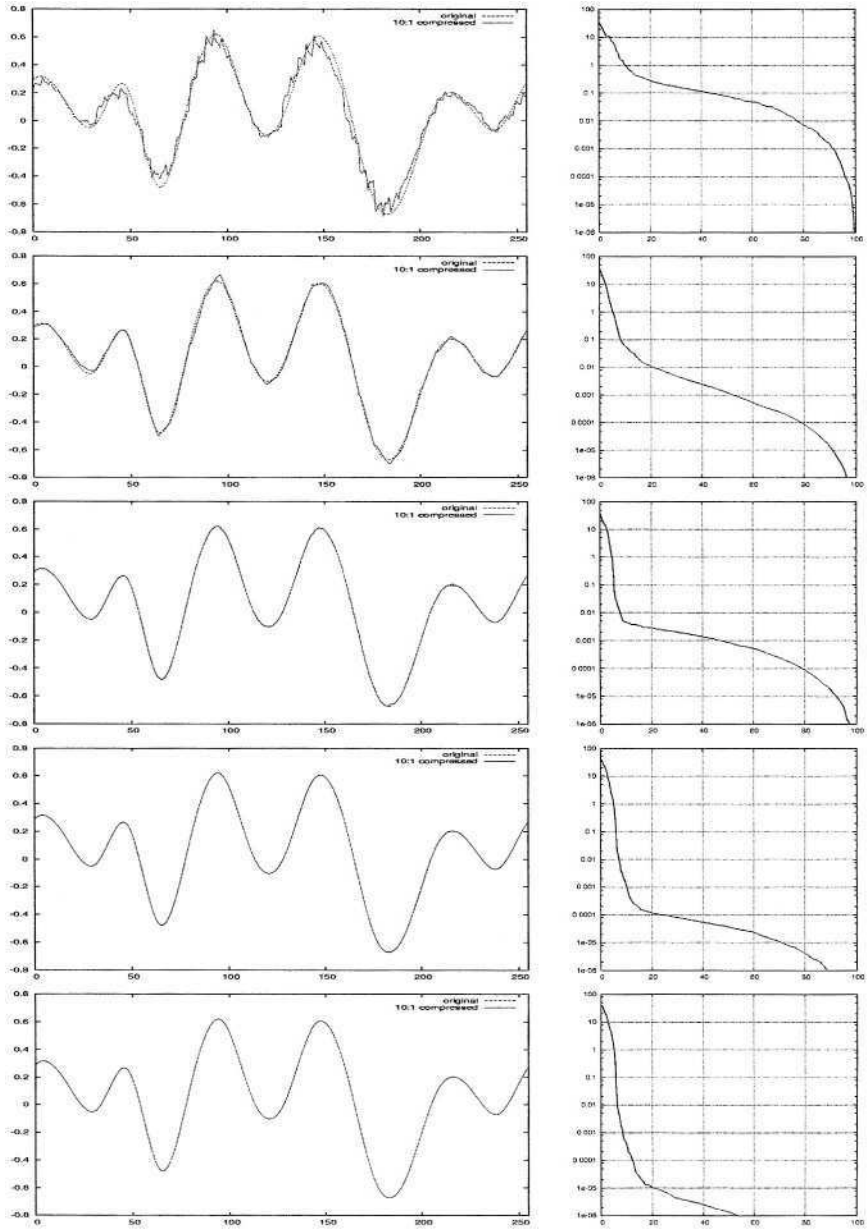


Fig. 8. The compression performance of the best wavelets after 1, 5, 10, 20, and 40 generations, from top to bottom. The plots on the left show the original data and the reconstructed data after 10:1 compression. The plots on the right show the distortion as a function of the percentage of coefficients used. By generation 40, the error has dropped by well over 3 orders of magnitude.

References

1. Gomez, F., Miikkulainen, R.: Solving non-markovian control tasks with neuroevolution. In: Proceedings of the International Joint Conference on Artificial Intelligence, San Francisco, CA (1999) 1356–1361
2. Jawerth, B., Sweldens, W.: An overview of wavelet based multiresolution analyses. SIAM Rev. **36** (1994) 377–412
3. Daubechies, I., Sweldens, W.: Factoring wavelet transforms into lifting steps. Journal of Fourier Analysis and Applications **4** (1998) 245–267
4. Davis, G., Nosratinia, A.: Wavelet-based image coding: An overview. Applied and Computational Control, Signals and Circuits **1** (1998)
5. Sweldens, W.: The lifting scheme: A custom-design construction of biorthogonal wavelets. Journal of Applied and Computational Harmonic Analysis **3** (1996) 186–200
6. Coifman, R., Wickerhauser, V.: Entropy-based algorithms for best basis selection. IEEE Transactions on Information Theory **38** (1992) 713–718
7. Wickerhauser, M.: Adapted Wavelet Analysis from Theory to Software. A. K. Peters, Wellesley, MA (1994)
8. Lankhorst, M.M., van der Laan, M.D.: Wavelet-based signal approximation with genetic algorithms. In: Evolutionary Programming. (1995) 237–255
9. Liu, C., Wechsler, H.: Face recognition using evolutionary pursuit. In: Proceedings of the Fifth European Conference on Computer Vision, Freiburg, Germany (1998)
10. Claypoole, R., Braniuk, R., Nowak, R.: Adaptive wavelet transforms via lifting. Transactions of the International Conference on Acoustics, Speech and Signal Processing (1998) 1513–1516
11. Erba, M., R.Rossi, Liberali, V., Tettamanzi, A.: Digital filter design through simulated evolution. In: Proceedings of the ECCTD 01, Espoo, Finland (2001)
12. Lee, A., Ahmadi, M., Jullien, G., Miller, W., Lashkari, R.: Design of 1-d fir filters with genetic algorithms. In: ISSPA 5th International Symposium. (1999) 955–958
13. Monro, D., Sherlock, B.: Space-frequency balance in biorthogonal wavelets. Transactions of the IEEE Int. Conf. on Image Processing **1** (1997) 624–627
14. Hill, Y., O'Keefe, S., Thiel, D.: An investigation of wavelet design using genetic algorithms. In: Microelectronic Engeneering Research Conference. (2001)
15. Villasenor, J., Belzer, B., Lia, J.: Wavelet filter evaluation for image compression. IEEE Transactions on Image Processing **2** (1995) 1053–1060
16. Daubechies, I.: Orthonormal bases of compactly supported wavelets. Comm. Pure Appl. Math. (1988) 909–996
17. Antonini, M., Barlaud, M., Mathieu, P., Daubechies, I.: Image coding using wavelet transform. IEEE Transactions on Image Processing (1992)
18. Sweldens, W.: The lifting scheme: A construction of second-generation wavelets. SIAM J. Math. Anal. **29** (1997) 511–546

Optimization of Constructive Solid Geometry Via a Tree-Based Multi-objective Genetic Algorithm

Karim Hamza[1] and Kazuhiro Saitou[2*]

[1] Ph.D. Candidate, [2] Associate Professor, Mechanical Engineering Department
University of Michigan, Ann Arbor, MI 48109-2102, USA
{khamza, kazu}@.umich.edu

Abstract. This paper presents the multi-objective evolutionary optimization of three-dimensional geometry represented via constructive solid geometry (CSG), a binary tree of boolean operations of solid primitives. NSGA-II is extended for binary tree chromosomes with customized crossover and mutation operators tailored for the evolution of CSG trees and applied for two-objective shape optimization of indoor modular space truss joints. The results show success in generating a variety of shapes over the Pareto front. A selection of Pareto-optimal shapes are manufactured using a solid freeform fabrication process.

1 Introduction

Shape optimization of three-dimensional solid bodies is a challenging task that finds many applications across a multitude of disciplines such as machine design, structures, micro electro-mechanical systems, fluid mechanics and aerospace. Automated shape optimization of three-dimensional geometry often requires the computer representation of the solid geometry [1,2], such as i) voxel representation, ii) octree representation, iii) constructive solid geometry (CSG) and iv) boundary representation (B-rep). In voxel representations [1,2], a solid is represented as a collection of small volumetric cells in uniform sizes. The octree representation [1] is a structured variant of voxel representation based on a hierarchical subdivision, which allows the shape representation with locally varying details.

By far, the family of shape optimization methods, which appear in the literature most often, is known as topology optimization [3,4]. Topology optimization relies mostly on voxel representation and sometimes on the octree. A possible reason for the popularity could be the ease of encoding and interpreting the design variables in terms of material/no material decisions. Many algorithms have been applied for topology optimization, including gradient based, evolutionary, stochastic and hybrid techniques. However, the fact remains that the final output of topology optimization

* Corresponding Author

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 981–992, 2004.

is a rough grid of material/no material decisions. Such output must thus be *realized* into a shape that can be manufactured; otherwise the result will not be useful.

Constructive solid geometry (CSG) [1,2] is a rather unique representation of solid body geometry, in the sense that its building blocks are solid primitives such as a cube, cylinder or a sphere. Shapes of increasing complexity are *constructed* from the primitives via merging or subtracting the primitives or other CSG solids. As such, most solid bodies represented by CSG are relatively easy to manufacture because all the surface features have simple geometry. CSG representation has been used for direct optimization of shape [5] as well as a post processor for realizing the complex geometry resulted from topology optimization [6].

Boundary representation (B-rep) [1,2,7] represent the solid geometry as the intersection of half spaces represented by the surfaces of the solid. B-rep is used as the internal kernel for most commercial computer aided design (CAD) software as it offers the best flexibility and speed for interactive user editing of the shape. However, the boundary information storage system is often too complex to allow for shape optimization. To the best of the authors' knowledge, there have been no successful shape optimization studies via direct manipulation of B-rep.

Although the CSG representation has several advantages in terms of feature simplicity compared to topology optimization, it seems to be less used for shape optimization in the literature due to the difficulties associated with its tree structure. This paper presents the multi-objective optimization of CSG trees using an extended NSGA-II [8,9]. The following sections present the mathematical formulation of the problem, the details of the extended NSGA-II, and a case study on the shape optimization of indoor modular space truss joints. The paper concludes with a summary and a further work.

2 Problem Formulation

2.1 Notations

A *solid body* is a closed subset S of the three dimensional space (Fig. 1).

A *solid primitive* is a solid body S_p uniquely defined by a few parameters. The solid primitives considered in this paper are: i) box, ii) cylinder and iii) sphere (Fig. 1). These solid primitives can be uniquely defined by defined an origin point, axes orientation and one to three sizing dimensions.

A *Boolean operation* is a mapping from two solid bodies to a solid body (Fig. 2). The boolean operations considered in this paper are: i) *union* (Fig. 2-a) represented as '+' and ii) *subtraction* (Fig. 2-b) represented as '-'.

Constructive solid geometry (CSG) tree is a binary tree T with solid primitives at terminal nodes and Boolean operations at non-terminal nodes. While a CSG T can uniquely represent a solid body $S(T)$, the inverse is not true: for example, a solid body in Fig. 2-c can be represented by either of the two CGS trees in Fig. 3.

n_p denotes the number of primitive solids (leaflet nodes) in the CSG tree.

n_B denotes the number of boolean operations solids (inner nodes) in the CSG tree. It should be noted that $n_B = n_P - 1$.

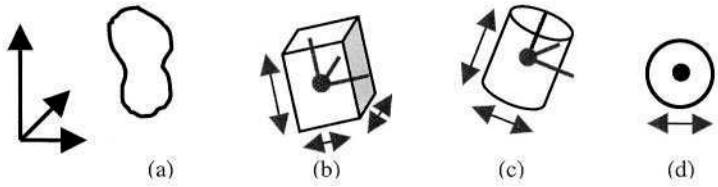


Fig. 1. (a) a general solid body is a closed subset of the three dimensional space, (b) a box can be uniquely defined by their origin location, orientation and three sizing dimensions, (c) a cylinder can be uniquely by its origin location, orientation and two sizing dimensions, (d) a sphere can be uniquely defined by its origin location and one sizing dimension.

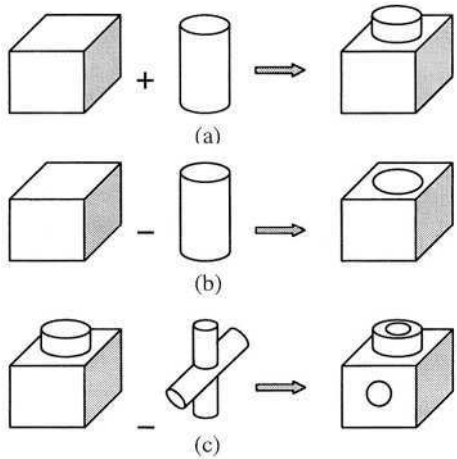


Fig. 2. (a) a box and a cylinder are combined via a union operation to form a padded box. (b) a box and a cylinder are combined via a subtraction operation to form a box with a hole. (c) Two solid bodies combine to form a more complex solid

2.2 Decision Variables

The decision variables are the information necessary to uniquely define a CSG tree: 1) number of primitive solids (n_p), 2) topology, 3) types of boolean operations at non-terminal nodes, and 4) types, origins, orientations and dimensions of solid primitives at terminal nodes.

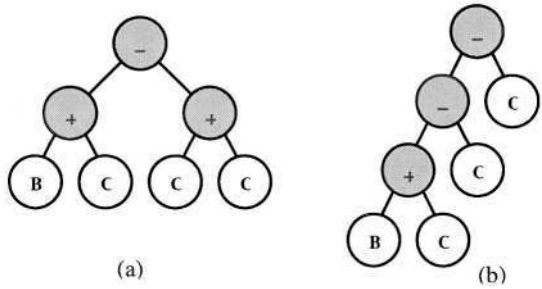


Fig. 3. Two CSG trees that can produce the solid body in Fig. 2-c. (a) the union of two cylinders are subtracted from the union of a box and a cylinder. (b) two cylinders are vertically and horizontally subtracted from the union of a box and a cylinder.

2.3 Objective Functions

Depending on the discipline that the shape optimization is intended for, there could be one or more objective functions. In the general case, the vector of objective functions can be written as:

$$\mathbf{f} = \langle f_1 \quad f_2 \quad \ldots \quad f_r \rangle \tag{1}$$

where $f_i = PerformanceMeasure_i(S(T))$, $i = 1$ to r

2.4 Constraints

Most applications require the final shape to be a single solid body. In other words, the solid represented by the CSG must be a *manifold* solid:

$$manifold(S(T)) = \text{true} \tag{2}$$

A manifold solid is a *connected set* in the three dimensional space. That is, any point within the solid body can be reached from any other point in the solid body, through some path that only traverses points within the solid body. Although making an exact check of the manifold condition for a general solid is a difficult task, there are algorithms that can quickly perform an approximate check with acceptable accuracy [7].

In terms of shape complexity, given the option for unlimited number of nodes in the CSG tree (combined with the sphere alone as a primitive), it is possible to get a near-zero representation of any solid shape in the three dimensional space. The proof to that is rather trivial, considering the limiting case of a sphere of a very small radius, the sphere approaches becoming a point. Given the option to do the union of unlimited number of points, it is possible to build-up any solid shape.

Practicality issues however, both in terms of computational resources as well as the manufacturability desire to decrease the geometric complexity (which is one of the

main drivers to use CSG, rather than other solid body representations), dictate that a cap should be placed on the maximum number of primitives to be considered in the CSG tree.

$$n_P \leq n_{P,\max} \tag{3}$$

The choice of $n_{P,\max}$ is up to the user and is essentially problem dependent. A large allowance on $n_{P,\max}$ would allow more flexibility in exploring more complex shapes. However excessively $n_{P,\max}$ is expensive in terms of computational resources and may have a negative effect of excessively increasing the size of the search space and thereby making the optimization task more difficult.

Aside from the manifold solid and maximum number of primitives, depending on the discipline of the problem, generic constraints can be incorporated into the problem. Generic constraints can have the functional form of:

$$feasible\,(S(T)) = \text{true} \tag{4}$$

3 Optimization Algorithm

3.1 Overall Flow

The overall flow of the optimization algorithm is similar to NSGA-II [8, 9] and is given as:

```
1.  Generate Initial Population
2.  Loop Until Termination: (Max. Number of Generations)
3.      Perform Pareto-Ranking of Current Population
4.      Preserve Elitism by passing copies of some top-rank members to
    new population
5.      Loop until New Population is full
6.          Select Parents for reproduction via binary tournament
7.          Perform Crossover and mutation then place the offspring in
            New Population
8.          When New Population is full, replace Current Population with
            the new one and repeat at Step #2.
```

Selection is performed via a binary tournament, in which two members are randomly selected from the current population. If one of the selected members has a better Pareto rank [9] than the other, the better member becomes a parent. If both members have the same rank, then one of them is randomly chosen to become a parent.

Elitism preservation is performed by passing copies of the top-ranked members to the new population prior to filling out the rest with the new offspring. If the number of top-ranked members in the current population is less than a threshold, their copies

are passed to the next generation. If there are more top rank members in the current population than a threshold, then only some of them are passed to the new population according to a nitching function that encourages the diversity of the Pareto front.

3.3 Chromosome Encoding and Evaluation

Encoding of the decision variables that define the CSG tree of a solid body in the linear format that is commonly used in multi-objective genetic algorithms seems inefficient. This is because the CSG is a variable-sized tree, which means that the length of the active portions of a linear chromosome are not guaranteed to be the same in two selected parents, which voids the use of many of the standard crossover techniques [10] for linearly encoded chromosomes. Furthermore, to adopt tree-structure crossover from a linearly encoded chromosome would mean continuous translation and de-translation between the chromosome and its equivalent tree, which is an unnecessary computational effort.

In this paper, the information of the decision variables is directly encoded as tree structure chromosome, which represents the CSG tree. This encoding scheme also has the advantage of allowing the chromosome is allowed to branch out and grow during the evolution of the CSG tree without a strict cap on the number of nodes or the topology of the tree.

Implementation of the CSG tree structure is performed via the C++ programming language, and is linked with the ACIS [7] solid modeling kernel, which facilitates translating the CSG tree into a format recognized by several computer aided design (CAD) software packages across several disciplines. The CAD packages could then report back the performance measures (problem specific objectives and constraints) of the solid shape.

3.4 Handling of Constraints

Keeping in mind that the evaluation of problem specific objectives and constraints using CAD packages might involve expensive computations such as finite element, dynamic and fluid mechanics simulations, the *constraints first – objectives later* approach [11] is adopted. Constraints are hierarchically listed staring with the manifold solid constraint (equation 2) then the cap on number of primitives (equation 3). Violating designs are given a *death penalty.* That is, any design that is infeasible in the first constraint will be dominated by any design that is feasible on that constraint, and so forth for the list of constraints. This approach of handling constraints removes the need to perform the expensive link with the CAD software for obviously bad designs as well as protect against CAD software instability, which can often happen if presented with a non-manifold solid or a solid of extreme complexity (reflected by too big a CSG tree).

3.5 Seeding of the Initial Population

Seeding of the initial population is an option, which the algorithm user might elect to activate. It allows the input of particular design guesses or previously known good designs into the initial population. When combined with elitism, seeding guarantees that the final outcome of the search will be no worse than the best of the previously known seeds. However, seeding might sometimes be inappropriate (especially for difficult feasibility problems), because it can cause excessive bias of the initial population and thereby lead to premature convergence of the search without much exploration beyond the given seeds.

3.6 Crossover and Mutation

Crossover and mutation operators are specially designed and are based on physical understanding of the problem represented by evolving the shape of a solid body.

- *Exchange of sub-trees:* This is a well established crossover technique in genetic programming [12, 13] for tree structures that present computer programs. Applying it to CSG trees corresponds to two parent solids exchanging geometric features. However, when the exchange is too randomly disruptive if it occurs too close to the root node of the tree, since it roughly corresponds to replacing half the solid with some other geometry that may not necessarily fit the first half. Therefore this form of crossover is used with medium overall probability.
- *Randomly Flipping a Boolean Operation:* This is a heavily disruptive mutation, which is used with low probability to induce diversity in the population.
- *Complete Randomization of a Primitive Solid:* This is another disruptive mutation, which is used with low probability to induce diversity in the population.
- *Slight Randomization of Origin/Orientation of a Primitive Solid:* This is used as a means of local adjustment/sampling of the positioning of primitive solids.
- *Short Local Search along Origin/Orientation of all Primitive Solids in Tree:* The short runs of local search (not pursued until reaching a local optimum) serve as a means of improving quality of the population members. However, local search should not be excessively used because employing it within the GA loop can lead to premature convergence as presenting extra computational cost.

4 Application Case Study: Indoor Modular Space Truss Joints

4.1 Problem Description

A schematic diagram of modular space trusses is shown in Fig. 4. A main advantage of space trusses is there efficient and light weight designs, yet a disadvantage is the extra complexity during manufacturing and erection. Producing space trusses in a modular form allows for mass production, which significantly reduces the manufacturing and erection cost. Large scale outdoor trusses generally have little restrictions

on the size considerations of the joints, which allows for sophisticated joints made of welded-plates, which are strong and light weight to be used. The indoor version of the modular space trusses are designed for quick erection/removal. Their main use is for hanging banners and other light weight objects. For the indoor version of the modular trusses, size and manufacturability considerations often dictate the shape of the modular joints to be simply spherical; a shape which can be challenged.

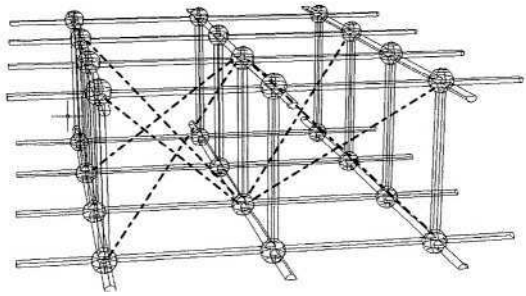


Fig. 4. A schematic diagram of a modular space truss (cross diagonal structural members removed from the diagram to allow better visibility)

The first objective is a measure of the strength of the joint. In this preliminary study, a fast rough calculation is sufficient. This is done by observing the material cross-section around load paths and using it as a measure of stiffness. The objective is to maximize the stiffness, but since the implemented algorithm is set for minimization, the first objective is given as:

$$\text{Min. } f_1 = -\sum_{1}^{\text{nLoadPathPoints}} \min(\mathit{MaterialDepth}(S(T))) \tag{5}$$

The second objective is minimization of weight or amount of material in the joint, which is given as:

$$\text{Min. } f_2 = \mathit{volume}\,(S(T)) \tag{6}$$

Aside from the manifold solid constraint and the cap on geometric complexity (maximum 20 primitive solids for this problem), the triple symmetry of the joint allows for reduction of the search space by limiting the search to one octant. The evolved geometry in the octant is then triple mirrored to produce a tri-symmetric joint. However to allow for this reduction, the evolved solid shape must be well connected to the three mirroring planes, or else the mirroring could produce a non-manifold solid. This is represented by the constraints:

$$\mathit{IntersectionArea}\,(S(T)) \geq 2\% \text{ of unit square, for XY, YZ and ZX planes} \tag{7}$$

It is important to note that the choice of objectives and constraints is highly application dependent. Although weight minimization is a common objective in many structural design problems, performance measures for strength are usually more difficult. In this particular study, knowledge about the function of the joint allowed for prior definition of the load paths. In general cases however, it would be necessary to link evolving geometry with automated meshing for finite element analysis (FEA) using CAD packages. For this study, evaluation of objectives typically takes less than two seconds on a 2.0GHz PC. Incorporating FEA would probably increase the computations ten-fold, thereby increasing typical time to perform one GA run from approximately one hour (as in this study) to about half a day, which is still quite feasible and could be pursued in future research.

4.2 Results and Discussion

Several runs of the implemented algorithm are performed for the truss joint. The main tuning parameters for the GA runs are listed in Table 1. A typical run produced the quasi-Pareto curve of Fig. 5. The main interest of this case study is observing how the shape of the joint changes along the trade-off between stiffness and weight. Therefore, the actual quantities of f_1 & f_2 are non-dimensionalized with respect to the highest and lowest found points. The CSG trees of the points in the Pareto-curve are exporting as Solid models via the ACIS library [7] and the rendered CAD models are displayed as small icons in Fig. 5. Furthermore, selections of the CAD models are input to a rapid freeform manufacturing machine. Photos of the actual manufactured models are displayed in Fig. 6.

Table 1. Tuning parameters of the GA runs for the indoor modular space truss joints case study

Population Size	100
Number of Generations	30
Threshold for number of elite members	20
Initial Population Seeding	i) 6 Seeds (solid cube, sphere, sphere with forking cylinders at 2 sizes), ii) Unseeded
Prob. Exchange of sub-trees	0.50
Prob. Flip Boolean Operation	0.20 / Number of nodes
Prob. Complete Randomize Primitive Solid	0.20 / Number of nodes
Prob. Local Positioning Shift	0.50 / Number of nodes
Prob. Short Local Search	i) 0.00, ii) 0.05

It is noted that the search space is large (even for the simple considered problem). A twenty-primitive solid tree would have 19 binary choices for the inner nodes (boolean operations), 20 discrete choices and approximately 20×6 continuous variables for the terminal nodes (primitive solids). It was observed in most runs however, that most of the higher fitness designs had few primitive solids providing material cover to the important zones in the truss joint. This observation permitted the use of relatively

small population size, while achieving consistent performance in most of the performed runs.

Other than the sample run used for manufacturing of the models, a total of twenty runs were performed, each five of which using different parameter settings in terms of population seeding and probability of performing local search as stated in Table 1. The top rank population members of each of those runs are collectively plotted in Fig. 7. It is observed in Fig. 7 that none of the runs with unseeded initial populations was able to reach the highest 10% joint stiffness region of Pareto-front. On the other hand, most of the seeded runs had several members in that region. This is understandably the effect of seeding, since one of the seeds is just a solid cube (filling all the search shape search space with material), which is obviously the Pareto-point that has the highest possible stiffness, yet heavy weight. In the low to mid level joint stiffness region however, the unseeded runs seem to be working better. This might indicate that seeding can be causing excessive bias to the higher stiffness region of the Pareto-front.

Although implementation of local search seems to make intuitive sense since a good proportion of the design variables are continuous, the effect of local search is hardly observable in the high and low joint stiffness regions. In fact, in the mid level region, it seems to have a negative effect. This could probably be attributed to premature convergence effects.

Overall, most of the runs performed well, which indicates consistency of success of the algorithm for the considered case study.

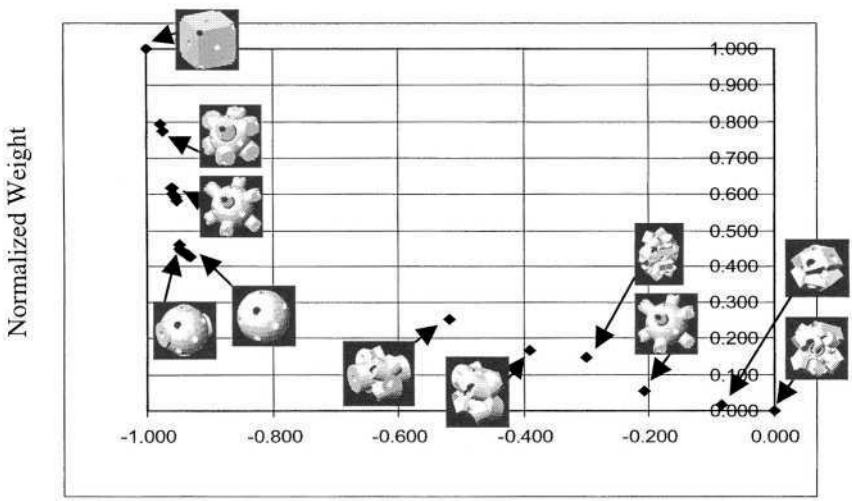


Fig. 5. Results of a typical run of the implemented algorithm showing the normalized trade-off between maximizing stiffness and minimizing joint weight. Also shown are CAD models of the first rank evolved CSG trees

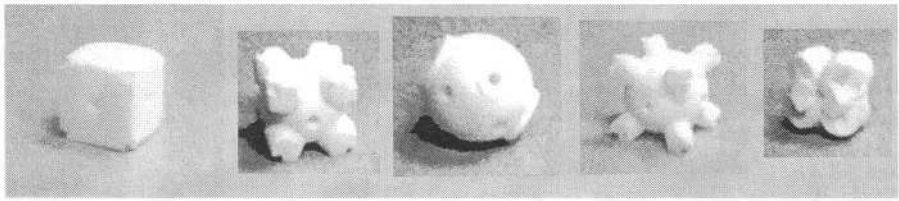


Fig. 6. Photos of selected quasi-Pareto optimal shapes that were rapidly manufactured using selective laser sintering

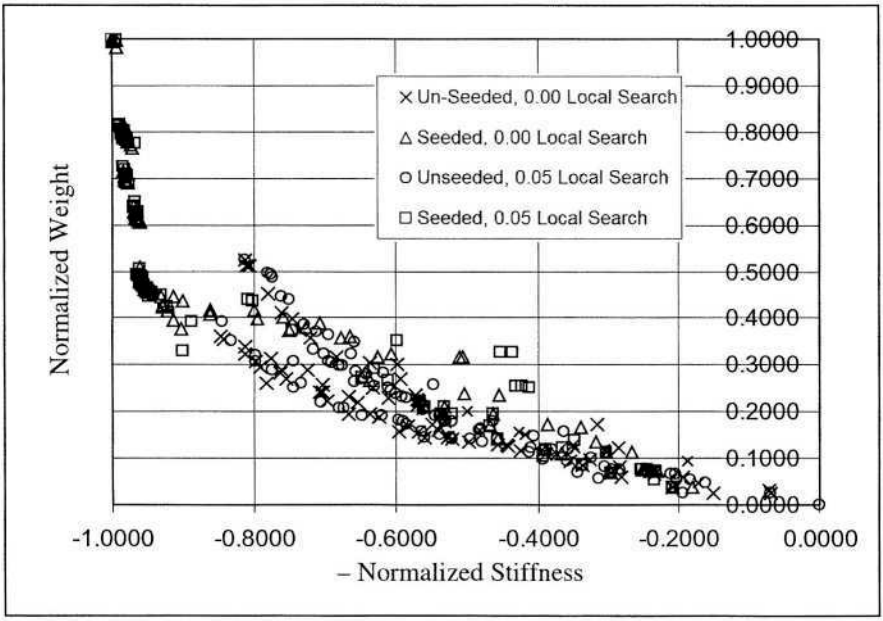


Fig. 7. Summary of top rank population members for twenty GA runs

5 Conclusion and Further Work

This paper presented an adaptation of NSGA-II for shape optimization of three dimensional solids by evolving CSG trees. The algorithm is set to directly evolve the tree structure by applying specially tailored crossover and mutation operators to a tree structure encoded chromosome. A simple case study involving shape optimization of indoor modular space truss joints is presented to demonstrate the algorithm. Models of selected quasi-Pareto optimal joint shapes were then manufactured via a rapid solid freeform fabrication technique.

Further work would include case studies of more complex components, possibly across several engineering disciplines. Also, more experimentation with the tuning parameters of the algorithm to determine the effects on the search efficiency is to be pursued.

Acknowledgements. The authors would like to extend special thanks to Prof. Suman Das from the University of Michigan, Ann Arbor. The manufacturing of the models of the modular truss joints was performed in the Solid Freeform Fabrication Lab as part of a course project on Solid Freeform Fabrication taught at the University of Michigan during the Fall semester of 2003.

References

1. Maentylea, M.: An Introduction to Solid Modeling. Computer Science Press, Inc., Rockville, Maryland (1988)
2. Lee, K.: Principles of CAD/CAM/CAE Systems. Addison Wesley, Reading, Massachusetts (1999)
3. Chapman, C., Saitou, K., and Jakiela, M., "Genetic Algorithms as an Approach to Configuration and Topology Design," Transactions of ASME, Journal of Mechanical Design, v. 116(1994)1005–1012
4. Bendsøe, M. and Kikuchi, N. Generating optimal topologies in structural design using a homogenization method, Computer Methods in Applied Mechanics and Engineering, 71 (1998) 197-224
5. Kodiyalam, S., Kumar, V., Finnigan, P.: Constructive solid geometry approach to three-dimensional structural shape optimization. AIAA, 30 (1992) 1408–1415
6. Chirehdast, M., Papalambros, P.: Conversion of spatial-enumeration scheme into constructive solid geometry. Computer Aided Design, 26 (1994) 302–314
7. Corney, J, Lim, T.: 3D Modeling with ACIS. Saxe-Coburg Publications, Stirling, UK (2001)
8. Deb, K., Argawal, S., Pratab, A., Meyarivan, T.: A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. Proceedings of the Parallel Problem Solving from Nature VI Conference, Paris, France (2000) 849-858
9. Coello, C., Van Veldhuizen, D., Lamont, G.: Evolutionary Algorithms for Solving Multi-Objective Problems. Kluwer Academic Publishers (2002)
10. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. 3rd edn. Springer-Verlag, Berlin Heidelberg New York (1996)
11. Kurpati, A., Azarm, S., Wu, J.: Constraint handling improvements for multiobjective genetic algorithms. Struct Multidisc Optim 23 (2002) 204–213
12. Koza, J.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press (1992)
13. Michalewicz, Z.: How to Solve it: Modern Heuristics. Springer-Verlag, Berlin Heidelberg New York (1999)

Co-evolutionary Agent Self-Organization for City Traffic Congestion Modeling

Luis Miramontes Hercog

Eguiara y Eguren #128. Viaducto Piedad.
México, D.F., 08200 México
Tel.:+5255-55199861, Fax:+5255-55199861
lmhercog@yahoo.co.uk

Abstract. This paper introduces a model based on a multi-agent system that learns using the extended classifier system (MAXCS). The agents are informed in the news, after the weather forecast, the congestion levels of the main city avenues and they decide which avenue to use the following day. The results are encouraging, as the agents, both homogeneous and heterogeneous adapt to the congestion thresholds set by the authorities. The main factors for this adaptation are the reward received by the agents and their perception.

Keywords: Road traffic self-organization, Learning Classifier Systems, XCS, Emergent behavior, heterogeneous agents.

1 Introduction

The government of Mexico City has recently (5th December, 2003) introduced a traffic system that measures the congestion in the main avenues of the city. This measure is done counting the number of cars per minute that cross certain key points in the avenues. Then, the congestion for each of the avenues is reported in the evening news following the weather forecast. It has been shown how a number of agents can adapt to several comfort thresholds in modes of transport if they perceive the same information, in this case the congestion report in the news [10].

This paper introduces a system where heterogeneous agents, that learn from the traffic reports, can co-evolve and produce traffic self-organization using inductive reasoning [1]. Inductive reasoning is ideal to solve ill-defined problems like the one analyzed here. There are avenue comfort thresholds set by the city government, though some agents ignore them. Each agent has a level of traffic tolerance, which is assumed to be +-0.1 of the threshold set by the authorities, in case they ignore the figure suggested by the government. The agents translate the capacity reports into rules, which they evolve using an extended classifier system (XCS) [18,4]. Several experiments with different thresholds and different number of agents show that this evolutionary computation approach for traffic self-organization is feasible and could be used both, by the people – to decide which avenue to take for their journey – and by the city authorities to assess

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 993–1004, 2004.

which city routes need expansion or an alternative path (such as another metro line, bicycle lanes, or which public transport mode to improve). The experimental results also show that when all the agents pay attention to the suggestion made by the authorities (homogeneous agents) or they are less tolerant to what the authorities suggest (heterogeneous less-tolerant agents), the comfort threshold in the avenues is achieved. As most of the agents do not pay attention to what the authorities suggest, the avenue usage depends on the reward received, rather than the tolerance of the agents. Furthermore, the multi-agent simulation allows individual behavior analysis. These results are supported by 20 sets of experiments. Section 2 presents the model on which this system is based. Section 3 gives a brief overview of the learning algorithm (XCS). Section 4 introduces MAXCS a multi-agent system that learns using XCS. Section 5 explains the experimental settings followed by the results in Section 6.

2 Traffic Simulation Model and Representation

Each of the main avenues of Mexico City, which are the arteries of the main routes through the city, have recently been provided with car counters and cameras to assess the congestion levels. The government has set a novel strategy of providing this data to the citizens after the weather news every evening. The government makes suggestions of how the congestion can impact the travel time based on the number of cars per minute that pass by the car counters. This model proposes and shows that instead of telling the people what to do, it is better to give them the congestion information and let them learn from it.

Therefore, the model used for this simulation tries to be as realistic as possible and takes only the information that the users receive from the traffic report every evening to plan their journey the following day.

It is assumed that the users modeled are traveling in the same routes in the city and that the number of avenues they can use is limited, therefore they have to make a decision on which avenues to use. Sometimes the users can decide to take an alternative route through the city, hence not congesting the avenues. All three possible choices are parallel, therefore no intersections need to be modeled.

The agents learn to model their environment by encoding it into rules. Each rule has a condition and action part representing if-then rules. The condition part represents several previous days of congestion information for each of the avenues. Each agent has comfort preferences, which the agents encode as a binary set: 1 represents that the avenue was not congested, 0 that it was. Every evening the agents compare the number of cars per minute in the avenue to their comfort threshold and encode the bit. For example, a bit set as 10 would tell the agent that it was good to use avenue 1 and not avenue 2. Then, each agent has bit sets for every day information. To keep some homogeneity in the system, the agents have the same memory (5 days). The action part tells the agent which avenue to use if any, i.e. 0 alternative route(not using the avenues that are monitored), 1 take avenue 1, 2 take avenue 2, etc. This model is based on previous findings [9].

$$R = \frac{1000}{e^{\left((congestion - CT)^2 \cdot 1000\right)^2}} \quad (1)$$

The agents learn through a reinforcement learning mechanism (XCS) and a reward is given based on equation 1 using the algorithm in Fig. 1, where *CT* is the avenue comfort threshold. For every step, all agents encode the levels of congestion of their preferred route and using XCS – explained in the next section – they decide which action to take, either one of the avenues, or a side street. As seen in Fig. 1, if the agents decide to take an alternative route they are rewarded if and only if all the avenues are congested – as it would take longer to reach their destination – and in that case they receive half of the maximum reward. Hence there are AV+1 actions for the agents.

In Fig. 1 the variables are: *MAXAV,* constant that indicates the maximum number of avenues. *MAXAGS,* constant that indicates the maximum number of agents. *MAXACTIONS,* constant value MAXAV+1. *maxReward,* constant with value 1000. *alt_route,* Boolean variable that indicates if action 0 was the best. *pay_Reward,* function that calculates the reward value using Eq. 1. *usage,* array indexed by avenue number, *avenue_result,* array indexed by avenue number, showing whether it was good (1) or not (0) to use an avenue. *rewards,* array of size MAXACTIONS+1, indexed by action number and which in rewards[0] keeps the reward for action 0. All the arrays' cells are initialized to zero.

3 Extended Classifier System (XCS)

Learning classifier systems (LCS) [15] are a machine learning system, situated in an environment that presents a problem that the system tries to learn and solve. The learning process takes place in discontinuous time intervals, where the system is presented a state and it gives an answer to it; depending on the answer there is a reward. If the reward is given just after the action is evaluated, the problem is considered a single step problem; if the reward is given after several actions are taken, it is called a multiple-step problem. If the sensors provide enough information for the system to differentiate the different states of the problem, i.e. each state is different to the system's sensors, the problem is called Markovian, if not it is called non-Markovian [17]. A problem is also called stationary if for the same state, the correct action is the same [17].

LCSs learn by evolving simple strings encoded as if-then rules using a genetic algorithm (GA) [13] and a reinforcement learning algorithm (RLA) [17] to determine the usefulness of the rules. The condition composed generally by 0, 1, # (where # is a wild card) and the action a bit set. The RLA is used to update the rules' fitness that can be, for example, the amount of food or the profit that the rule generates when is activated by the state that the system perceives from the environment. XCS [18,4] is a LCS that uses 3 parameters to measure a classifier's usefulness: the prediction (p) of the reward, the error (ϵ) of the prediction and the fitness *(F).* The fitness of the rule is an inverse function of the error. XCS uses a Temporal Difference (TD) algorithm to update the values, i.e. learning classifier systems are evolutionary reinforcement learning methods.

```
void Calculate_and_Give_Rewards()
{
  for agent=1 to MAXAGS
   usage[agent.action]=usage[agent.action]+1
  next agent

  for avenue=1 to MAXAVS
  if(usage[avenue]/MAXAGS)<=thresholds[avenue] then
       avenues_result[avenue]=1
    else
       avenues_result[avenue]=0
    end if
  next avenue

  alt_route=true
  for avenue=1 to MAXAVS
    if avenues_result[avenue]=1 then
     alt_route=false
  next avenue

  for action=0 to MAXACTIONS
   if action=0 then
     if alt_route=true then
        rewards[action]=maxReward/MAXAVS
     else
        reward[action]=0
     end if
   else
     rewards[action]= avenue_result[action]*
                      getReward(usage[action],
   end if             thresholds[action])
  next action

  for agent=1 to MAXAGS
    update agent values using rewards[agent.action]
  next agent
}
```

Fig. 1. The pseudo code algorithm for the reward calculation and agent update. This algorithm is run every time step after all the agents take their action [11].

The interactions between XCS and its environment in a single step problem are cognitive cycles: perception of the environment, activation of the rules that match the current environmental state (forming the match set [M]), action selection (forming the action set [A]), environmental evaluation of the action, reward and reinforcement of the rules that fired the action. The action is selected from all the possible actions in [M], randomly if exploring or deterministically

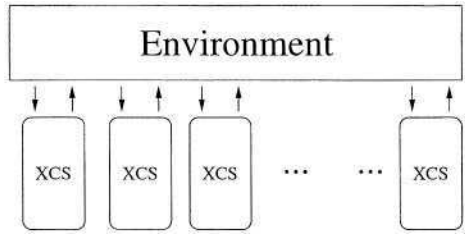


Fig. 2. MAXCS: each agent is a single, independent XCS.

if exploiting. When exploiting, the prediction value ($\frac{\Sigma F_j p_j}{\Sigma F_j}$, where j are the classifiers with the same action) is calculated for all the possible actions in [M], then the action with the highest value is taken. The GA is used to create new rules from fit rules. The GA is invoked when the average of the time stamp of the rules in [A] is greater than the GA threshold (θ_{GA}) [18,4]. The rules generated by the GA are inserted back in the population, and in order to keep the size of the population constant, the rules are deleted based on the estimated size of [A], the fitness and the experience of the rule (for all the deletion techniques and more details about XCS the reader is referred to [18,16,4]).

4 MAXCS: A Multi-agent System That Learns Using XCS

MAXCS [12,9] is a multi-agent system that learns using a learning classifier system, each agent is represented by an XCS (Fig. 2).

The accuracy based fitness, the RLA used by XCS to learn, and its genetic algorithm let XCS evolve accurate and compact rule sets [16]. These rules as in other LCSs, can be read as if-then rules. This property has been very useful for the analysis of the behavior of the agents.

MAXCS exploits the possibility of representing individuals directly, (both their behavior and their interactions) and provides the detailed analysis required to represent agents correctly [7]. A full "dissection" can be made analyzing the population of each agent, as well as their individual answers, providing a clearer result that is easier to understand.

The activation of each set of rules can reveal what is happening individually in the system, unveiling some features of the system that otherwise would remain unexplained.

This multi-agent system satisfies the requirements of multi-agent simulation, based on the idea of a computerized representation of entities' behavior in the world. This computerized representation gives the possibility of representing a phenomenon, which emerges from the interactions of a set of agents with their own operational autonomy [7].

Holland [14] proposed the use of learning classifier systems to represent the agents for an artificial stock market [2], which started the research on agent-based computational economics (ACE), e.g. [14,2].

The rationale behind applying learning classifier systems to social simulation, in this paper, is that they are self-explanatory: by looking at their rule population and the update of the estimation values, a step by step analysis can be made. By having a self-explanatory system, for social simulation, the behavior of the agents can be analyzed in detail and properly explained. From an individual analysis viewpoint this feature can be very valuable. For the work presented here, this can be interpreted as a snapshot of a person's mind at the exact moment of taking the decision of which avenue to take, and the reasons for that decision.

Davidsson [5] has shown how multi-agent based simulation, and other micro simulation techniques, explicitly attempts to model specific behaviors of specific individuals. He compared them favorably to macro simulation techniques that are typically based on mathematical models, where the characteristics of a population are averaged together and the model attempts to simulate changes in these averaged characteristics for the whole population.

Davidsson's [5] findings – combined with Ferrari's [8] – have proved that a mathematical simulation model has to be discarded and restated from the beginning when new variables are added to the system – bring another opportunity niche for the micro-LCS-based simulation presented here.

On the other hand, economists (e.g. [1]) and game theorists (e.g. [3]) have shown that some human decisions are not very rational. Based on this assumption of the lack of rationality in human decisions, MAXCS is used to try to achieve the suggested usage of the avenues in the city, based on inductive reasoning agents [9]. Furthermore, the strategies used by the agents are not set in advance, but they are evolved and chosen autonomously by each agent. Hence, the simulation would provide a framework for assessing the different avenues and their congestion levels and could be used for planning.

5 The Experiment

After the successful use of 10 homogeneous agents and 2 avenues (not shown), a more difficult setting has been selected. First, 101 homogeneous agents with 2 avenues, after this setting yielded self-organization, heterogeneous agents were tested. The heterogeneity is computed randomly by assigning 3 types of agents (type 1, 2 and 3), which take the thresholds told by the authorities, and randomly add or subtract 0.01 to them. Then, during the run, the modulus by 5 operation is computed using the agent number to assign the agent type: 1 for type 1, 2 for type 2, 3 for type 3, leaving 0 and 4 with the thresholds told by the authorities. Therefore, only 40% of the agents pay attention to what the authorities suggest for comfort threshold.

Each agent is represented by homogeneous XCS with conditions of size 5 and 3 possible actions with the parameter values: # probability= 0.333, explore/exploit rate(P_x)= 0.5, crossover rate (χ)= 0.8, mutation rate (μ)= 0.02,

θ_{GA} = 25, minimum error (ϵ_0)= 0.01, subsumption experience 50 and learning rate (β)= 0.2, subsumption deletion (a type of rule "condensation") is used in the GA and the action set, these parameter values have yielded good results before [11]. The populations of the agents are initialized using supersaturation (each 25% of the population is generated with a different random seed) [6]. The covering mechanism is used only when [M] is empty (θ_{mna}) and all three possible actions are generated, taking as initial values p=20.00, ϵ=0.0 and F=0.1. Exploration in these experiments is done by selecting randomly an action with P_x=0.5 from [M] (first 2500 steps). P_x=1.0 for full exploitation trials(last 2500 steps). The former to allow the agents to learn, the latter to test what they have learned. Avenue 1 has a threshold of 30 cars/min and avenue 2 of 50 cars/min., set to 0.3 and 0.5 respectively for the experiments.

Agent heterogeneity is given by how the agents perceive their environment and react to the congestion of the avenues. After the adaptation to the 0.01 random tolerance change 0.1 was tested too. The use of a percentage rather than the exact number of cars per minute allows the system to be flexible and increase the number of agents as it is needed. As the number of agents used in these experiments is 101, the threshold is almost the same as the number of cars per minute tolerated by the agents.

6 Results

The results are presented differently depending on the focus that is given. First, the 20 runs, with different random seeds, are averaged to test if the agents are learning. Figures 3 and 4 show the average of the 20 runs: Fig. 3 shows all 101 homogeneous agents, Fig. 4 shows a congestion coefficient random variation of 0.01 for each agent's perception, which is similar in overall behavior to a variation of 0.1 (only shown as individual behavior in Fig. 6). Then, to unveil the reason for the difference between the runs, an individual run is analyzed. As the individual behavior would get lost with an averaged agent avenue usage.

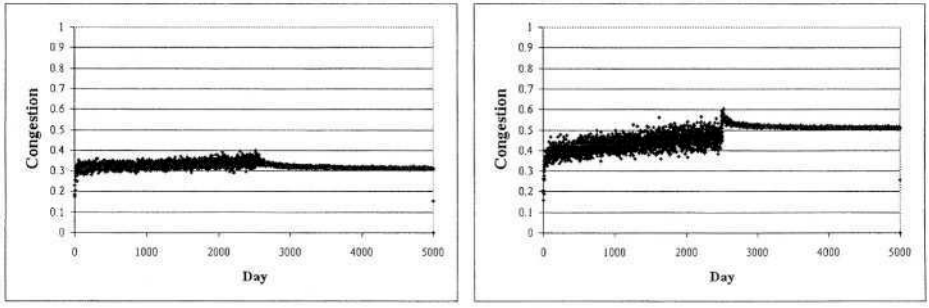


Fig. 3. Average of 20 runs using 101 homogeneous agents with CT_1=0.3 (left) and CT_2=0.5 (right).

A pattern can be perceived in Figs. 3 and 4: the agents adapt to the threshold set very early in the experiment, it takes the agents less than 30 days to achieve the congestion levels set, especially avenue 2 in Fig. 3 is under-congested until day 1500. Once the exploration phase is switched off and the agents use the rules they have learned (day 2500), a closer value to that set by the authorities is achieved in all of the cases.

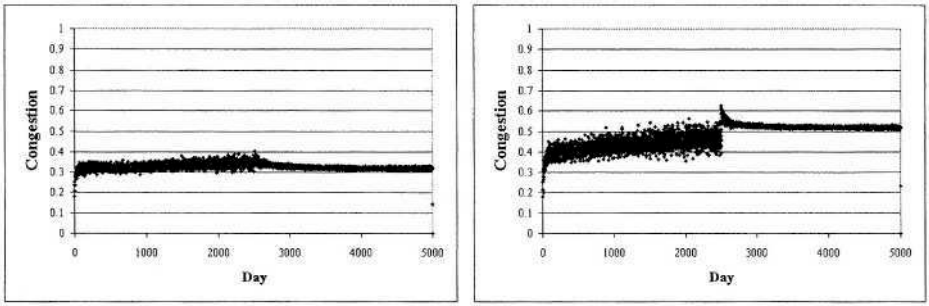


Fig. 4. Average of 20 runs using 101 heterogeneous agents with CT_1=0.3 (left) and CT_2=0.5 (right); 40% of the agents pay attention to the authorities and the rest have +- 0.01 in their thresholds randomly. A +-0.1 change yielded a similar plot, but a different individual behavior.

The experiments with homogeneous agents (Fig. 3) and those with a random 0.01 change (Fig. 4) are more similar than both of the heterogeneous agents. A gap of 0.01 between the congestion wanted and the value achieved by MAXCS. Homogeneous agents yield a value closer to that set by the authorities than heterogeneous agents.

Agents changing their actions all the time produce the variation above the congestion threshold. This phenomenon is produced by the nature of the problem: there is not enough capacity for all the agents to use the avenues, therefore some of them will have to take the alternative routes using little streets in the city. Even though all of them (for the sake of the model) would prefer to use the main avenues, as they would move faster.

The agent individual behavior is analyzed in figures 5 and 7. Each of the squares represents an agent in the system. The squares are shaded depending on the use of the avenues every 100 days: [0,25] white square, [26,50] gray square, [51,75] dark gray square and [76,100] black square. Agent 1 is the top left agent, agent 10 is at the end of the first row and agent 101 is the square at the bottom left. Hence, those agents using an avenue all the time are shaded in black, and those not using it in white, those using it not so often in gray. For example, in Fig. 5 it can be seen that agents 10, 35 and 95 are using always the side streets, while those agents in gray are changing between both avenues and the side streets.

These individual behavior figures are plotted using a single run, as averaging would blur the results, due to the random avenue usage. Random meant as that for different runs, different agents will use different avenues, not random behavior of the agents as such.

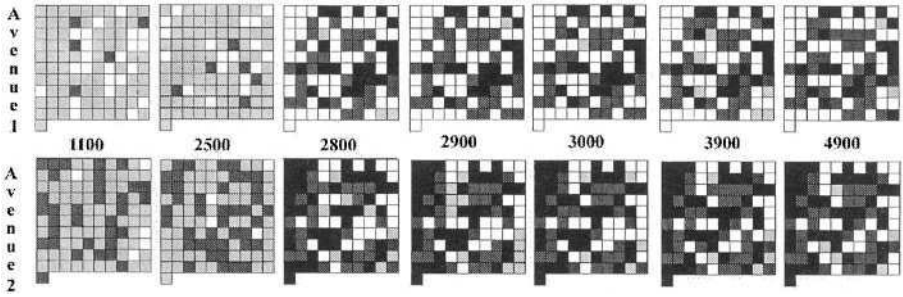


Fig. 5. Individual avenue usage for 101 homogeneous agents.

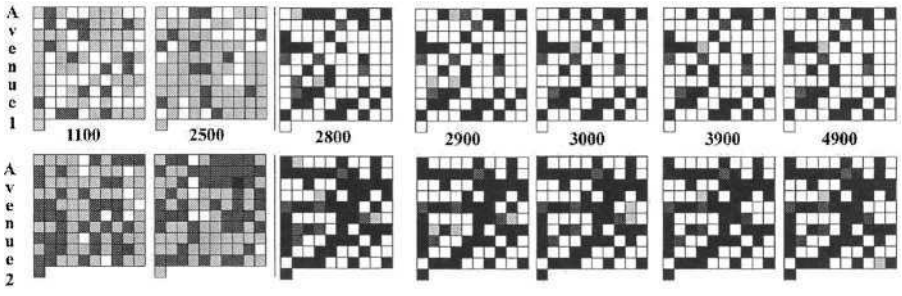


Fig. 6. Individual avenue usage for 101 heterogeneous agents with a 0.1 random change.

The three individual behavior graphics are taken at different steps, mainly the first one (1100) at a time where the agents have adapted already, are still exploring, hence they have not fixed to use a specific avenue. The fixation comes after step 2500, when the GA is not working anymore and the agents are exploiting their knowledge. It can clearly be seen a group of agents which is in light gray shade. These agents are balancing the usage of the avenues and the side streets.[1] The fact that some agents fix to a given avenue, either 1 or 2, leaves with no choice the other agents, but to keep trying the side streets and the avenues. When these vacillating agents take the side streets, they allow the others

[1] These agents are called vacillating agents, and they produce the Nash equilibrium of the problem [11].

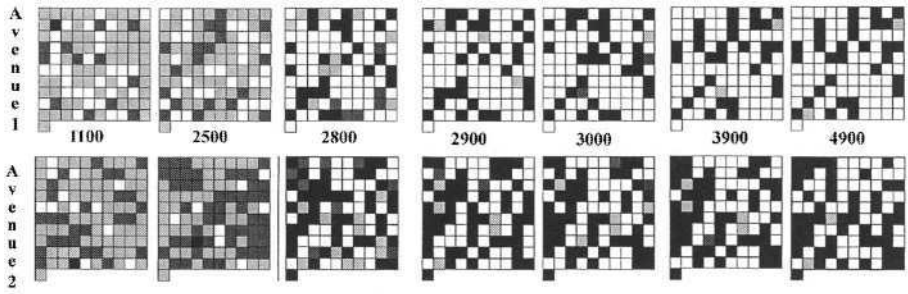


Fig. 7. Individual avenue usage for 101 heterogeneous agents with a 0.01 random change.

to use the avenues without congestion, while when they take the avenues they over-congest them, producing comfort in the side streets. Hence, they balance the system.

From the different experimental settings shown in Figs. 3 and 4, it can be inferred, from the agents behavior, that the main factor for adaptation is the reward, rather than the perception of the agents. Therefore, this simulation can be used by the authorities to set the policies (i.e. rewards) to achieve the congestion levels wanted in the city avenues. It can be seen that heterogeneity is a crucial factor in the agents' behavior (Fig. 7), as they fix quicker to either avenue, leaving 20% of the agents to change between avenues and side streets. Analysis of the evolved rules show very general rules, (e.g. # # # # # # # # # # :2) for some agents, but very specific (e.g. 100##11#01:0) for others. This specificity is produced by the initial preference of the agents for the side streets or a given avenue. Even though one would think that the latter example would be more advanced than the former, it is the opposite. The agents that fix to use certain avenue, as the first rule shows: whatever happens take avenue 2, is the best strategy. This is due to the nature of the problem, because by doing this, the agent will profit from the vacillation of those that are changing between the side streets (action 0) and the different avenues (actions 1 and 2).[2] As Littman [17] has pointed out, when agents in a multi-agent problem achieve a Nash equilibrium, it is considered an optimum performance. Hence, the solution that emerged from MAXCS's co-evolution is the optimum performance. This is the reason given by the author to claim that the city authorities and the people in general could use the model presented here to forecast any changes in the avenue congestion or to forecast the avenue usage.

[2] Because a Nash equilibrium is the best possible action, given the other agents' action and no agent can profit better, but by changing strategy [3]. The interested reader can find the game theoretical discussion of the problem in [9].

7 Conclusions

This paper has shown how a homogeneous and heterogeneous multi-agent system can learn from the traffic reports co-evolving to produce traffic self-organization using inductive reasoning. Mainly this co-evolution being an effect of the reward the agents receive, rather than their perception of the traffic information.

Whether they decide to use it or not, the traffic information is relevant for some of the agents, but it is not a crucial factor. The solution has been found in some agents that use consistently either avenue 1 or 2, and leave no option for the others to change from the side streets to the avenues, balancing the avenue congestion.

Another factor that is crucial for the agents' behavior was the level of tolerance set individually, as the agents have learned mainly based on the reward they receive, rather than in their individual perception.

Several experiments with different thresholds and different number of agents show that this evolutionary computation approach for traffic self-organization is feasible and could be used both, by the people – to decide which avenue to take for their journey – and by the city authorities to assess which city routes need expansion or an alternative path (such as another metro line, bicycle lanes, or which public transport mode to improve).

The learning classifier system approach has been successful in achieving optimal performance (in the form of a Nash equilibrium) and in evolving readable rules, which indicate the preference of the agents, hence an individual behavior pattern can be analyzed.

The results reported are encouraging and more research is planned in this direction.

References

1. W.B. Arthur. Complexity in Economic Theory: Inductive Reasoning and Bounded Rationality. *The American Economic Review,* 84(2):406–411, May 1994. http://www.santafe.edu/~wba/Papers/El_Farol.html.
2. W.B. Arthur, J.H. Holland, B. LeBaron, R. Palmer, and P. Taylor. Asset Pricing Under Endogenous Expectations in an Artificial Stock Market. In W.B. Arthur, S. Durlauf, and D. Lane, editors, *The Economy as an Evolving Complex System, II,* Reading, MA., 1997. Addison-Wesley. Proceedings Volume XXVII.
3. K. Binmore. *Fun and games: a text on game theory.* D.C. Heath, 1992.
4. M.V. Butz and S.W. Wilson. An algorithmic description of XCS. In P. L. Lanzi, W. Stolzmann, and S. W. Wilson, editors, *Advances in Learning Classifier Systems. Third International Workshop (IWLCS-2000),* pages 253–272, Berlin, 2001. Springer-Verlag.
5. P. Davidsson. Multi-Agent Based Simulation: Beyond Social Simulation. In S. Moss and P. Davidsson, editors, *Second International Workshop, MABS 2000,* volume 1979 of *LNAI,* pages 97–102. Springer-Verlag, 2000.
6. L. Davis, C. Fu, and S. W. Wilson. An Incremental Multiplexer Problem and its Uses in Classifier System Research. In W. Stolzman, editor, *Fourth International Workshop on Learning Classifier Systems - IWLCS-2001,* pages 23–32. Springer-Verlag, 2002.

7. J. Ferber. *Multi-Agent Systems: An introduction to distributed artificial intelligence*. Adisson Wesley, 1999.
8. P. Ferrari. A model of urban transport management. *Transportation Research Part B,* (33):43–61, 1999.
9. L. Miramontes Hercog. *Evolutionaty and Conventional Reinforcement Learning in Multi-agent Systems for Social Simulation.* PhD thesis, South Bank University, 2003.
10. L. Miramontes Hercog. *Learning Classifier Systems Applications,* chapter Traffic Optimization using Multi-agent Systems and Learning Classifier Systems. LNAI. Springer-Verlag, 2004. To appear.
11. L. Miramontes Hercog. Multi-agent inductive reasoning using co-evolutionary classifier systems: the Multibar Problem, 2004. Submition to Evolutionary Computation Journal.
12. L. Miramontes Hercog and T.C. Fogarty. Social simulation using a Multi-Agent Model based on Classifier Systems: The Emergence of Vacillating Behaviour in the "El Farol" Bar Problem. In P.L. Lanzi, W. Stolzman, and S. W. Wilson, editors, *Advances in Learning Classifier Systems. Proceedings of the Fourth International Workshop in Learning Classifier Systems 2001.* Springer-Verlag, 2002. LNAI series 2321. ISBN: 3-540-43793-2.
13. J.H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.
14. J.H. Holland and J. Miller. Artificially Adaptive Agents in economic theory. *American economic review,* 81(2):265–370, 1991.
15. J.H. Holland and J.S. Reitman. Cognitive Systems Based on Adaptive Algorithms. In D. A. Waterman and F. Hayes-Roth, editors, *Pattern-directed inference systems.* New York: Academic Press, 1978.
16. T. Kovacs. XCS Classifier System Reliably Evolves Accurate, Complete, and Minimal Representations for Boolean Functions. In Roy, Chawdhry, and Pant, editors, *Soft Computing in Engineering Design and Manufacturing,* pages 59–68. Springer-Verlag, London, 1997.
17. M. L. Littman. Value-function reinforcement learning in markov games. *Journal of Cognitive Systems Research,* (2):55–66, 2001.
18. S.W. Wilson. Classifier Fitness Based on Accuracy. *Evolutionary Computation,* 3(2):149–175, 1995.

Validating a Model of Colon Colouration Using an Evolution Strategy with Adaptive Approximations

Džena Hidović and Jonathan E. Rowe

School of Computer Science, The University of Birmingham, Birmingham B15 2TT, UK
{D.Hidovic, J.E.Rowe}@cs.bham.ac.uk

Abstract. The colour of colon tissue, which depends on the tissue structure, its optical properties, and the quantities of the pigments present in it, can be predicted by a physics-based model of colouration. The model, created by analysing light interaction with the tissue, is aimed at correlating the histology of the colon and its colours. This could be of a great diagnostic value, as the development of tissue abnormalities and malignancies is characterised by the rearrangement of underlying histology. Once developed, the model has to be validated for correctness. The validation has been implemented as an optimisation problem, and evolutionary techniques have been applied to solve it. An adaptive approximate optimisation method has been developed and applied in order to speed up the computationally expensive optimisation process. This works by iteratively improving a surrogate model based on an approximate physical theory of light propagation (Kubelka Munk). Good fittings, obtained under the histologically plausible values of model parameters, are presented. The performances of the new method were compared to that of a simple Evolution Strategy which uses an accurate, but expensive, Monte Carlo method. The new method is general and can be applied with any surrogate model for optimisation.

1 Introduction

The colour of tissue is a result of light interacting with the tissue while propagating through it. It therefore directly depends on tissue architecture, structure and composition. Many pathological changes are characterised by rearrangements of tissue structure and composition, which implies that also the colour of normal and abnormal tissue differ from each other. However, the changes in colour may often be very subtle and invisible to the human eye. To overcome this problem, a novel approach to interpreting the medical images of tissue has been developed and successfully implemented on the skin [1,2]. Our current work concentrates on extending the application area of that approach to the colon.

The basic idea of the research lies in understanding the physics of colour image formation: that is, the interaction of incident light with the tissue. White light interacts with the tissue structure and composition and penetrates it to a certain depth. Some of the light is absorbed and scattered forward by particles and macromolecules inside the tissue, and some remitted at its surface. The remitted portion of light gives tissue its colour. The fractions of the light that are remitted, scattered and absorbed depend on the optical properties of the tissue itself, which in turn directly depend on its structure and

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1005–1016, 2004.

the pigments present in it. By simulating the processes of light interaction with tissue for all plausible normal variations of its histological parameters, it is possible to predict all its possible colours. This means that a tissue which has a normal architecture will have colour that can be predicted by the simulations. Any deviation from these predicted colours is a sign of abnormality, since differences in the colour between normal and abnormal tissue are due to the rearrangements of its architecture and composition, as argued by many clinicians and physicians.

The key concept in interpreting the images of tissue in terms of its histology is a *model of colouration* which is a set of correspondences between the parameters characterising the tissue and its colours. The model is constructed by taking the optical properties of tissue and calculating the corresponding spectral reflectances by mathematical models of light propagation. Spectra are then convolved with the curves of the response system and colours, typically RGB, are obtained. The model can then be used to perform the inversion process, i.e. to infer the combination of histological parameters which lead to a particular colour. The model of colon colouration which predicts the light reflected back from the colon tissue, and its parameters, will be be briefly described in the next section.

Once developed, the model has to be validated for correctness. Validation, described in more detail in section 3, is normally done by comparing the light reflected from the tissue (spectral reflectance) predicted by our theoretical models, and the measured spectral reflectance from the clinical experiments. This problem is implemented as an optimisation problem with the goal of minimising the distance between the measured and predicted spectra. To solve the optimisation problem, evolutionary algorithms have been used. The high cost of calculating the spectral reflectance accurately resulted in the development of an adaptive approximate optimisation method. The basic idea lies in creating a surrogate model, and iteratively correcting the error between the accurate and approximate models which solve the original problem. More detailed description of the method and the motivation for its developement are given in section 4. As shown in section 5, the new method gives satisfactory results in significantly less time than that required using only the accurate but computationaly expensive model of light transport in tissue. A discussion of the results and some limitations of the method will conclude the paper.

2 Background: Model of Colon Tissue Colouration

Colon tissue is a layered structure (figure 1) composed of four layers, characterised by different optical properties. Starting from the innermost layer, sequentially they are: mucosa, submucosa, muscularis externa propria and serosa.

White light penetrating into the colon tissue is mainly scattered by a network of collagen fibres whose different sizes in colon layers, imply different scattering properties. Most of the light remitted at the tissue surface is the result of backward light scattering which occurs in the mucosal layer. Besides being scattered, the incident light gets absorbed on its way inside the colon tissue. The absorption is mainly due to oxy and deoxy hemoglobin which form the main part of the red blood cells. Light transmitted through the muscle layer forms just a small fraction of the incident light and

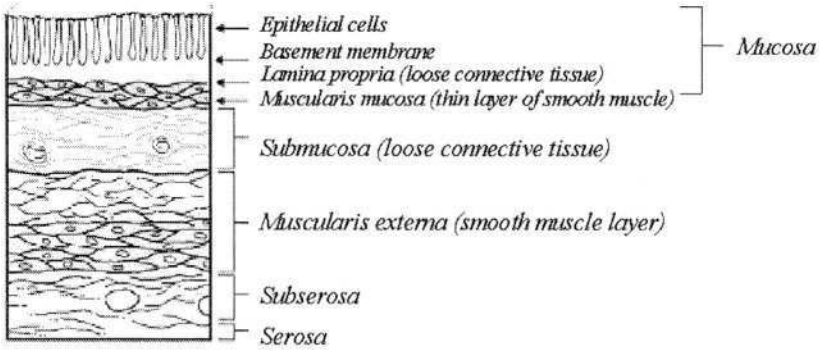


Fig. 1. Colon architecture

is strongly forward directed. Hence, it does not influence the light remitted from the colon. Therefore, our model simulates the light interaction with only the first three layers.

For calculating the spectra of the colon we use the Monte Carlo method [3,4], which is considered to give an accurate solution to the radiative transfer equation, describing light transport in tissue. Given the information about the absorption and scattering coefficients of each of the layers composing the model, their respective thicknesses, refractive indexes and anisotropy factors, the Monte Carlo method calculates tissue spectral reflectance, by statistically analysing the interaction of photons with the tissue. Each photon is traced independently on its way through the tissue, until it is either completely absorbed by the pigments of tissue; transmitted into deeper layers; or it escapes at the surface. In order to get accurate analysis, a large number of photons, usually $10^5 - 10^6$, have to be simulated. That gives a huge computational complexity to the Monte Carlo method.

In order to apply the above method to calculating the light remitted from the colon, scattering and absorption coefficients, and thicknesses of the first three layers of the colon tissue have to be known. Given that the previous literature reports little on the quantitative optical properties of the colon, we have modelled scattering and absorption properties ourselves. Scattering coefficients were calculated using Mie theory [5] starting from collagen fibre size and density. Absorption coefficients on the other hand are expressed as a function of haemoglobin concentration and saturation of each layer.

The need to calculate the optical coefficients prior to calculating the spectral reflectance, resulted in the following parameters of our model:

- *haemoglobin concentration,* i.e. the amount of hemoglobin per unit volume of tissue. This parameter can be expressed as a product of hemoglobin concentration per unit volume of blood and the volume fraction of blood in a particular tissue.
- *size of collagen fibres,* i.e. diameter of collagen fibres.
- *density of collagen fibres,* i.e. number of collagen fibres per unit volume of tissue.
- *thickness* of tissue layer

Each of the above parameters was defined for both mucosa and submucosa resulting in eight model parameters. For each parameter a range of all values corresponding to

normal colon tissue was defined. All the ranges have been confirmed as histologically plausible by a pathologist with a special interest in colon tissue. Optical properties of muscularis externa were fixed and hence no additional parameters regarding that layer were considered. There were therefore a total of eight model parameters.

3 Model Validation

The main objective of the validation process was to verify the correctness of the model of colon colouration. This has been done by comparing spectra computed from the model by Monte Carlo simulations with spectra measured from the real tissue. If the model is correct, then it must be able to generate spectra which approximate well the measured spectral reflectances. However, this is only the first step, as the inverse is not necessarily true.

The simulated spectra were compared against the spectra obtained by diffuse reflectance spectroscopy *in vivo* [6], where a bundle of fibre optics which deliver and collect the incident and reflected light respectively is passed through a working channel of an endoscope and placed in contact with the colon wall of the patients during ordinary colonoscopy procedures. In particular, 84 spectra of normal colon, kindly provided by Kevin Schomacker, have been used in the validation. No spectra of abnormal colon were used, as the model developed so far simulates the light interaction with only normal tissue. The measured spectra was recorded every 2 nm in the range from 300 run to 800 nm. However, given the huge computational complexity of Monte Carlo simulations, which is proportional to the number of wavelengths at which the reflected spectrum is calculated, the measured and simulated spectra were compared at only the following set of 17 wavelengths (chosen to give a sufficiently accurate characterisation of the spectra): {450, 480, 506, 514, 522, 540, 548, 560, 564, 568, 574, 586, 594, 610, 640, 676}. Most wavelengths were selected in the green region of the visible light, because that part of the spectra is heavily changed by the light absorption of haemoglobin.

The validation was implemented as an optimisation problem, with the aim of minimising the distance between the simulated and measured spectra. The distance between the two curves was calculated using the following measure of distance between two sets of points:

$$d(y, z) = \frac{1}{m} \sum_{i=1}^{m} |y_i - z_i|$$

where y_i is a value of the spectral reflectance measured on real colon tissue *in vivo* at the wavelength w_i, z_i is a value of reflectance simulated using our model of colon colouration (starting from a particular set of parameter values) at wavelength w_i, and m is the total number of wavelengths (in our case $m = 17$).

In addition to the eight model parameters described in the previous section, a scale factor was introduced to account for the adjustments of the normalisation process, where the measured reflectance spectra were normalised to a spectrum remitted from a reflectance standard. This is a standard procedure in experimental spectroscopy to account

for the characteristics of the illumination used in the measurements. There were therefore nine parameters to be optimised.

Originally, a novel (1+1)-ES was used to solve to optimisation problem [7]. At each iteration, a set of parameters was chosen as described in [7], and the corresponding reflectance spectra was calculated. The high computational complexity of the Monte Carlo method (in our simulations 60000 photons were used), resulted in an optimisation procedure that was very slow. The execution time of each Monte Carlo simulation was typically several minutes. For (1+1)-ES to find a good approximation, typically a few hundred of iterations had to be executed. Consequently, we developed a method that makes use of a fast approximate model of light transport, as explained in the next section.

4 Approximate Models

The problem of model validation, from an optimisation point of view, may be summarised as follows. We have a set of parameters, each lying in some bounded interval, describing a physical model. For any particular choice of parameter values, we can calculate the resulting optical spectrum using an expensive Monte Carlo algorithm. Given a particular spectrum (measured during colonoscopy), we need to find the associated parameter values. The fitness function is the distance between the spectrum derived from the Monte Carlo algorithm and the target spectrum (as described above).

This situation, in which the fitness evaluation is computationally expensive, is quite common in real-world optimisation problems. A typical strategy for dealing with this situation is to use approximate models, or *surrogates,* that are faster to evaluate. These may either come from fitting a model to sampled data, or may have derived from a theoretical analysis of the problem. In our case, the Kubelka Munk theory [8,9], which is an approximate analytical solution to the radiative transfer equation, provides a fast approximate method. The Kubelka Munk method describes the two flux theory of light transport in a seminfinite slab of particular thickness. Propagation of light is limited to being only directly forward and backward oriented, which makes this method essentially a one dimensional approximation.

The derivation of surrogates and their management within the optimisation process has been considered by a number of authors (for example, see [10] for a useful survey). If the surrogate comes from fitting a model to sample data (e.g. by training a neural network [11], or fitting a smooth function over a mesh of points [12]), then it can be periodically improved by incorporating more data points as the search progresses. The idea is that one interleaves periods of search with updates to the surrogate (e.g. by retraining the neural network on the new data), in such a way that the surrogate becomes more accurate as the search converges to an optimum.

In our situation, we have a surrogate model derived from an approximate physical theory of light transport. It does not make sense to "re-train" this model on the basis of sampled data. However, we can build a model of the *error* between the Kubelka Munk method and Monte Carlo. One could, perhaps, train a neural network to model this error, but we propose a much simpler and faster method.

The overall management scheme is as follows:

1. Initially, set the error model to be zero.
2. Optimise using Kubelka Munk, plus the error model.
3. Run Monte Carlo on some points near the "optimum" found in step 2.
4. Use these points to update the error model.
5. Go to step 2.

This cycle is repeated as many times as necessary (or can be afforded). Note that each cycle requires the optimisation of an approximate model (at step 2). We use a novel (1+1)-ES to perform this optimisation, which it does efficiently and accurately. This Evolution Strategy is based on a new mutation probability distribution and is described in detail elsewhere [7]. In this paper, we concentrate on the method for updating the error model, which we describe in the next section. At each cycle, we produce an improved surrogate model, that estimates the error function within the vicinity of the optimum point found using the surrogate. When a new cycle begins, with a new surrogate, we start the next run of the (1+1)-ES from that point, ensuring that the search takes place that is both near the true optimum, and in a region for which the new surrogate is an accurate model.

5 The Error Correction Algorithm

Let us re-state the problem in more general terms. We have n parameters and a genetoype-phenotype map $M : \mathbb{R}^n \to \mathbb{R}^m$. There is a function $\phi : \mathbb{R}^m \to \mathbb{R}$ which evaluates the phenotype. The fitness function is given by $\phi \circ M$. The map M is computationally expensive. We also have an alternative map $K : \mathbb{R}^n \to \mathbb{R}^m$ which is an approximation to M. The map K is computationally efficient.

In our example, we have $n = 9$ parameters, and M is the Monte Carlo algorithm for generating the corresponding spectra evaluated at $m = 17$ wavelengths. ϕ is the distance function $\phi(s) = d(s, \tau)$, where s is the simulated spectrum, and τ is the target spectrum. K is the Kubelka Munk method.

Define the *error function* $E : \mathbb{R}^n \to \mathbb{R}^m$ to be $E(x) = M(x) - K(x)$. This is the function we will try to model more and more accurately at each cycle. Let us suppose that at cycle t the approximate error model is E_t. Initially, we have $E_0(x) = 0$ for all x. The surrogate that is used in cycle t is $S_t = K + E_t$. We suppose that running the optimisation algorithm with surrogate S_t has produced the parameter vector $z \in \mathbb{R}^n$. This means that $S_t(z) = K(z) + E_t(z)$ is very close to the target spectrum τ.

We seek to construct our next approximation to the error function, E_{t+1}. We will approximate the error function E at z by estimating its differential dE at z. The differential is the best linear approximation to the function at a given point. So we wish to find a matrix A_{t+1} which is an estimate of dE at z, and then set

$$E_{t+1}(x) = A_{t+1}(x - z) + E(z)$$

which we can see agrees exactly with E at the point z. Define a vector

$$b_{t+1} = E(z) - A_{t+1}(z)$$

then our new error model is

$$E_{t+1}(x) = A_{t+1}(x) + b_{t+1}$$

The problem is to find the matrix A_{t+1} and, from there, the vector b_{t+1}.

Notice that $E(z)$ is the difference between the accurate model M and the initial surrogate $S_0 = K$. We would like to incrementally improve our approximation in terms of the behaviour of the *current* surrogate S_t. We can do this as follows. Define a *surrogate error* function

$$F_t(x) = M(x) - S_t(x) = M(x) - K(x) - E_t(x) = E(x) - E_t(x)$$

This implies that $E(x) = F_t(x) + E_t(x)$. We now find an approximation to $F_t(x)$ of the form $B_t(x - z) + F_t(z)$, where B_t is an estimate of the differential of F_t at z. Then we set

$$\begin{aligned} E_{t+1}(x) &= B_t(x - z) + F_t(z) + E_t(x) \\ &= B_t(x) - B_t(z) + F_t(z) + A_t(x) + b_t \\ &= (A_t + B_t)(x) + b_t + F_t(z) - B_t(z) \end{aligned}$$

This equation gives us an algorithm for updating the surrogate model at each cycle.

1. Initially set $A_0 = 0$ and $b_0 = 0$.
2. At cycle t get the error, $F_t(z)$, between the accurate model M and the current surrogate S_t at the point z found by the optimisation procedure.
3. Estimate the differential of F_t at the point z by the matrix B_t.
4. Set $A_{t+1} = A_t + B_t$
5. Set $b_{t+1} = b_t + F_t(z) - B_t(z)$

The surrogate model at each cycle is $S_t(x) = K(x) + E_t(x) = K(x) + A_t(x) + b_t$.

It remains to show how to calculate B_t. Let e_k be the vector with zeros everywhere except at position k, where there is a one. Then $B_t(e_k)$ tells us the kth column of the matrix B_t. To find this, we simply make the approximation to F_t agree with F_t at a small perturbation $z + \delta e_k$ from z (where δ is small compared to the defining bounds of parameter k). That is, we require

$$F_t(z + \delta e_k) = B_t(z + \delta e_k - z) + F_t(z)$$

Therefore

$$\begin{aligned} B_t(e_k) &= \frac{1}{\delta}(F_t(z + \delta e_k) - F_t(z)) \\ &= \frac{1}{\delta}(M(z + \delta e_k) - S_t(z + \delta e_k) - F_t(z)) \end{aligned}$$

which we can calculate by running the accurate model M and the surrogate S_t on the perturbed vector $z + \delta e_k$. We repeat this for each parameter k to get the whole of the matrix B_t. We see that each cycle therefore requires $n + 1$ runs of the accurate model M.

6 Experimental Results

We tested the adaptive error correction algorithm on a sample of 20 spectra drawn randomly from our collection of normal colon tissue images. Since running the Monte Carlo algorithm takes several minutes, we allowed a maximum of 200 runs of this algorithm for each spectrum. Since there are 9 parameters, each cycle of the error correction algorithm requires 10 runs of Monte Carlo. We therefore ran 20 cycles of the algorithm. The optimisation of the surrogate at each cycle was done with our (1+1)-ES with novel mutation probability distribution [7]. This was allowed 2000 evaluations of the surrogate at each cycle. This was usually sufficient to optimise the surrogate within a small tolerance.

In comparison, we also tried to solve the optimisation problem by just using the (1+1)-ES with 200 function evalutions using Monte Carlo. Six of the results are illustrated in figure 2 which shows the original spectra, and the best achieved by the error correction algorithm and the (1+1)-ES (by itself). We also compared the two algorithms after 50, 100, 150 and 200 runs of Monte Carlo (that is, after 5, 10,15 and 20 cycles of the error correction algorithm). The results are presented in table 1. These show clearly that the new algorithm significantly outperforms the plain (1+1)-ES. It also appears to be more robust, in that the results show a smaller standard deviation for the new algorithm.

Table 1. Results of running the error correction algorithm versus (1+1)-ES on 20 spectra, after 50, 100, 150 and 200 iterations of the Monte Carlo algorithm. The average fitness (distance from target spectrum) is shown with standard deviation in brackets. Significance is measured by a paired t-test.

Iterations	Error Correction algorithm	(1+1)-ES	Significance
50	0.0176 (0.0102)	0.0269 (0.0219)	89.13%
100	0.0130 (0.0063)	0.0183 (0.0099)	97.74%
150	0.0111 (0.0055)	0.0161 (0.0080)	99.07%
200	0.0096 (0.0044)	0.0151 (0.0075)	99.93%

7 Discussion

The first observation to make about the experimental results, is that they confirm that the physical model of the colon tissue can account for the spectra observed in normal tissue. In addition to the 20 spectra from the sample data set used in this paper, further testing on the remaining 52 spectra confirm these findings. Good fittings obtained under the assumption of histological plausibility of the model parameters, which was confirmed by a pathologist with special interest in colon tissue, are a step forward in proving the correctness of the model. However, the validation process was done only on the normal spectra as the model developed so far predicts the light interaction with the normal colon

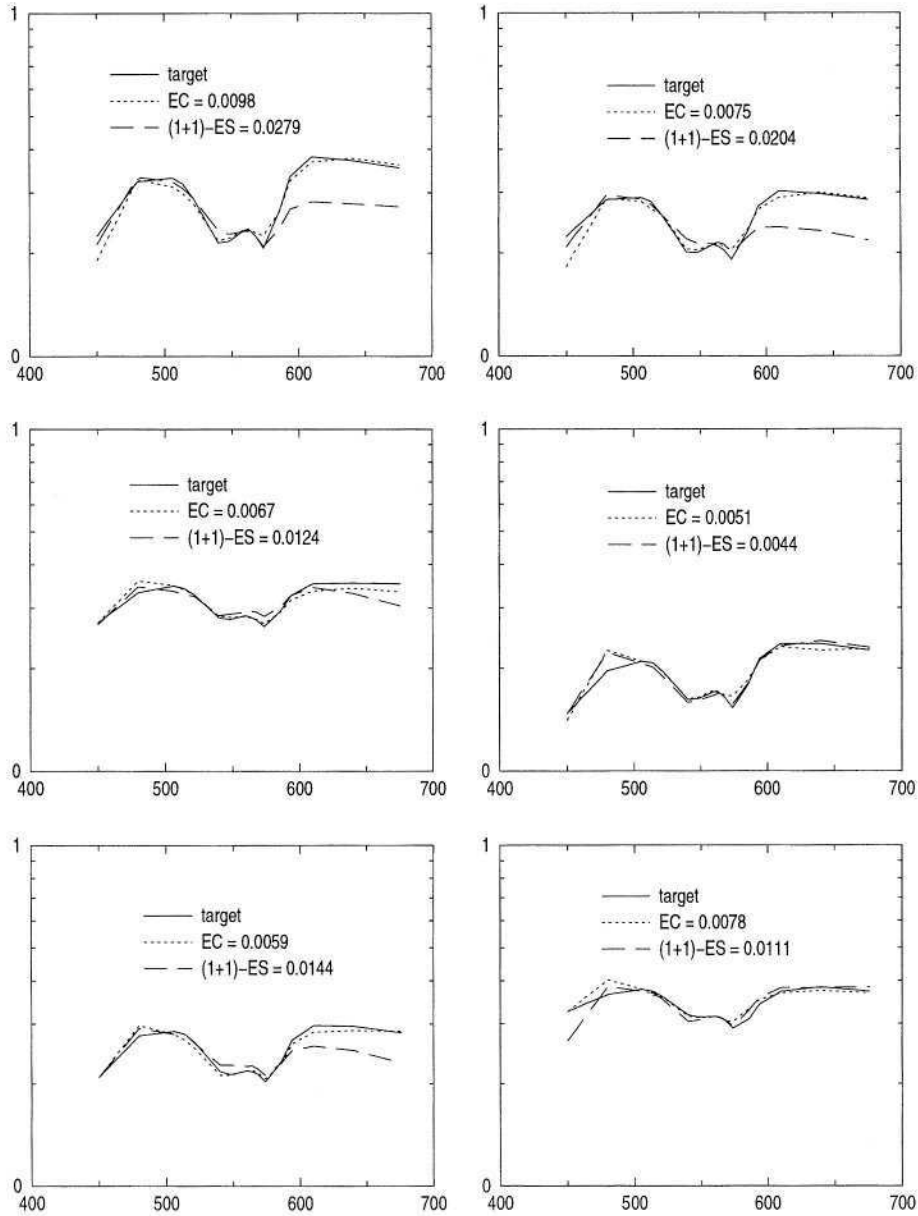


Fig. 2. Six examples of evolved spectra using the error-correcting algorithm (EC) and plain (1+1)-ES. Each algorithm uses the Monte Carlo algorithm 200 times for each spectrum. The final fitness is shown in each case, given by the distance to the target spectrum (solid line)

tissue. Further work is being done to analyse the effects of abnormal tissue structure (e.g. cancer). First findings suggest that due to the structural and compositional differences between the two types of tissue, the spectra of abnormal colon differs from that of the normal tissue. This is promising that our approach could provide diagnostically significant information. Detailed studies, however, have still to be done.

The second observation is that the error-correcting algorithm is generally more efficient at finding the optima that the plain (1+1)-ES, although this algorithm is itself very good (getting even close to the optimum in only 200 function evaluations is no mean feat!). The fact that the (1+1)-ES is a good algorithm directly helps the error-correcting algorithm. This is because we can be fairly sure that the *surrogate* optimisation problem can be solved efficiently at each cycle. Clearly, if we could not find the surrogate optimum, then we would not be looking in the right area to correct the surrogate.

This leads on to another observation about how the error-correcting algorithm is working. We need to be confident that the optima that are found for the surrogate are reasonably close to the optimum of the accurate model. At least, the closer it is, the fewer cycles will be necessary to find it. And obviously, if the surrogate is completely wrong, then it will be misleading rather than helpful to use it. In our case, the initial surrogate, the Kubelka Munk model, is actually not a great approximation. Apart from the physical assumptions it makes about the way light scatters in a tissue, it also calculates the total diffuse reflectance from the surface. What we actually need is the amount of reflectance within the given area of the collection probe, which is considerably smaller (especially at the red end of the spectrum). Nevertheless, after 20 cycles, the corrected surrogate is acting as a very good approximation in the vicinity of the optimum — which is also where the search is taking place.

A more technical issue is the question of how accurately we can model the error function $E(x) = M(x) - K(x)$ by an affine function. This is equivalent to asking how good an approximation is the differential of E to E itself at a given point. We know that (assuming E is differentiable) it is arbitrarily good within a sufficiently small radius of the point. So if the error function varies very slowly, then we should be able to get a good approximation. If it varies very rapidly (or indeed is not differentiable) then we could have a problem. But note that, if the surrogate has a optimum near the genuine optimum, we only require that the error function varies smoothly in that region of the search space.

A related issue is: even if the differential actually does give a good approximation to the error function, how good is our estimate of the differential? Recall that we estimate this by sampling in each parameter direction and then making our estimate agree with the error function at each of the sampled points. This is determined by the choice of the parameter δ. If this is too large (compared to the rate at which the error function is varying) then the estimate will be poor. On the other hand it can't be made arbitrarily small — we need to pick up the variation of $E(X)$ in the vicinity of z to a significant degree. In our experiments we scaled all our parameters to lie in the range [–1, +1] and took $\delta = 0.025$. Other values also work. It is possible that this parameter could be adapted in response to the size of the error found, but we have not investigated this.

8 Conclusions

In this paper, we have presented a method for validating a model of colon colouration. The model allows us to calculate the spectral reflectance of colon tissue for a range of histological parameters using Monte Carlo simulation. The validation process was formulated as an optimisation problem. Given that the Monte Carlo method is an expensive (yet accurate) method of light propagation in tissue, the optimisation procedure was very slow. In order to speed up the optimisation, we have developed a method using a surrogate approximate model, based on Kubelka Munk theory. Our method is adaptive: the error between the surrogate and the accurate model is estimated and refined at each cycle. This leads to a more efficient use of the accurate model and, in general, leads to better results, faster. The method is general, and can be used with any surrogate model for optimisation.

The validation process of the model of colon colouration has shown that this model is able to generate the spectral reflectance of colon tissue. Moreover, the ranges of its parameters have been confirmed as histologically plausible, which means that the model can be used to predict the colours occurring in the normal colon tissue.

The next step in our approach is the inversion of the model described above in order to derive histological parameters for given tissue colours. The inverse mapping will then be used to create parametric maps, one for each parameter, which, at each pixel, show the magnitude of the relative histological parameter.

Acknowledgements. Jonathan Rowe did some of the work while visiting Darrell Whitley, funded in part from National Science Foundation grant number IIS-0117209. The colonoscopy spectra were kindly given to us by Kevin Schomacker of MediSpectra, Lexington MA.

References

1. Claridge, E., Cotton, S.D.: Developing a predictive model of human skin coloring. Proceedings of SPIE **2708** (1996) 814–825
2. Claridge, E., Cotton, S.D., Hall, P., Moncrieff, M.: From colour to tissue histology: Physics based interpretation of images of pigmented skin lesions. Medical Image Analysis **7** (2003) 489–502
3. Prahl, S.A., Keijzer, M., Jacques, S.L., Welch, A.J.: Monte carlo model of light propagation in tissue. In Mueller, G., Sliney, D., eds.: SPIE Proceedings of Dosimetry of Laser Radiation in Medicine and Biology. Volume IS 5. (1989) 102–111
4. Wang, L., Jacques, S.L.: Monte Carlo modelling of light transport in multi-layered tissues in standard C. Univ of Texas, MD Anderson cancer center (1998)
5. Prahl, S.A.: Mie theory. Oregon Medical Laser Centre, http://omlc.ogi.edu/software/mie/index.html (2000)
6. Ge, Z., Schomacker, K.T., Nishioka, N.S.: Identification of colonic dysplasia and neoplasia by diffuse reflectance spectroscopy and pattern recognition techniques. Applied Spectroscopy **52** (1998) 833–839
7. Rowe, J.E., Hidović, D.: An evolution strategy using a continuous version of the Gray-code neighbourhood distribution. To appear in Proceedings of GECCO (2004)

8. Kubelka, P., Munk, F.: Ein beitrag zur optik der farbanstriche. Zeitschrift für Technishen Physik **12** (1931) 593–601
9. Egan, W.G., Hilgeman, T.W.: Optical Properties of Inhomogeneous Materials. Academic Press (1979)
10. Jin, Y.: A comprehensive survey of fitness approximation in evolutionary computation. Soft Computing (2003) In press.
11. Jin, Y., Olhofer, M., Sendhoff, B.: A framework for evolutionary optimization with approximate fitness function. IEEE Transactions on Evolutionary Computation **6** (2002) 481–494
12. Brooker, A.J., Dennis, J., Frank, P.D., Serani, D.B., Torczon, V., Trosset, M.: A rigourous framework for optimization of expensive functions by surrogates. Structural Optimization **17** (1998) 1–13

Evolution-Based Deliberative Planning for Cooperating Unmanned Ground Vehicles in a Dynamic Environment

Talib Hussain, David Montana, and Gordon Vidaver

Department of Distributed Systems and Logistics, BBN Technologies
Cambridge MA, USA, 02138
{thussain, dmontana, gvidaver}@bbn.com

Abstract. Many challenges remain in the development of tactical planning systems that will enable automated, cooperative replanning of routes and mission assignments for multiple unmanned ground vehicles (UGVs) under changing environmental and tactical conditions. We have developed such a planning system that uses an evolutionary algorithm to assign waypoints and mission goals to multiple UGVs so that they jointly achieve a set of mission goals. Our evolutionary system applies domain-specific genetic operators, termed tactical advocates because they capture specific tactical behaviors, to make targeted improvements to plans. The plans are evaluated using a set of tactical critics that together comprise a multiobjective fitness function. Each critic evaluates a plan against criteria such as avoiding an enemy or meeting mission goals. Experimental results show that this approach produces high-quality plans with the potential for real-time dynamic replanning.

1 Introduction

Recent advances in technologies for the control of unmanned ground vehicles (UGVs) have demonstrated the ability to perform local path navigation while traversing unknown, off-road terrain. Moreover, these technologies permit simple longer-range path planning, such as navigation between human-specified waypoints. However, the challenge remains to develop technologies for automated generation of plans that result in the achievement of higher-level mission goals (such as reconnaissance, surveillance, and target acquisition) despite changing environmental conditions, evolving mission requirements, and the need to coordinate multiple UGVs [1].

In response to an environment and a set of mission requirements that are dynamically changing, the planning system must perform replanning of both the *reactive* (local) and *deliberative* (global) varieties. Examples of reactive replanning are when a UGV avoids an obstacle or turns to run away from an enemy. An example of deliberative replanning is when a UGV discovers a previously unknown enemy and modifies its entire path to circumvent the enemy and remain hidden en route to its next mission goal. Another, more complex, example of deliberative replanning is when a UGV, after discovering an enemy and realizing that it can no longer reach its next

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1017–1029, 2004.

mission goal in time, trades goals with another UGV that has a clear path to the first UGV's mission goal. While there has been some previous work done on deliberative planning for robots and UGVs, progress has been slow, with much more practical work in the reactive planning area.

Our approach is to view the entire deliberative planning problem as an optimization problem to determine an operation plan for multiple UGVs that achieves multiple mission goals while satisfying multiple tactical criteria as best as possible based upon the most recent environmental and tactical situation knowledge available. An *operation plan* is defined as a set of paths, one for each UGV, in which each path is a sequence of navigation waypoints. A *mission goal* is defined as a geographical location or area that must be visited, along with some measure of the time at which that area should be visited. A UGV may be assigned zero or more mission goals. A *tactical criterion* is defined as a property of an operation plan that is desirable in the context of the current state of the environment, such as enemy avoidance, hazard avoidance, stealth or rapid achievement of mission goals.

This deliberative UGV planning problem shares some important characteristics with a classic optimization problem, the vehicle routing problem with time windows (VRPTW) [2]. In both problems, multiple vehicles need to move in such a way as to arrive at particular locations during particular time windows. However, the UGV planning problem has some critical extra complications. One is that the paths between locations are not well defined, and the planning algorithm must find a good path over some combination of roads and off-road terrain. A second complication is that there are a greater number of criteria to consider in determining a good plan.

We use an evolutionary algorithm to search for a good solution. For such a complex optimization problem, an evolutionary algorithm is a good approach. In addition to their ability to search efficiently through large and complex spaces, evolutionary algorithms offer the advantage of being easily tailored to a particular domain for improved performance. We take advantage of this with our use of tactical advocates and tactical critics. The advocates are domain-specific mutations that modify a plan based on knowledge about good tactics. The critics compute the different evaluation metrics corresponding to different criteria of what constitutes a good plan. The structure of the software and algorithm makes it easy to add new advocates and critics and hence to incorporate domain knowledge. We discuss in detail this evolutionary algorithm approach, which we refer to as Advocates and Critics for Tactical Behaviors (ACTB), in Section 3.

To validate our approach, we have developed test scenarios in which multiple UGVs cooperate to solve complementary and competing mission goals while minimizing mission completion time as well as minimizing risk to mission success. One such scenario incorporates the actual terrain that the Army uses as a testbed for some of its UGVs. As we discuss in Section 4, the system has demonstrated that constantly improved plans can be quickly generated, both before and during plan execution, in response to changes in the tactical situation.

2 Background

The deliberative planning problem we are investigating seems like it should be amenable to a variety of well-studied techniques. However, we now argue that these techniques do not actually apply.

One set of approaches that do not apply is the traditional Artificial Intelligence (AI) planning algorithms. Classical planning [3], hierarchical-task-network planning [4], and case-based planning [5] use symbolic planning based on logic and reasoning. However, this problem is essentially numeric and hence not suited to reasoning about goals and subgoals. The higher-level strategic planning problem, how to decide what the mission goals are, is potentially well matched to AI planning techniques, but we are interested in the tactical planning problem where the mission goals are already known.

A second set of techniques that largely do not apply is those for coordinated robot planning. Many have collision avoidance during path planning as a primary concern [6]. In our problem, there is so much space compared to the number of vehicles that the low-probability case of a potential collision can be handled by reactive planning, and we place our deliberative planning emphasis on how to share the work rather than avoid collisions. Other multi-robot planning algorithms are concerned with formations and moving in unison rather than dividing the workload [7]. Mataric does investigate a variety of ways of coordinating robot behavior by dividing the work, e.g. [8]. However, this workload decomposition is generally reactive rather than deliberative, losing the benefits of planning ahead for multiple goals. Furthermore, path planning is treated as a separate problem, thus not considering issues such as an enemy between a UGV/robot and a nearby goal point when assigning goals.

Perhaps the work closest to what we are doing is that by Carnegie-Mellon University (CMU) on control of UGVs. The core of the CMU control system is the Distributed Architecture for Mobile Navigation (DAMN) [9]. Among other features, DAMN provides a sophisticated reactive control component. DAMN contains behaviors, each of which represent some higher-level navigation goals, such as 'road following', 'seeking the next navigation goal', 'obstacle avoidance', 'avoid hazards'. Each behavior provides a vote on the next direction to take, and a command arbiter decides upon the best direction, which is then taken by the UGV. While most of the behaviors are reactive, there is one behavioral input from a deliberative planner called the global navigator [10]. The global navigator is capable of determining a full path to a goal position using a D* (dynamic A*) search algorithm. However, this approach still does not incorporate as many criteria and as much information at the deliberative planning level as we believe are necessary to determine mission assignments and paths that are not fooled by local gradients.

As we mentioned in Section 1, the problem we are solving is to first order a combination of the vehicle routing problem and robotic path planning. Genetic algorithms have been used for each of these tasks in the past. An example of a genetic algorithm for path planning is [11]; an example of a genetic algorithm for vehicle routing is [12]. The novelty of this problem is jointly solving the two problems, plus being able to adapt the solution dynamically to a changing environment.

3 Technical Description

3.1 System Design

The ACTB system addresses the need to perform continual deliberative planning within a dynamic environment in which UGVs move and knowledge regarding the environment and tactical situation may change. We have developed a simulation-based system in which a deliberative planning process explicitly interacts with a simulated world environment in a continual cycle, as illustrated in Figure 1. The deliberative planning process uses the ACTB genetic algorithm to evolve multiple notional operation plans for a fixed number of generations. After the genetic run, the best plan is then adopted as the current execution plan. In the simulated world environment, the execution plan is communicated to the UGVs, which use a simple (non-reactive) execution model to visit their waypoints. s execution proceeds, simulated world events, such as the discovery of a new enemy location, may occur. These events trigger the deliberative planning process to evolve a new plan that incorporates the new tactical situation. Additionally, at regular intervals, the execution process may be suspended and the deliberative process executed to explore further improvements to the current operational plan. The population of the genetic algorithm is persistent across runs.

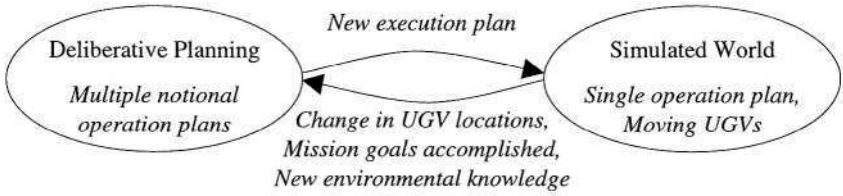


Fig. 1. Interaction between deliberative planning process and simulated world

ACTB is programmed in Java 1.4 and the simulation environment uses the OpenMap™ geographical system [13] to represent terrain information, provide basic functionality for making geographic inquiries, and provide a graphical interface.

3.2 ACTB Genetic Algorithm Design

The ACTB genetic algorithm is based upon the fundamental notion that significant improvements to a plan may be made through a succession of small, goal-directed changes. These goal-directed changes are made using domain-specific genetic operators termed *tactical advocates.* A tactical advocate promotes the use of a specific tactical behavior during deliberative planning, where a *tactical behavior* is defined as an action performed by a UGV that may generally satisfy one or more tactical criteria. For example, a tactical behavior may be to follow a road, as opposed to travelling cross-country. Such a behavior is tactical in that it may lead to an improvement in the

speed with which the UGV accomplishes its tasks, or the rapidity with which it moves away from a known enemy. In addition to the tactical advocates, traditional domain-generic operators are also used to augment the search capabilities of the system and maintain diversity.

The ACTB genetic algorithm accommodates the multiobjective nature of the deliberative planning problem through the use of multiple, distinct evaluation components to determine fitness, thereby following an established approach for solving multiobjective optimization problems [14]. Specifically, a tactical critic represents a domain-specific evaluation component that computes a single term in a fitness function. Each tactical critic evaluates how well a given operation plan satisfies a tactical criterion. For instance, a critic for safety may evaluate a plan to determine how much danger the UGVs are placed in due to traveling too close to a known enemy. The outputs of multiple critics are combined using a weighted sum to form a single fitness value. In the military context, the weights associated with the critics reflect the tactical priorities of the operation.

The ACTB genetic algorithm accommodates the constraint-based nature of the deliberative planning problem by allowing ostensibly "illegal" individuals into the population (i.e., those that violate constraints) and using fitness values to reflect the magnitude of the violations. This is an example of an established approach for handling constraints [15]. Specifically, when a tactical critic evaluates a given operation plan against a tactical criterion, it assigns a penalty if the plan violates that criterion. For example, a critic to evaluate whether the path is traversable will accept a path that crosses water (an untraversable terrain), but assign a high penalty. To enable a relative judgement amongst "illegal" plans, critics will typically assign a penalty that is proportional to the degree of the violation. For example, the amount of distance "traveled" in water will determine the magnitude of the penalty.

An important property of the tactical critics is that they exploit the most recently available environmental knowledge. As such, the fitness of an individual plan in the population may vary whenever the environment state varies. In the simulated environment, the genetic algorithm may be run many times, each time for a small number of generations. The population is persistent across runs, but may require re-evaluation at the beginning of a run if the environment state has changed.

3.3 Genetic Representation

Given n UGVs, a genome is defined as a set of n chromosomes, where each chromosome defines the path for one of the UGVs as a variable length sequence of geographical locations, or *waypoints*. For the purposes of evaluating the fitness of a genome, every successive pair of waypoints is assumed to be connected with a straight line. Each chromosome therefore defines a piece-wise linear directed path. An important aspect of the genetic representation is that the first waypoint in each path represents the *next* waypoint of the corresponding UGV. The first segment in a path is inferred to be the straight line between the UGV's current location and the first waypoint in the path.

In order to enable effective genetic manipulations, the representation has three types of waypoints, each representing a different conceptual aspect of a path. A *mission-point* is a waypoint that attempts to satisfy a given mission goal at a specific location, and the sequence of mission-points determines the order in which the UGV accomplishes its assigned missions. A *route-point* is a waypoint that marks a specific location on the map, and a sequence of route-points is used to determine the general route followed by the vehicle between two mission-points. A *travel-point* is a waypoint that marks a specific location on the map, and a sequence of travel-points is used to specify a detailed route followed between two route-points. A key feature of travel points is that they are not available for selection as points of genetic manipulation. Rather, they are used to incorporate specific path segments between two consecutive mission or route-points. The relative benefits of these segments may then be evaluated through genetic search. In the current system, route-points are used by the road-following advocate (see below) to represent complex road segments.

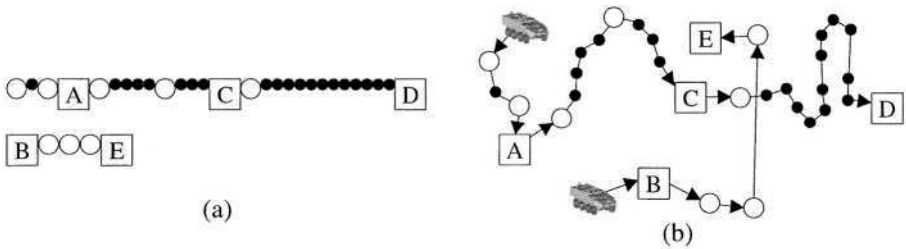


Fig. 2. (a) Genome with two chromosomes as sequences of three types of waypoints, and (b) inferred path for each UGV from its current location

Figure 2a illustrates a sample genome that has two chromosomes. Each mission-point is flagged with the name of the mission at that location (e.g., A, B, etc.) and is represented as a square. Each route-point is represented as an open circle, and each travel-point is represented as a solid circle. Figure 2b illustrates the geographical locations of the waypoints and the inferred directed path for each UGV among its waypoints, starting at the UGV's current location. Note that UGV paths may cross.

The three types of waypoints are used to enable genetic manipulations targeted to different levels of planning. For example, manipulation of mission-points and their order performs the task of scheduling missions, while manipulation of route-points serves performs the task of route planning. In the context of a UGV platform, no distinction is required between the three types of waypoints.

3.4 Tactical Advocates

Three tactical advocates are used in the current ACTB system. The *Mission-allocation* advocate exploits knowledge of the mission goals and their requirements to allocate mission goals to the UGVs. It operates in two modes. The first mode is selected if there is an outstanding mission goal that has not been assigned to any UGV within a given genome. In this mode, the advocate assigns an outstanding mission goal to a randomly selected UGV by inserting a mission-point into the path next to

the existing route-point or mission-point that is closest to the mission. The order of the new mission is determined by the existing information in the UGV's path. The second mode is used when all missions have been assigned within a plan. The advocate randomly removes a sequence of mission-points (one or more) and all intermediate waypoints from a path, and inserts that sequence before or after a randomly chosen mission-point on a randomly chosen path in the plan. Thus, the missions may be inserted within the same UGV's path, thereby performing an effective re-ordering of the mission goals, or in the path of a different UGV, thereby performing a switch of mission goals between UGVs. In this offspring, the new ordering of missions is randomly determined.

The *Avoid-untraversable* advocate exploits terrain knowledge and a model of the movement capabilities of the UGVs to determine routes that do not have waypoints in untraversable terrain. For instance, rivers and lakes may be untraversable. The advocate identifies all waypoints over all chromosomes in a genome that lie in untraversable terrain. It randomly selects one of these waypoints and moves it to a location on traversable terrain. The new location is selected by searching on an arc towards the "traversable-predecessor" of the selected untraversable-waypoint. Any intervening waypoints are eliminated (since they necessarily would have been untraversable).

The *Road-following* advocate exploits knowledge of the road network to determine a path segment between any given pair of mission or route points that makes maximal use of roads. All roads are represented symbolically within Openmap. The road-following advocate randomly chooses two mission or route points on a randomly chosen chromosome. Using deterministic routines that query the Openmap road representation, the advocate first determines the closest road point to each selected waypoint and then obtains the shortest road path between those road points. This road path is represented as a sequence of travel points with route-points at the ends. The new sequence replaces the path between the original selection points.

The use of travel-points rather than route-points to determine road following is important. A segment of road may be highly curved or irregular, and as such require a large number of points in order to specify that segment in a piece-wise linear manner. If the travel-points were included as possible points of selection by the other advocates and genetic operators, the process of selecting waypoints for adaptation would be overwhelmed by the large number of travel-points. For example, the nudge-waypoint operator (see below) would spend the majority of its time moving road points, and thus be highly ineffective at optimizing the route between mission goals.

A first glance, the road lookups of the road following advocate seem to serve a purpose similar to the shortest path lookups of existing planning techniques. However, the road lookup is limited to identifying only small road segments, and has no impact upon the exploration of cross-country paths by other genetic operators. Rather than simply looking up a shortest path between two mission goals, which may be tactically poor, the ACTB genetic algorithm uses multiple genetic operators to determine routes, and creates routes by making a number of small changes at random locations in the chromosome. This enables the GA to explore a wide variety of routes between the two mission goals and adapt that route according to multiple tactical criteria. For example, some segments may result in a poor fitness according to one

tactical critic, and eventual removal or modification of those segments through genetic operators and advocates may produce an improved path according to that critic.

3.5 Genetic Operators

Several traditional mutation and crossover genetic operators are used within ACTB to maintain enough variability in the population so that the tactical advocates continue to make novel plans rather than continually rehashing old ones. Three mutation operators are used, each of which performs a mutation on a randomly chosen chromosome within the genome. Only mission-points and route-points are manipulated. The *insert-waypoint* mutation operator will randomly select a waypoint on the chromosome and insert a single waypoint before or after that point. The geographic location of new waypoint will be a small random distance in a random direction from the line connecting its neighbors. The *remove-section* mutation operator will randomly select two waypoints in the chromosome and remove them and all waypoints between them. The *nudge-waypoint* mutation operator will randomly select a waypoint from the chromosome and modify its geographical location slightly in a random direction. Two crossover operators are used. The *path-crossover* operator is applied to a single genome parent, and performs variable-length one-point crossover between two randomly chosen chromosomes within the genome. The *plan-crossover* operator is applied to two genome parents, and performs variable-length one-point crossover between a randomly chosen chromosome in one parent and a randomly chosen chromosome in the other.

3.6 Tactical Critics

Five tactical critics are used in the current ACTB system, and all return evaluations that are greater than or equal to 0, where lower numbers indicate better plans.

The *Traversability* critic exploits terrain knowledge and a model of the movement capabilities of the UGVs to identify all portions of the path that cross untraversable terrain. It returns a penalty proportional to the distance traveled on untraversable terrain over all chromosomes (i.e., we allow a path to cross untraversable terrain but penalize accordingly).

The *Safety* critic exploits knowledge of the known enemy locations and a model of enemy capabilities to evaluate whether a given plan puts one or more UGVs in danger by placing them too close to a known enemy. It returns a penalty proportional to the distance traveled within danger range of any enemies over all chromosomes (i.e., we allow a path to cross dangerously close to enemies, but penalize accordingly).

The *Stealth* critic exploits knowledge of the known enemy locations and line-of-sight computations to evaluate whether a given plan puts one or more UGVs at risk by placing them in the line-of-sight of a known enemy. Line-of-sight is computed using an Openmap routine and a model of the surveillance capabilities of the enemy. The advocate returns a penalty proportional to the distance traveled within surveillance range of the enemy (i.e., we allow a path to cross within sight of enemies, but penalize accordingly).

The *Mission-success, Total-duration* and *Max-duration* critics use a deterministic greedy algorithm to interpret how a given chromosome would be executed by a UGV. The algorithm assumes a model of UGV movement speed over different terrain types and evaluates the travel time between successive mission-points based upon the distances and terrain traversed over the (piece-wise linear) path between them. Each UGV is assumed to travel as fast as possible between mission points, and then wait as little as needed (if early) to meet the time window requirement (i.e., the greedy choice). Thus, no special representation of time windows is required in the genome. The mission-success critic evaluates how well a given plan comes to accomplishing all mission goals, and returns a penalty proportional to the number of failed goals and degree of failure. The total-duration critic evaluates how long each UGV takes to execute its chromosome, and returns the sum of the durations of all chromosomes. The max-duration critic evaluates how long each UGV takes to execute its chromosome, and returns the longest duration over all chromosomes.

4 Experimental Results

The ACTB system was tested under four conditions to demonstrate the effectiveness of the tactical critics for multiobjective optimization and examine the search capabilities of ACTB when using tactical advocates in conjunction with traditional genetic operators over using traditional genetic operators alone. The experiment examined the basic tactical route planning capabilities of the ACTB system. Time scheduling aspects of the problem were minimized by making the mission time windows very wide. However, path duration was still an important factor (i.e., do all the missions as soon as possible).

In all conditions, a steady-state genetic algorithm was run using a fixed population size of 50, fitness-proportional selection was used, and offspring competed with all members of the population. Most advocates and genetic operators were applied with the same likelihood of selection (i.e., 1.0). To encourage the system to explore complex paths, *insert-waypoint* was applied with twice the likelihood of the above (i.e., 2.0), and *remove-section* with half the likelihood (i.e., 0.5). Critic weights were selected to assign a very high penalty to untraversable portions of the routes and to missed missions, a moderate penalty to exposure to the enemy (i.e., completing the mission is more important than avoiding the enemy), and a small penalty to path duration; the penalty for *maximum-duration* was weighted twice as strongly as *total-duration* to encourage a more equal distribution of mission goals among UGVs.

Final plans developed in the four experimental conditions are illustrated in Figures 3 and 4. In the first pair of conditions (Figure 3a and 3b), the system examined basic routing in a simple situation with no known enemies. In the second pair of conditions (Figure 4a and 4b), the system examined tactical routing in a situation with two known enemies. In the figures, three friendly UGVs are located in the top left and each UGVs path is indicated by a different line thickness; off-road terrain is shown in white, and all intersections of road with water are bridges. Travel on-road is assumed to be roughly 10 times faster than travel off-road. These figures show that in

all conditions, the system was able to evolve plans that were traversable, met all mission goals and distributed mission goals among all three UGVs. In both enemy conditions, the evolved paths avoided the known enemies.

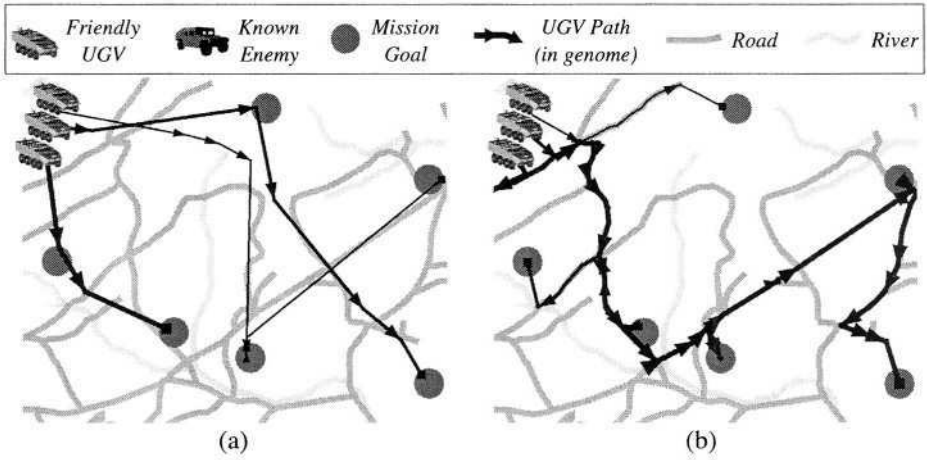


Fig. 3. Evolved paths with no known enemies using (a) traditional genetic operators and (b) both traditional genetic operators and tactical advocates

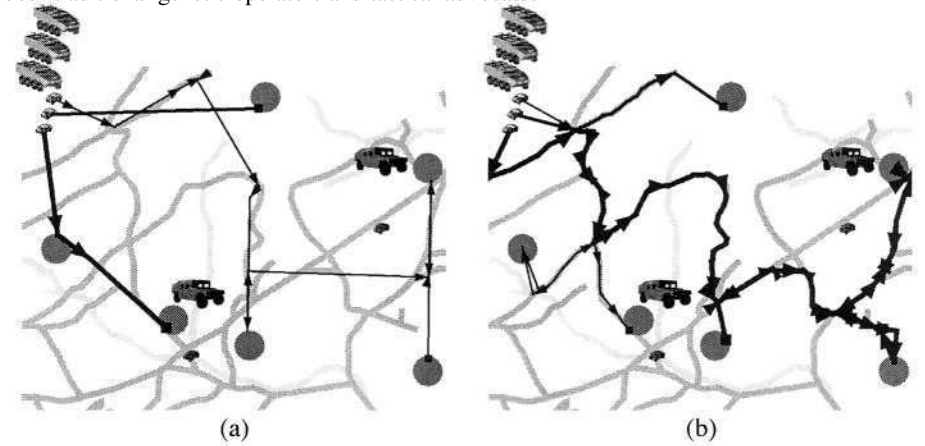


Fig. 4. Evolved paths with two known enemies using (a) traditional genetic operators and (b) both traditional operators and tactical advocates

Figure 5 illustrates the fitness of the best individual each generation in each experimental condition. Each condition was run for 30000 generations to ensure convergence, but all conditions had converged by 7000 generations. The range of fitness values was very large due to high penalty values assigned by the tactical critics, and the results are graphed on a logarithmic scale to emphasize the changes over evolution. A fitness value under 1000 indicates no penalty except for duration of path. The ACTB system clearly demonstrated better plans when using advocates, as illustrated by the better (i.e., lower) fitness values overall and by the rapid achievement of

a plan with no major penalties around 1000 generations as opposed to over 4000 generation for the traditional conditions. The final plan generated in the advocate conditions completed all missions in roughly half the time of the plans generated in the traditional conditions. As illustrated in Figures 3b and 4b, this improvement is clearly due to improved road following.

We have also tested the system in a dynamic simulation mode, as described earlier, in which new enemies may be detected as the UGVs are executing an operation plan. The ACTB system has demonstrated the capability for rapid and effective replanning in response to these changes in the tactical situation, as illustrated in Figure 6. Figure 6a illustrates a plan under execution immediately before the discovery of the enemy. Figure 6b illustrates the re-planning activity initiated upon the discovery (after a few generations). Figure 6c illustrates the new plan generated after 300 generations and passed to the UGVs for execution. Note that after replanning, all UGV paths avoid the area surrounding the enemy.

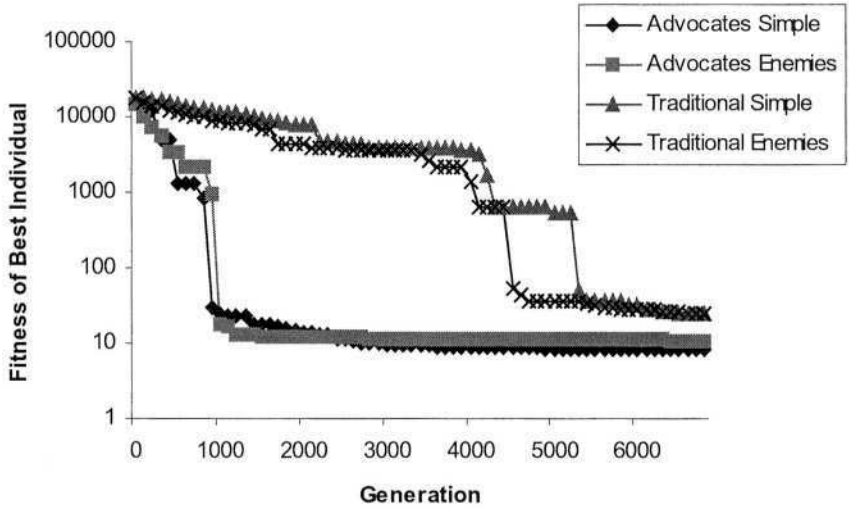


Fig. 5. Fitness of best individual by generation for all four experimental conditions

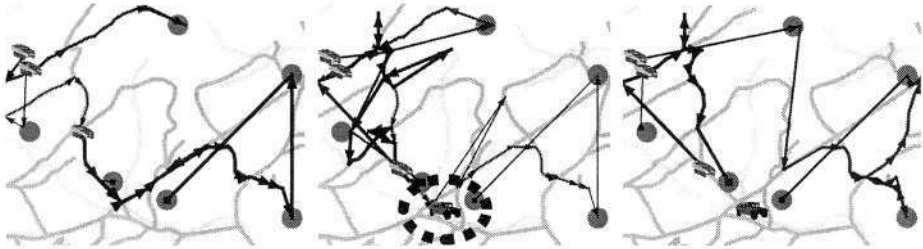


Fig. 6. Sequence of three plans illustrating (a) execution plan before, (b) new plan immediately upon and (c) new plan shortly after discovery of a new enemy

5 Conclusion

We have implemented and evaluated an approach to deliberative planning for coordinating UGVs. This approach is based on representing the planning problem as an optimization problem and using a genetic algorithm to search for a good solution. Multiple evaluation components, called tactical critics, enable the evolution of plans satisfying multiple tactical constraints. Domain-specific operators, called tactical advocates, greatly speed the search process yielding rapid plan turnaround. By continually searching for improvements to the plan, we ensure that the plan will adapt to changes in the tactical situation. We have provided preliminary evidence that the ACTB system maintains good plans in response to such changes. We are exploring the development of advocates and critics for additional tactical behaviors and mechanisms for distributing the evolutionary algorithm to make ACTB amenable to implementation within UGV platforms directly.

Acknowledgements. We would like to acknowledge the efforts of Stephen Milligan, Richard Lazarus and Disk Estrada in the development of the ideas presented in this paper, and the efforts of Aaron Iba, Brian Krisler and Sarah Siracuse in the development of the simulation environment used to test our approach.

References

1. National Research Council Staff: Technology Development for Army Unmanned Ground Vehicles. National Academies Press, Washington, D.C. (2002)
2. Solomon, M.: Algorithms for the vehicle routing problem with time windows. Transportation Science **29** (1995) 156-166
3. Fikes, R., Nilsson, N.: STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence **2** (1971) 189-208
4. Nau, D., Cao, Y., Lotem, A., Muñoz-Avila, H.: SHOP: Simple hierarchical ordered planner. In: International Joint Conference on Artificial Intelligence (1999) 968-973
5. Spalazzi, L.: A survey on case-based planning. Art. Intelligence Review **16** (2001) 3-36
6. Svestka, P., Overmars, M.: Coordinated path planning for multiple robots. Robotics and Autonomous Systems **23** (1998) 125-152
7. Balch, T., Arkin, R.: Behavior-based formation control for multi-robot teams. IEEE Transactions on Robotics and Automation **14** (1998) 926-939
8. Gerkey, B., Mataric, M.: Sold!: Auction methods for multi-robot coordination. IEEE Transactions on Robotics and Automation **18** (2002) 758-786
9. Rosenblatt, J.: The distributed architecture for mobile navigation. Journal of Experimental and Theoretical Artificial Intelligence **9** (1997) 339-360
10. Brumitt, B, Stentz, A.: Dynamic mission planning for multiple mobile robots. In: IEEE International Conference on Robotics and Automation (1996) 2396-2401
11. Ashiru, I., Czarnecki, C., Routen, T.: Characteristics of a genetic based approach to path planning for mobile robots. Network and Computer Applications **19** (1996) 149-169

12. Baker, B., Ayechew, M.: A genetic algorithm for the vehicle routing problem. Computers and Operations Research **30** (2003) 787-800
13. BBN Technologies. Openmap™: Open System Mapping Technology. openmap.bbn.com
14. Coello, C.: A comprehensive survey of evolutionary-based multiobjective optimization techniques. Knowledge and Information Systems **1** (1999) 269-308
15. Coello, C.: A survey of constraint handling techniques used with evolutionary algorithms. Technical Report Lania-RI-99-04, Laboratorio Nacional de Informática Avanzada (1999)

Optimized Design of MEMS by Evolutionary Multi-objective Optimization with Interactive Evolutionary Computation

Raffi Kamalian[1], Hideyuki Takagi[2], and Alice M. Agogino[1]

[1] University of California, Berkeley, CA, 94720, USA
{raffi,aagogino}@me.berkeley.edu,
[2] Kyushu University, Fukuoka, 815-8540, Japan
takagi@design.kyushu-u.ac.jp

Abstract. We combine interactive evolutionary computation (IEC) with existing evolutionary synthesis software for the design of micromachined resonators and evaluate its effectiveness using human evaluation of the final designs and a test for statistical significance of the improvements. The addition of IEC produces superior designs with fewer potential design or manufacturing problems than those produced through the evolutionary synthesis software alone as it takes advantage of the human ability to perceive design flaws that cannot currently be simulated. A user study has been performed to compare the effectiveness of the IEC enhanced software with the non-interactive software. The results show that the IEC-enhanced synthesis software creates a statistically significant greater number of designs rated *best* by users.

1 Introduction

Microelectrical Mechanical Systems (MEMS) is an emerging field of research with application in a wide variety of areas such as Radio Frequency (RF) communications, optical networking, and environmental monitoring. Current computer-aided design tools for MEMS design are rudimentary and much of current design is based on engineering experience and *back of the envelope* calculations. We have developed automated synthesis tools that allow for the creation of complex devices with desired performance, optimized for a number of design constraints and competing performance goals [3,8,9]. Other notable research in using evolutionary approaches in the MEMS field includes the work of Li [4] and Ma [5]. This work differs from ours in that the focus is on mask layout, fabrication and parametric modeling – but not design synthesis. At a higher level, Fan has worked on using system-level synthesis of RF bandpass filters made up of MEMS components using genetic programming techniques on bond graphs [2].

One of the limitations of our current evolutionary multi-objective optimization (EMO) approach is that it depends on simulation software to evaluate design quality. Unfortunately, there are many design issues that cannot be currently detected by the simulation software. These issues lead to either poor performance

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1030–1041, 2004.

or premature failure. While many of these potential problems are clearly visible to a human user they would be extremely difficult, if not impossible, to mathematically model and simulate.

We propose to combine the EMO approach with interactive evolutionary computation (IEC) techniques to embed the human user's visual inspection and domain knowledge into the computer-aided MEMS design process. IEC uses evolutionary optimization to evolve a group of designs similar to EMO, but instead relies on a human user to rate each design based on his or her subjective evaluation for performance and shape. This allows the human's judgment and preferences to further shape which designs are developed, avoiding potential design flaws that are hard to model analytically.

2 MEMS Synthesis

The design example chosen for synthesis is a simple surface micro-machined resonating mass. This mass is suspended above the substrate by four legs, comprised of several beam segments. The center mass has two electrostatic "comb drives" attached to it in order to facilitate actuation and capacitive position sensing during characterization. Currently the center mass and comb drive geometry are fixed, only the contents of the four legs are variable. Each leg is comprised by a variable number of beams, and each beam has its own length, width, and angle as free variables (see Fig. 1).

Four objective functions are formulated as a minimization of the distances to four goals: resonant frequency (100 kHz), suspension stiffness in the lateral direction (100 N/m), stiffness in longitudinal direction (1 N/m), and device area (device area goal = 0, i.e. area is minimized). The device area is defined by the area contained within a rectangle bounding the resonator's center mass, comb drives and beams, but not the anchors and contact pads.

Table 1. Design parameters/constraints used for MEMS resonator synthesis comparison.

Parameter Name	Value
Center mass	5.3066e-011 kg
Leg symmetry constraint	On
Manhattan angle only constraint	Off
Max number of beams per leg	7
Min number of beams per leg	1
Max beam length	$100\mu m$
Min beam length	$10\mu m$
Max beam width	$10\mu m$
Min beam width	$2\mu m$
Max beam angle	$\pi/2$
Min beam angle	$-\pi/2$

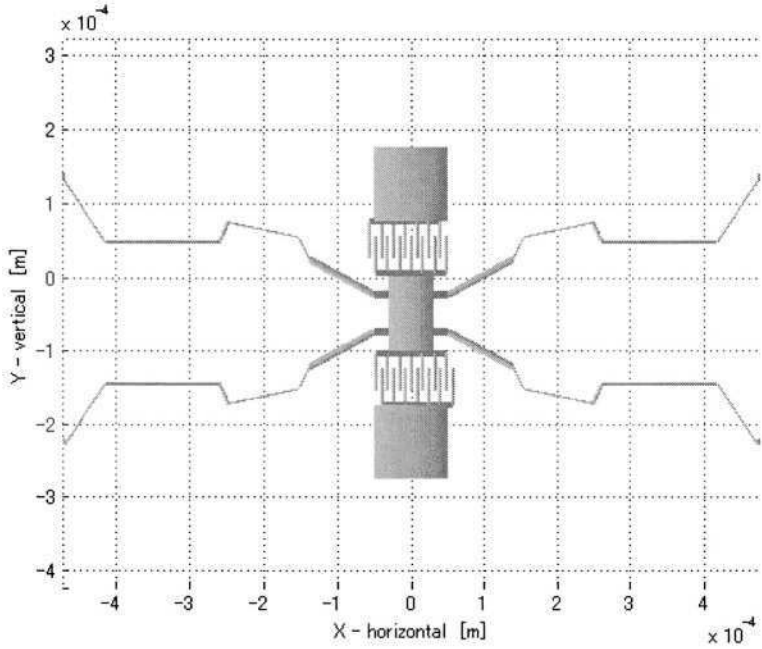


Fig. 1. Example of a MEMS resonator synthesized with EMO software, with symmetric leg constraint applied.

Each beam variable has a set of inequality constraints (max/min length, width and angle), and there is a limit on the number of beams per leg (see Table 1). The center mass is considered a parameter and not a design variable for this research. Basic geometrical checking is performed to prevent beams from crossing each other as such designs could not be fabricated or operated. Furthermore, different symmetry and angle constraints are considered, such as symmetric legs or Manhattan angle constraints. For this research, we limit ourselves to symmetric legs only.

3 Evolutionary Optimization for MEMS

3.1 Evolutionary Multi-objective Optimization

We use a multi-objective genetic algorithm (MOGA) as our EMO approach to developing a population of optimal solutions (see Fig. 2). Given a higher-level description of the device's desired behavior, an initial population of candidate designs is generated randomly from a number of available components such as anchors, beams, electrostatic gaps, and combs. Each design is checked for basic geometrical validity, and our MEMS simulation tool, SUGAR developed at UC

Berkeley [1], evaluates its performance for the multiple objectives: area, resonant frequency, and stiffness. The MOGA is then applied to the initial population to iteratively search for functional designs by applying the genetic operations of selection with elitism, crossover, and mutation to create the next generation of designs. This process continues until an optimal set of Pareto optimal solutions is synthesized.

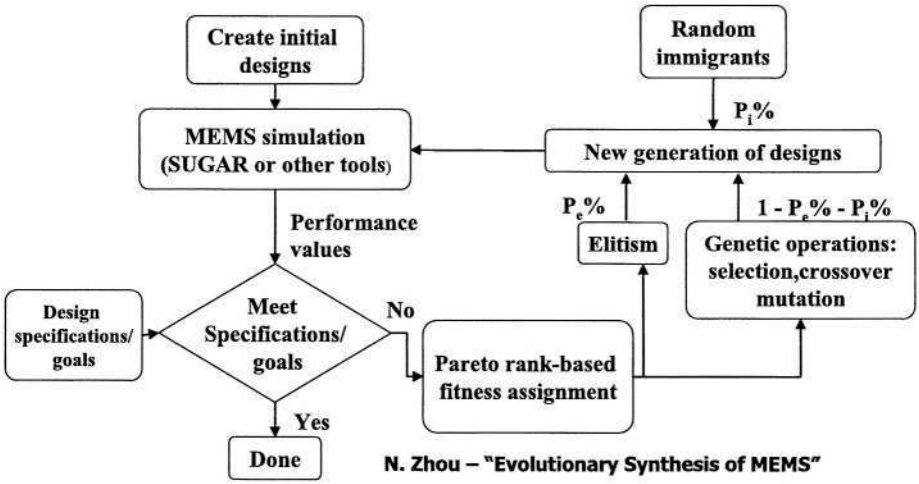


Fig. 2. Evolutionary synthesis.

In order to compare an EMO-only approach to the EMO+IEC approach, we first solve the MEMS resonator example with four objectives using a MOGA implementation with an initial population size of 400, and using the genetic operators of cross-over and mutation. A non-dominated or Pareto set of approximately 60-100 designs is achieved after 30 generations for this example.

Although these designs are Pareto optimal in the solution space according to the objective performance as simulated in SUGAR, they may contain flaws that prevent them from being suitable designs for fabrication. These flaws include potential stress concentrations, *near misses* where two legs come very close to each other without actually crossing in the static case, but may collide while the structure is being resonated, as well as other flaws or features that a human engineer may reject based on previous experience or their engineering knowledge. Of the 60-80 designs in a typical Pareto set returned by the EMO, only 25-30 are within 50% of our most critical objective - resonant frequency. Of these, only approximately four or five designs are free of non-simulated design flaws.

The current simulation software has no means to simulate stress concentrations, transient responses of structures, and other features that a designer might

perceive as *awkward* or difficult to fabricate or use. Therefore alternative means of evaluating a design must be sought.

3.2 Interactive Evolutionary Computation (IEC)

IEC is a method for optimizing a system using subjective human evaluation as part of the optimization process. It is well suited for optimizing systems whose evaluation criteria are preferential or subjective, such as graphics, music and design, and systems that can be evaluated based on expert's domain knowledge. Fields in which this technology has been applied includes graphic arts and animation, 3-D CG lighting, music, editorial design, industrial design, facial image generation, speech and image processing, hearing aid fitting, virtual reality, media database retrieval, data mining, control and robotics, food industry, geophysics, education, entertainment, social system, and others [7].

The IEC implementation used for this research is a single objective genetic algorithm, similar in structure to the MOGA, except instead of using a simulation tool to gauge the performance of a particular design for multiple objectives, the design is given a single integrated preference score by an IEC human user, and this score is used as the sole objective for ranking design individuals by fitness. In this case, IEC is used to measure the human user's satisfaction with a particular design based on its shape and simulator performance. This allows human knowledge and expertise to be embedded into the synthesis process.

3.3 Integration

Human fatigue is one of the difficulties faced in implementing an IEC approach [7]. Human interaction is required for every evaluation, thus limiting the population size and number of generations that can be used. The current EMO requires on the order of 12,000 evaluations (a population size of 400 over 30 generations) to evolve a group of good solutions from randomly generated initial design individuals. This magnitude of evaluation is not feasible for human interaction, therefore randomly generated starting points for IEC is not practical.

One solution to this problem is to combine the non-interactive EMO and IEC together to use the best attributes of each: the tireless, rapid synthesis of EMO with the ability of IEC to overcome many design flaws and embed human knowledge. The method of combining EMO and IEC chosen was serial; EMO is run, and the individuals of its final evolutionary generation are then used by IEC as an initial population for further interactive evolution. This integration of EMO and IEC is illustrated in Fig. 3.

As the population produced by the EMO process is much larger than that of IEC, a manual data selection step between the two components is added. This allows the human user to select only the designs that are contained within a hyper-rectangle in the objective space (see Fig. 4). Although we use a hyper-rectangle in our experiment, the integrated EMO+IEC concept need not restrict the shape of data selection area.

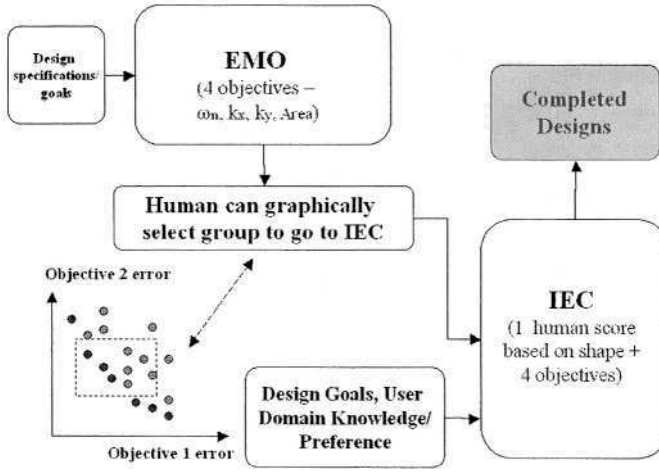


Fig. 3. Integration of EMO system and IEC system in our experiment.

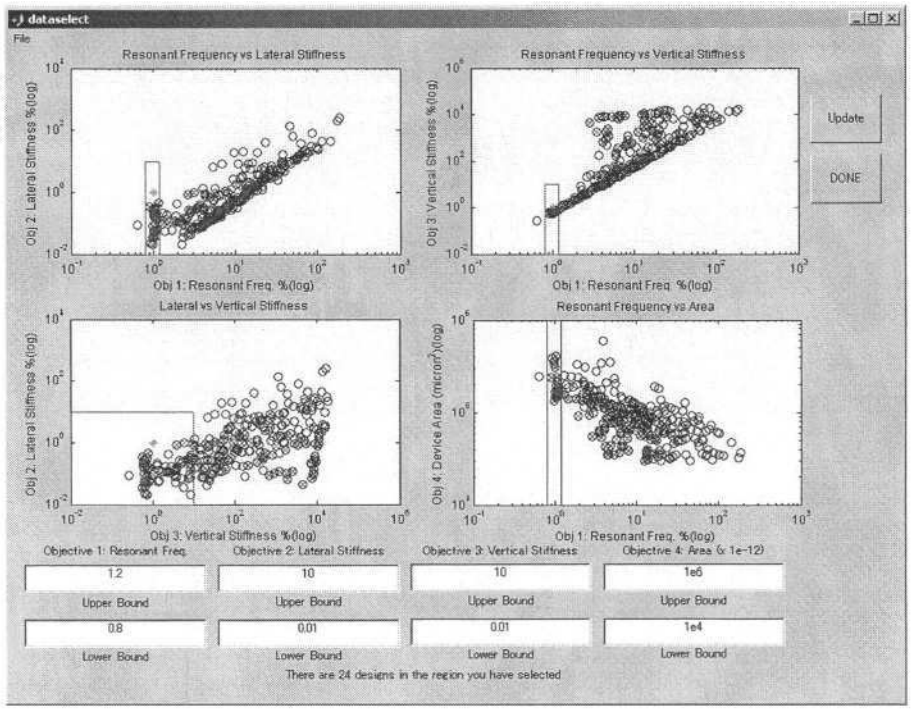


Fig. 4. Data Selection interface to select initial population for IEC.

This step serves to cull the initial population for IEC from a wide array of designs spread around the objective space down to a smaller number of high ranking designs centered around the objective goals. For example, there is little value in further evolving a design that has a resonant frequency that is an order of magnitude or more away from our goal resonant frequency.

Once a region of the EMO solution space is selected, the design candidates contained within it are passed on to the IEC component. These designs serve as a pool from which the initial population is drawn as well as new immigrants in later IEC generations.

For the MEMS resonator example, the IEC interface presents the user with nine designs at a time; each design is graphically displayed, and its performance simulated by SUGAR is presented as a percentage of the goal for each of the four objectives (Fig. 5). The users select a preference score from '*1*' (worst) to '*5*' (best) based on their impression of the shape and performance numbers. As a larger population size will give better, more diverse results, the IEC population size can be set to more than nine. For this paper, a population size of twenty-seven was used; therefore three windows of nine designs each were presented to the user for each evolutional generation.

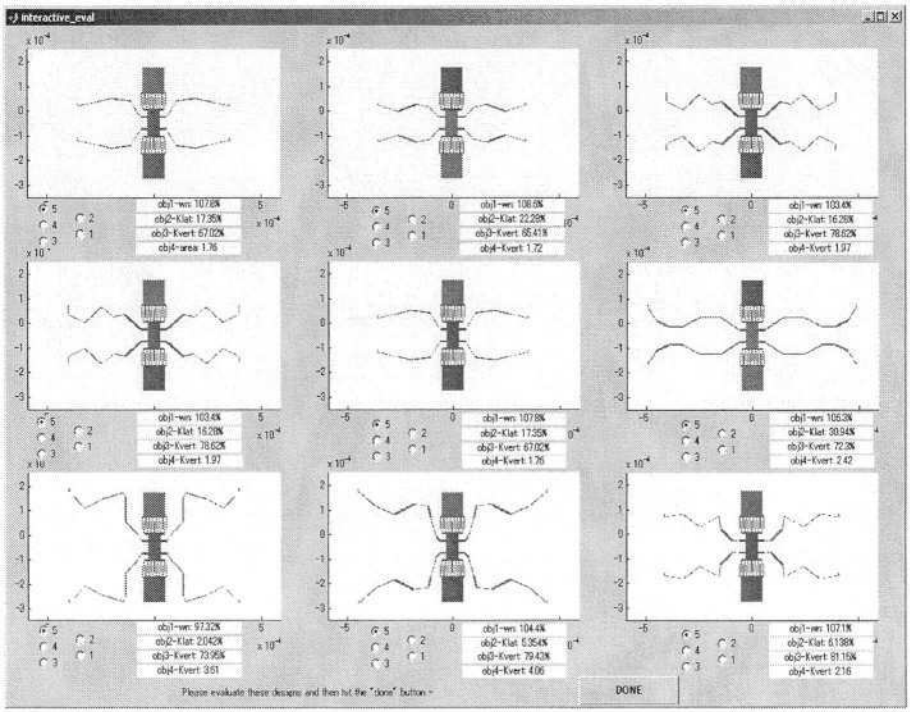


Fig. 5. Rating window for IEC; user gives preference score to nine designs at a time based on shape and performance.

At the completion of each generation's evaluation, the user preference scores are used by the EMO to evolve the next generation. This process continues until the user chooses to end the evolution process.

4 Experimental Evaluation

4.1 Experimental Setup

To gauge the effectiveness of the EMO+IEC implementation, a blind evaluation test for EMO+IEC vs. EMO-alone was performed. First, the MEMS design with EMO-only was conducted, and then IEC applied to the EMO output, using the proposed approach. Designs obtained by both approaches were visually evaluated based on users' domain knowledge. Statistical tests were applied to the difference of evaluation to both approaches.

Users started with an identical population of 400 resonators with symmetric legs generated with EMO. They selected a region to serve as the IEC initial population and scored designs for several generations. Users continued until either satisfied with the results or until the 10th generation was reached.

As a rule of thumb, IEC users were instructed to focus their preference scoring on the resonant frequency, the primary objective of the resonator design, and modify that score up or down based on the performance of the other objectives as well as the user's subjective evaluation of the design shape based on their domain knowledge.

A subjective test for comparison by the same user was conducted after each IEC test was completed. The population of the final generation of EMO+IEC designs was compared side by side to the EMO-only final Pareto set, using the same interface as the IEC window. The EMO+IEC and EMO-only designs were interspersed to ensure unbiased and consistent scoring between the two sets of designs. The users were not informed as to which approach produced any of the designs being evaluated.

Due to the large size of the Pareto design sets – 60 to 80 returned by the EMO-only step in the process for the 4 objectives – only a subset was shown to the user for scoring in this final subjective comparison with EMO+IEC. This subset was generated by taking only the members of the Pareto set within 50% of the resonant frequency goal, reducing the number from 60-80 down to a more manageable 25-35. This step removes the irrelevant designs, only requiring a human score on potentially good designs. This step can be justified because no user will give a resonator design significantly deviating from the primary objective goal a '*5*' score, and it saves the human user the tedium of scoring dozens of poor performing designs; recall that the Pareto set contains designs that are optimal in one or more objective, but could perform very poorly in the other objectives.

The scores given to each approach were tallied and compared. The number of '*5*' scores given by the users for each approach was chosen as the primary metric for effectiveness as we believed that the number of highly ranked feasible

designs best measures the ability of the EMO+IEC system to further hone designs developed by the EMO and in incorporate design issues that are difficult, if not impossible, to be numerically optimized.

The degree of MEMS experience in the background of the users was also recorded. Although one might argue that experts with MEMS design experience are better suited to rate MEMS design candidates than non-experts, we note that many of the potential design flaws identified can be observed by basic human visual recognition without necessarily requiring extensive training and experience in the specific field of MEMS. Thus this raises the question of whether non-experts might be trained to recognize good and bad resonator features and perform comparably to experienced MEMS designers in this process.

4.2 Experimental Results

Subjective tests and comparisons were performed on eleven engineering graduate students from the University of California at Berkeley. The number of the best scores, '*5*', given to the designs in the Pareto set of the final generation of the EMO-only and the EMO+IEC tests are presented in Table 2. Here, '+1', '0', and '–1' in the sign column mean that the number of *5*'s in the EMO+IEC Pareto set is more, equivalent, or less than that of EMO-only, respectively.

Table 2. Test results for the user evaluations of MEMS designs with EMO-only and IEC+EMO.

User	Expert?	IEC+EMO: # of *5*'s	EMO: # of *5*'s	sign
1	Y	7	9	−1
2	Y	12	6	1
3	Y	7	3	1
4	N	6	2	1
5	Y	4	4	0
6	Y	11	9	1
7	N	8	7	1
8	Y	1	0	1
9	N	6	3	1
10	N	12	7	1
11	N	9	2	1

An evaluation of the results in Table 2 shows that the EMO+IEC is significantly better than the EMO-only using both the *sign* test [6] ($p < 0.02$), and the *Wilcoxon Matched-Pairs Signed-Ranks* test [6] ($p < 0.01$).

Fig. 6(a) shows an example of a design in the Pareto set returned by the EMO that was given a low score of *'1'* by one of the users. In this case, the user penalized the presence of a sharp corner in the meandering springs due to the potential of the associated stress concentration to lead to premature stress

failure. Fig. 6(b) shows an example design generated by the same user with EMO+IEC in which performance numbers predicted by the simulator software are similar, but the user's interaction has produced a design without potential operational flaws.

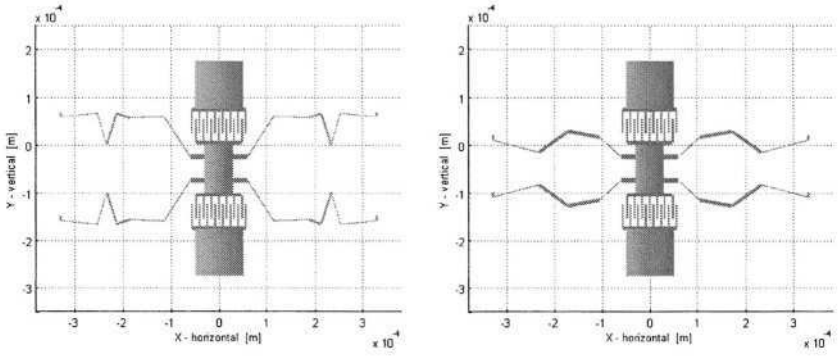


Fig. 6. (a) Left: MEMS resonator design produced by EMO-only, given a low score by a user due to potential stress concentrations in the legs. (b) Right: High scoring EMO+IEC design generated by the same user.

Unfortunately, with only six experts participating, we were not able to draw any statistically relevant conclusions about whether MEMS expertise has any impact on the success of the EMO+IEC framework. As we refine our work in future experiments, we hope to involve more MEMS experts in the testing.

5 Conclusions and Future Work

The impact of human interaction with the synthesis process has yielded some interesting results. We observed that humans tended to give low scores to designs that had particular features the user deemed undesirable. Over the course of several generation this caused families of related designs having those features to die off, thus blocking a road of evolution that the user believes is not worth pursuing. In our IEC test, with only a small population of twenty-seven this led to a rather homogenous population at the end of the evolution process, with most designs receiving a *'5'* score being variations on two or three dominant shapes. This phenomenon could be used as a variant to the EMO+IEC approach, where the human's role is focused explicitly on *killing off* unpromising design concepts.

We have fabricated MEMS resonator designs created with the EMO+IEC and EMO-only software to support these results. Devices were created in the MUMPS process and are currently being tested at the Berkeley Sensors and Actuator Sensor. A scanning electron micrograph of a design produced by an EMO+IEC process is shown in Fig. 7. The performance of these devices will be

used to improve our evolutionary synthesis approach as well as the performance of the SUGAR MEMS simulation software. The performance differences between the EMO and EMO+IEC generated devices can be used to further validate the results of the user study presented in this paper.

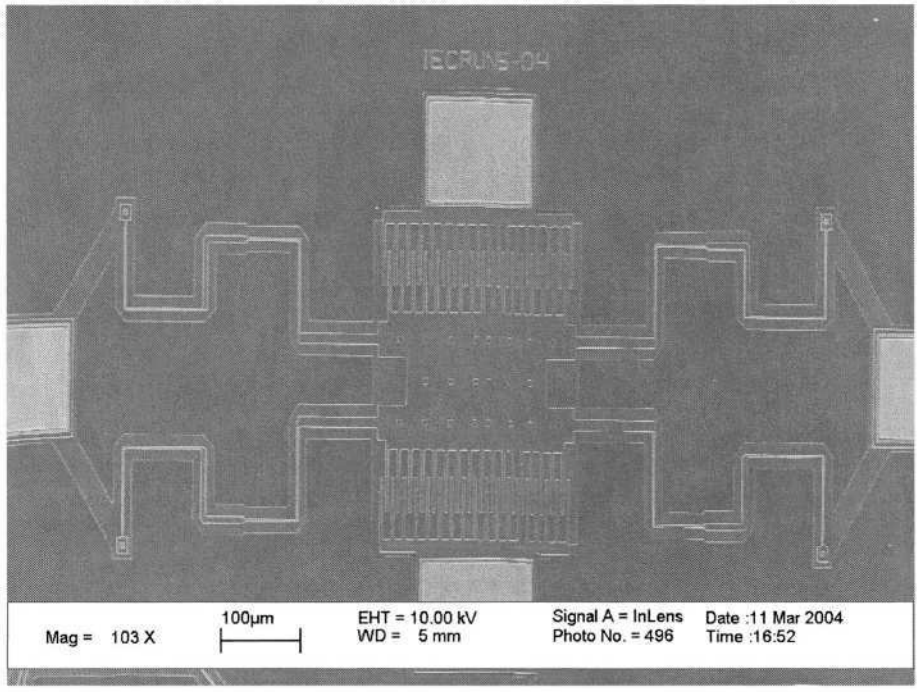


Fig. 7. SEM micrograph of symmetric MEMS resonators created with EMO+IEC.

The results of user testing on the MEMS resonator synthesis example show that using IEC to further evolve designs generated by EMO can produce better results than those using EMO alone. Use of EMO alone does not include critical factors that are difficult, if not impossible, to simulate. As these factors can have a major impact on the effectiveness of a design when fabricated, IEC is able to outperform by incorporating human domain knowledge to produce top ranking designs that are more suitable to the user's judgment of well performing designs.

Acknowledgements. This research was conducted (in part) through the National Science Foundation (NSF) East Asian Summer Institutes (EASI) Program, co-hosted by the Japanese Society for the Promotion of Science and NSF grant CCR-DES/CC-0306557. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF. The authors also wish to thank the

engineers and computer scientists who participated in the user testing from the Berkeley Expert Systems Technology Laboratory and the Berkeley Sensors and Actuator Center at UC Berkeley.

References

1. Clark, J.V., Bindel, D., Kao, W., Zhu, E., Kuo, A., Zhou, N., Nie, J., Demmel, J., Bai, Z., Govindjee, S., Pister, K. S. J., Gu, M., Agogino, A.M.: "Addressing the Needs of Complex MEMS Design," IEEE Int. MEMS Conf., Las Vegas, NV, USA, (Jan., 2002) 204–209.
2. Fan, Z., Seo, K., Hu, J., Rosenberg, R., and Goodman, E.: "System-Level Synthesis of MEMS via Genetic Programming and Bond Graphs," Genetic and Evolutionary Computing Conf. (GECCO2003), Chicago, Springer, Lecture Notes in Computer Science, (July, 2003) 2058–2071.
3. Kamalian, R., Zhou, N., and Agogino, A. M.: "A Comparison of MEMS Synthesis Techniques," 1st Pacific Rim Workshop on Transducers and Micro/Nano Technologies, Xiamen, China, (July, 2002) 239–242.
4. Li, H. and Antonsson, E. K.: "Evolutionary Techniques in MEMS Synthesis," 25th Biennial Mechanisms Conf., ASME Design Engineering Technical Conf., (DETC'98), Atlanta, GA, USA (1998) DETC98/MECH-5840.
5. Ma, L. and Antonsson, E. K.: "Robust Mask-Layout Synthesis for MEMS," Int. Conf. on Modeling and Simulation of Microsystems, Chapter 5: Optimization, Nanotech, **1**, (2001) 128–131.
6. Siegel, S. and Castellan, N.J.: *Nonparametric statistics for the behavioral sciences (2nd ed.),* London: McGraw-Hill (1988).
7. Takagi, H.: "Interactive Evolutionary Computation: Fusion of the Capacities of EC Optimization and Human Evaluation" *Proceedings of the IEEE,* **89**(9), (2001) 1275–1296.
8. Zhou, N., Zhu, B., Agogino, A. M., and Pister, K. S. J.: "Evolutionary Synthesis of MEMS MicroElectronicMechanical Systems Design," Intelligent Engineering System through Artificial Neural Networks Artificial Neural Networks in Engineering (ANNIE2001), **11,** ASME Press, (2001)197–202.
9. Zhou, N., Agogino, A. M., and Pister, K. S. J.: "Automated Design Synthesis for Micro-Electro-Mechanical Systems (MEMS)," CD-ROM Proc. of ASME Design Automation Conference, Montreal, Canada (Sept.-Oct., 2002).

Hybrid Genetic Algorithms for Multi-objective Optimisation of Water Distribution Networks

Edward Keedwell and Soon-Thiam Khu

Centre for Water Systems, School of Engineering and Computer Science and Mathematics, University of Exeter, North Park Road, Exeter, UK
{E.C.Keedwell, S.T.Khu}@exeter.ac.uk

Abstract. Genetic algorithms have been a standard technique for engineers optimising water distribution networks for some time. However in recent years there has been an increasing interest in multi-objective genetic algorithms that allow engineers a set of choices when implementing a solution. A choice of solutions is vital to help engineers understand the problem and in real world scenarios where budgets and requirements are flexible. This paper discusses the use of a local search procedure to speed up the convergence of a multi-objective algorithm and reports results on a real water distribution optimisation problems. This increase in efficiency is especially important in the water network optimisation field as the simulation of networks can be prohibitively expensive in computational terms.

1 Introduction

1.1 Water Distribution Network (WDN) Modelling

Water distribution network modelling is used for a variety of purposes such as: strategic and master planning; system design; energy management and daily operations; fire protection studies and emergency responses; water quality investigations; and many others. The computer modelling of water distribution networks continues apace in the water industry as computers become increasingly powerful, more complex systems are available to be modelled. Open source software such as EPANET (Rossman, 1993), which is the water distribution modelling software used in this paper, allows easy simulation of these complex systems. In recent years, computational methods such as non-linear programming, dynamic programming and search techniques have been used to optimise the design, operation and rehabilitation of these networks

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1042–1053, 2004.

1.2 Network Optimisation

The optimisation of WDN models can primarily be used for three purposes: the design of networks for new supply areas (design problems); modifying existing designs to meet new demands or other factors (rehabilitation problems) and modifying network parameters to ensure that they are accurate with respect to the real world (calibration problems). This paper is concerned with the problems of rehabilitation and design although the methods described herein can theoretically be used for any of these problems. A standard rehabilitation problem is concerned with taking an existing network and modifying its parameters or components to satisfy new constraints in the performance of the network. Therefore any process designed to optimise a rehabilitation problem must consider both the performance of the network (to be maximised) and the cost of the proposed solution (to be minimised). A design problem is similar except that instead of deciding on pipe sizes to meet some new criteria, the pipe sizes of the entire network must be decided from scratch. The results in later sections show that the proposed hybrid method is applicable to both rehabilitation and design problems.

1.3 Optimisation Algorithms

As described in 1.2, a rehabilitation problem is concerned with taking an existing network and modifying its components, however, in the actual WDN there exist a large number of components which act in combination. This combinatorial effect means that even a network with a modest number of components will contain a huge number of potential solutions and therefore that it is not possible to evaluate every possible solution within the timeframe of the project. Therefore there exist a number of algorithms which are designed to find near optimal solutions from the overall set in real-time. Notably genetic algorithms (GAs) have found considerable success in this domain (Savic and Walters, 1997) and are the subject of much of the rest of this paper.

Many engineering problems involve conflicting objectives, and the optimisation of WDNs is no exception. For instance in a rehabilitation problem, the optimal monetary solution is to leave the system as it is with no modifications, equally though, investing a huge amount of money in the system will give superior network performance. Clearly there is a trade-off here between investment and network performance. It is this trade-off, which, by using the same evolutionary processes as a standard GA, can be optimised. The goal of a multi-objective genetic algorithm (MOGA) therefore is to yield a set of solutions which represent the optimal trade-off between two or more conflicting aspects of a system. In this paper we describe the use of MOGAs on a standard rehabilitation problem.

1.4 Hybrid Algorithms

The above scenario shows that GAs, and in particular, MOGAs can be used to present realistic solutions to the problems of optimising WDN performance. However, these population-based techniques suffer from difficulties when applied on today's complex models. The problem exists because WDN modelling, especially for large networks and extended period simulation can incur large computation time. Typically, GAs and MOGAs use a population size of 50-200 and 100-10,000 generations. This can therefore lead to anywhere between 5,000 and 2 million model evaluations for an optimisation run. This is evidently not feasible even if each model simulation requires 1 minute to run on a standard machine – 1 x 2,000,000 = 2 million minutes = 3.8 years. Even for the shortest runs (5,000 evaluations), if the objective function takes 1 minute to evaluate, then running times of 3 ½ days can be expected. Thus, it is imperative that for simulation of large WDNs that the number of evalutions for any optimisation algorithm to be reduced to a manageable number.

The main work in this paper therefore is to investigate the possibility that a MOGA-Local Search hybrid algorithm can exploit the behaviour of the GA whilst reducing the overall number of evaluations required to obtain near-optimal solutions. In particular we explore a combined method of neighbourhood search and NSGA-II, the current best performer in WDN MOGA optimisation.

2 Method

2.1 Trade Off Surfaces for Multiple Objectives

In this paper, the version of MOGA used is the Non-Dominated Sorting Algorithm-II (NSGA-II) (Deb et al, 2001), which is currently considered to be one of the best algorithm for water distribution system optimisation (Farmani *et al,* 2003). The hybrid approach uses NSGA-II unchanged except for the fact that the local search element is run periodically within the NSGA-II execution and the results from the local search are returned to the population and the optimisation continues. Both the local search and all MOGAs (including NSGA-II) rely on the concept of domination. With single objective GAs, solutions can be compared with respect to their fitness and a selection made. However, MOGAs need to find a trade-off of solutions which is, ideally, both optimal and well-spread over the objective space. Figure 1 shows such a comparison between trade-off surfaces.

Domination

One solution is said to dominate another if it is better (what constitutes better is determined by the problem, maximisation or minimisation) or equal in every objective.

Non-Domination
One solution can be said to be non-dominated if there is no other solution in the solution set which dominates it.

Figure 1 shows the concepts of domination and non-domination. Each of the solution sets marked by a "circle" are said to be non-dominated as they have at least one better objective value. Head deficit is a measure of the ability of the system to meet the original pressure requirements of the WDN. The cross in Figure 1 however is dominated by "C" by virtue of the fact that it has higher cost and head deficit.

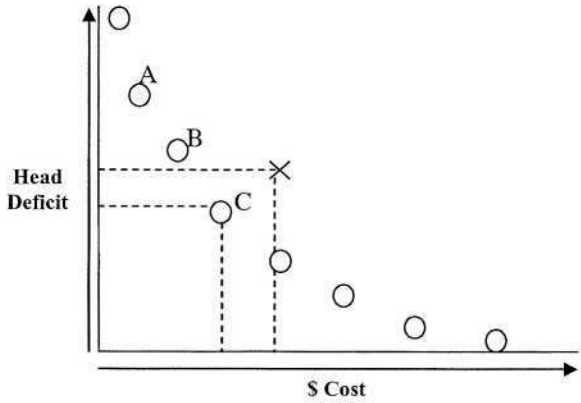


Fig. 1. Dominated and non-dominated solutions.

2.2 Multi-objective Area Measurement

The problem of comparing one or more pareto-curves in an experiment is non-trivial and a variety of methods have been put forward to evaluate these curves. One of the most obvious methods is to depict the curves on a chart and visually inspect which curve is closer to the optimum. However, whilst this can be effective, it is neither very scientific nor suitable for determining optimality during the course of an optimisation. The most suitable measure for this in our opinion is the hyper-volume or S-metric (Zitzler, 2000). Essentially this computes the area of the objective space that the pareto-curve covers, therefore meaning that the greater area covered, the more points that are dominated by the current curve. The method of computing the metric is shown in Figure 2 where each of the points has its area computed against the worst solution, W. The shaded area represents the area that is covered by the metric and as can be seen when minimizing both objectives for this method, the closer the solution set is to the optimum, the larger the area will be.

To succeed, the worst possible solution (W) has to be determined as a reference point to compute the bounds of the search space. In the experiments below we determine the worst hydraulic solution to be where every pipe diameter is minimal and the worst

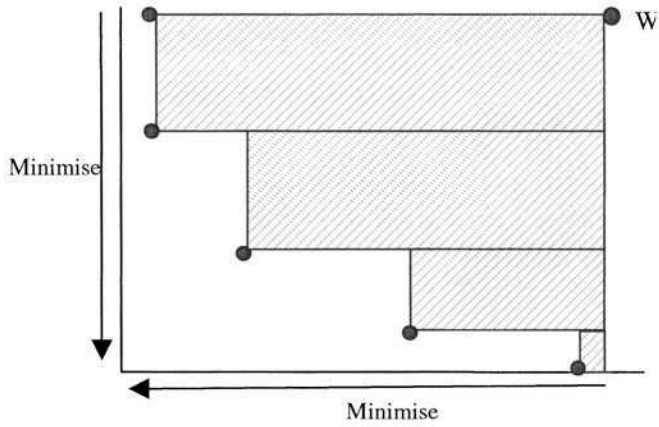


Fig. 2. Computation of the hyper-volume metric.

cost solution to be where every pipe diameter is maximal. Also, in the following experiments, we have normalised the calculations such that the maximum theoretical area is 1.0. For a precise definition of the S-metric, readers are referred to Zitzler (2000).

2.3 Hybrid Algorithm

The hybrid algorithm is used to drive the MOGA solutions towards the optimum during the optimisation to reduce the need for extended runs which require more model simulations. It uses a simplistic local search for a set number of iterations to find new, dominant solutions which are then entered back into the population of the MOGA. The standard local search is a neighbourhood search, it alters the diameters of two pipes at a time (one increment and one decrement) until it finds a solution which dominates the current solution. This is then selected and the process is repeated until some predetermined number of iterations has been completed.

This domination search ensures that only solutions which dominate the current best set are included in the search and therefore the insertion into the MOGA population. Once the solutions have been discovered, they are added back into the population, replacing the lowest ranked (and therefore most-optimal) solutions in the MOGA. Therefore the algorithm drives the MOGA to a more optimal solution set, but potentially at the price of some diversity in the population.

In following experimentation, we considered three separate local search components for the hybrid algorithm. The previous paragraph describes the operation of the simplest local search component that is used in the standard hybrid. However, because the problem is well-known a greater amount of knowledge can be built into the local search execution. This information takes the form of knowing which parts of the current solution are not meeting the requirements of the optimisation in one of the

objectives. In this example, we know that some nodes will have a pressure deficit and some a pressure surplus, therefore the advanced heuristics utilise this knowledge.

Each of the local search techniques must increment and decrement a pipe diameter simultaneously at each iteration. This is because to achieve a solution that dominates the current starting point, the solution must attain both lower cost and improved hydraulic performance. Simply incrementing a pipe diameter could yield better hydraulic performance, but cost will be increased, and by decrementing one, the reverse is true. The following hybrids use this combined increment/decrement behaviour:

1. **The standard hybrid** uses a simple neighbourhood search, taking the current solution and incrementing or decrementing the decision variables blindly. In this search, a neighbourhood of solutions is considered which are one increment and decrement away from the current solution. This hybrid moves along the chromosome incrementing and decrementing until such time as it finds a solution that dominates the current one whereupon it moves to this solution and restarts.
2. **The list heuristic** builds a list of those nodes that do not have their pressure requirements met (deficits) and those which have too much pressure (surpluses). These two lists are sorted to discover the largest deficits and surpluses and the algorithm proceeds down the list, incrementing the immediately upstream pipes of the deficit list and decrementing the immediately upstream pipes of the surplus list. If there are only deficits or surpluses in the network, then the algorithm uses opposite ends of the same list (i.e. it will attempt to increment the greatest deficit and decrement the smallest).
3. **The upstream heuristic** finds the nodes in the network with the greatest deficit and surplus. The search algorithm then proceeds upstream of each of these nodes, and repeats. Therefore after discovering the node with greatest deficit, the algorithm increments the upstream pipe adjacent to that node, and then the pipe further upstream, and then further upstream and so on, until a dominating solution is discovered.

The hybrids discussed here all attempt to discover a solution which dominates the starting solution. However, each of the techniques must be computationally efficient to improve GA performance so that they require less time than the standard GA. To attempt to achieve this, the heuristics include knowledge of the network nature of the problem to pinpoint first those areas where the largest improvements can be made. By modifying pipe diameters close to those nodes which are maximally different from optimal, it is hoped that both heuristics will find solutions more quickly. The upstream heuristic extends this concept by attempting to ensure that those least optimal nodes are satisfied no matter how far upstream the problem pipe is.

3 Experimentation

3.1 New York City Tunnels Problem

The work carried out in the following sections is based on the New York City tunnel (NYT) rehabilitation problem (Schaake and Lai, 1969). This is a well known problem which has been studied extensively in the literature (Bhave and Sonak, 1985; Murphy et al., 1993; Savic and Walters, 1997; Farmani, 2003 etc.) and involves replacing old trunk pipes with new pipes of larger diameters or putting in new trunk pipes alongside old ones within the network to meet new demands for the City of New York. Figure 3 shows the existing network configuration.

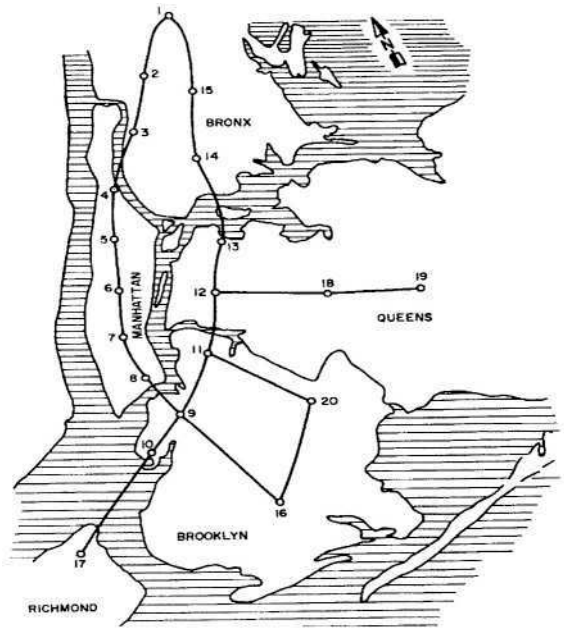


Fig. 3. Pictorial representation of the New York City Tunnels problem

There are a maximum of 21 new pipes to be laid, with the option of doing nothing. The new pipes vary in diameters, ranging between 60 – 204 inches. There is one reservoir which supplies the water to the network which contains 20 nodes which have various demands. A full enumeration of all possibilities would require 21^{16} ($1.43 * 10^{21}$) Epanet runs.

From an optimisation perspective, the objective of the NYT problem is to modify the rehabilitated pipe diameters to meet the demands at the nodes. The current optimal solution for this is 38.64 million dollars and no pressure deficit although this can vary slightly depending on the modelling software and parameters used.

3.2 Parameters

During experimentation with the algorithm we have discovered a number of parameters that gives the algorithm flexibility to discover the near- optimal sets. They include the following:

- *Local search frequency* – determines the number of times the local search is used throughout the optimisation.
- *Local search iterations* – determines the computational effort given to the local search when it is used
- *Replacement strategy* – determines which individuals in the GA population to replace, the best? Or the worst?
- *MOGA Parameters* – Those that can effect hybrid optimisation, such as population size.

Generally speaking we have found that low frequency, low iteration searches have produced the best results in conjunction with an optimal-replacement strategy which replaces the best-ranked solutions in the current GA population.

3.3 Experimental Procedure

Several experiments have been run on the New York Tunnels optimisation problem to determine the efficacy of the hybrid techniques described above.

Experiment 1 shows graphically the S-metric performance of each of the algorithms on a single optimisation run.

Experiment 2 evaluates the performance of each of the hybrids over 22 different random seeds and discovers how many model evaluations each algorithm requires to achieve a specific S-metric value.

These experiments are designed to compare the use of a hybrid optimisation procedure with NSGA-II, which is one of the best current multi-objective genetic algorithms.

4 Results

4.1 Experiment 1

In this experiment, each of the algorithms was run with the same random seed, and the S-Metric values at various points in the optimisation recorded. Each algorithm was run with a population size of 100, mutation and crossover rates of 0.9. The local search intervened once (at 50 generations) for 4 iterations. The resulting graph (Figure 4) gives an impression of the speed of convergence for all of the algorithms.

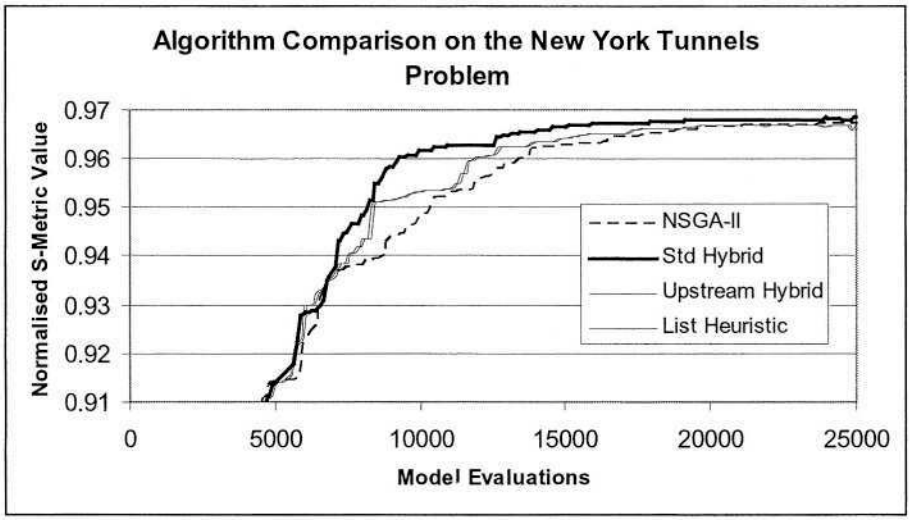


Fig. 4. Graph depicting the area under the Pareto curve against the computational cost, measured in model evaluations, for all algorithms

From Figure 4, it can be seen that the standard hybrid method outperforms all the other methods by some considerable margin in this case. It is also interesting to note that each of the hybrid methods outperforms the standard multi-objective GA on this problem but that the list and upstream heuristics are very close in performance. However, as this is only for one case the next experiment uses multiple trials to determine the performance of each algorithm.

4.2 Experiment 2

In this second experiment, NSGA-II and each of the hybrid algorithms was run with the same set of 22 different random seeds (12345 & 0-20). The number of model evaluations required to reach a fixed S-metric value of 0.95 is recorded. Table 1 shows the average, minimum, maximum and standard deviation of the number of model evaluations required to achieve this limit on the New York Tunnels problem. Each algorithm was run with 22 different random seeds and the local search run for 4 iterations. The table also shows these statistics for NSGA-II alone to achieve this area coverage.

Each of the hybrid techniques achieves some computational saving in comparison with the standard NSGA-II runs. The averages reveal a maximum of 7% model evaluation saving over the 22 runs when using the standard hybrid. The lowest performance of each algorithm is very similar and this suggests that the random seed is very important in the optimal execution of NSGA-II on this problem.

Table 1. Algorithm model evaluation comparison to 0.95 S-metric value

Algorithm	Mean	Min	Max	Std Dev
Standard Hybrid	6982	5845	8255	604.77
List Heuristic	7326.82	5900	8501	743.74
Upstream Heuristic	7326.82	5900	8501	743.74
NSGA-II	7495.46	5900	10000	911.03

Two further runs of this experiment with higher attainment targets of area coverage shows the extra computation required to achieve a marginally more optimal solution. Table 2 shows the extra model evaluations required to progress from 0.95 to 0.955 and 0.96 for each of the algorithms.

Table 2. Additional model evaluations required to achieve higher S-Metric values

Algorithm	0.955	0.96
Standard Hybrid	836.36	1981.82
List Heuristic	1204.55	2622.73
Upstream Heuristic	1204.55	2622.73
NSGA-II	1054.54	2609.10

This experiment shows again that by augmenting local search, better results can be achieved over the standard NSGA-II. In the best case (the standard hybrid), an S-metric value of 0.96 was achieved with an average of over 11% less model evaluations than the standard GA for the 22 random seeds. This indicates that the benefits of the local search procedure are still being felt later on in the optimisation.

5 Conclusion

Evolutionary multi-objective optimisation is currently one of the most useful tools available for the optimisation of water distribution systems because it allows the decision maker a choice of options. However, as with other generational evolutionary algorithms, the number of model evaluations incurred by such a system is so great that often they are difficult to apply to real-world systems. The hybrid approaches detailed in this paper show that local search can be used as an effective method of speeding up the search for a Pareto front over the current state-of-the-art multi-objective optimiser, NSGA-II. Both experiments show that the standard hybrid performed better than the heuristic approaches however the relatively poor performance of the heuristics in this application can potentially be explained by a single factor.

In this WDN, the largest head deficits tend to occur at the farthest end of the network from the reservoir, and therefore making a change locally to these nodes relies heavily on the construction of the rest of the network. Put simply, if the pressures upstream of the node are too low, increasing the local pipe diameters is going to have a minimal effect on the head at that node whilst significantly increasing costs. Each of the heuristics is designed to target these high deficit nodes first and therefore wastes a number of model evaluations searching these options that are only of minor hydraulic benefit whilst incurring high cost, and therefore are non-dominant.

This paper therefore shows that each of the hybrid optimisation procedures described here can improve on the performance of the standard NSGA-II. This algorithm is currently one of the best for multi-objective optimisation and therefore this represents a significant result. However, these results also highlight the need for careful selection of heuristics in local search as intuitive heuristics to the problem did not perform as well as expected.

In summary, the domain of water distribution network optimisation is notoriously computationally expensive and the computational savings made by a hybrid technique such as this could mean that a wider variety of complex real-world systems can be optimised using multi-objective evolutionary techniques.

References

Bhave P. R. and Sonak V. V. (1992). A critical study of the linear programming gradient method of optimal design of water supply networks. Water Resour. Res., 28(6),_1577 – 1584.

Deb, K. and Goel, T. (2001) " Controlled elitist non-dominated sorting genetic algorithms for better convergence ," Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization (EMO-2001), 7--9 March. (Zurich, Switzerland), (pp. 67--81).

Deb, K., Agrawal, S., Pratap, A., Meyarivan, T. (2000). A Fast Elitist Non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. Proceedings of the Parallel Problem Solving from Nature VI Conference , 16-20 September. (Paris, France), (pp. 849--858).

Farmani, R., Savic D. and Walters G., (2003) "Multi-objective optimisation of water systems: A comparative study" Pumps Electromagnetic Devices and Systems Applied to Urban Water Management (Cabrera, Cabrera Jr. Eds) AA Balkema Publishers pp 247-255

Goldberg, D.E., (1989) *Genetic Algorithms in Search, Optimisation and Machine Learning.* Addison Wesley Publishing Company Inc.

Keedwell, E.C. and Khu, S.T. (2003) "More choices in water system design through hybrid optimisation" Advances in Water Supply Management: Proceedings of the International Conference on Computing and Control in the Water Industry pp 257-265

Murphy L. J., Simpson A. R. and Dandy G. C. (1993). Pipe network optimisation using an improved genetic algorithm. *Res. Rep. No. R109. Dept. of Civil and Envir. Engrg.,* Univ. of Adelaide, Australia.

Rossman L., (1993). "EPANET", U. S. Environment Protection Agency.

Savic D. and Walters G., (1997) "Genetic Algorithms for Least-Cost Design of Water Distribution Networks" J. Water Resources Planning and Mgt., 123(2), 67-75

Schaake J. and Lai D. (1969). Linear programming and dynamic programming applications to water distribution network design. *Rep. 116, Dept. of Civil Engrg., Massachusetts Inst. of Technol.,* Cambridge, Mass.

Vairavamoorthy K. and Ali M., (2000). Optimal design of water distribution systems using genetic algorithms. Computer Aided Civil and Infrastructure Engineering, *15,* 374-382.

Zitzler E., Deb K. and Thiele L., (2000). Comparison of multiobjective evolutionary algorithms: Empirical results. Evolutionary Computation, *8(2),* 173-195.

A Hybrid Genetic Approach for Circuit Bipartitioning

Jong-Pil Kim[1], Yong-Hyuk Kim[2], and Byung-Ro Moon[2]

[1] Electronics and Telecommunications Research Institute
161 Gajeong-dong, Yuseong-gu, Daejeon, 305-350 Korea
kimjp@etri.re.kr
[2] School of Computer Science & Engineering, Seoul National University
Shillim-dong, Kwanak-gu, Seoul, 151-742 Korea
{yhdfly, moon}@soar.snu.ac.kr

Abstract. We propose a hybrid genetic algorithm for partitioning a VLSI circuit graph into two disjoint graphs of minimum cut size. The algorithm includes a local optimization heuristic which is a modification of Fiduccia-Matheyses algorithm. Using well-known benchmarks (including ACM/SIGDA benchmarks), the combination of genetic algorithm and the local heuristic performed better than hMetis, a representative circuit partitioning algorithm.

1 Introduction

Hypergraph partitioning is an important problem and has many applications including design automation of VLSI chips and multi-chip systems. A hypergraph *H(V, E)* consists of a set of nodes *V* and a set of nets *E*. Each *net* $e \in E$ is a subset of two or more nodes in *V*. The nodes associated with a net are called *pins*. A bisection of *H* is dividing the set *V* into two disjoint subsets V_1 and V_2. The cut size $c(V_1, V_2)$ is defined as

$$c(V_1, V_2) = |\{ e \in E \mid e \cap V_1 \neq \phi \text{ and } e \cap V_2 \neq \phi \}|.$$

In other words, the cut size is the number of nets whose end points are in different subsets. The *min-cut bisection problem* minimizes the $c(V_1, V_2)$ subject to $|V_1| = |V_2|$. Since it is NP-hard problem [16], heuristic algorithms, which yield approximate solutions in acceptable times, are generally used. These include iterative improvement methods [12] [13] [14] [20] [23], meta-heuristic methods such as simulated annealing (SA) [18] [22], large-step Markov chain (LSMC) [15] [25], tabu search (TS) [3] [11] [27], and genetic algorithm (GA) [4] [7] [8] [17] [21] [24] [28] [30]. An extensive survey of hypergraph partitioning appeared in [2].

The Kernighan-Lin algorithm (KL) [20] is a representative iterative improvement algorithm for general graphs. KL starts with a random initial two subsets and proceeds by iteratively swapping nodes. Subsequently, Schweikert and Kernighan [29] modified the algorithm for hypergraphs. Fiduccia and Mattheyses algorithm (FM) [14] is a variation of KL, which reduces the time complexity

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1054–1064, 2004.

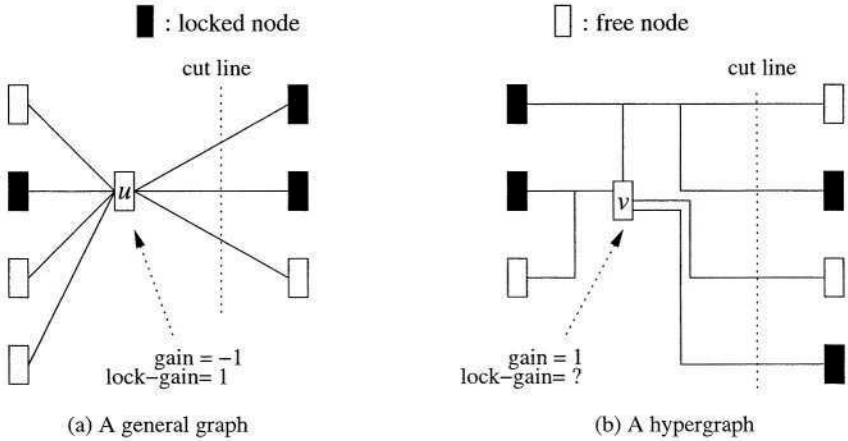


Fig. 1. Gain vs. lock gain

of the algorithm to $O(|P|)$ with effective data structures, where $|P|$ is the number of pins in a hypergraph. Krishnamurthy [23] enhanced FM by introducing the concept of level gains and obtained better results in small graphs. Dutt and Deng proposed PROP [12] which uses a probabilistic gain computation, and CLIP/CDIP [13] which uses a cluster-removal method.

Lock gain is a new measure which showed improvement over the traditional gain for general graphs [21]. In this paper, we adapt the lock gain for hypergraphs within the framework of FM [14]. We combine the lock-gain-based hypergraph partitioning heuristic with a steady-state genetic algorithm.

The remainder of this paper is organized as follows. Section 2 summarizes the FM algorithm and explains the lock gain. In Section 3, we describe the local optimization heuristic for min-cut bisection problem. Details of the genetic algorithm are provided in Section 4. We show the experimental results in Section 5 and summarize the study in Section 6.

2 Preliminaries

2.1 Fiduccia-Matheyses Algorithm

The FM algorithm [14] starts with two random initial subsets. It moves a node at a time from one subset to the other in an attempt to minimize the cut size. Each node v has a *gain* $g(v)$ which represents the cut size reduction by moving v to the opposite subset:

$$g(v) = |\{e \in I(v)\}| \; - \; |\{e' \in C(v)\}|$$

where $C(v)$ is the set of nets containing v in which all pins lie in the subset to which v belongs; $I(v)$ is the set of nets such that all the other pins except v

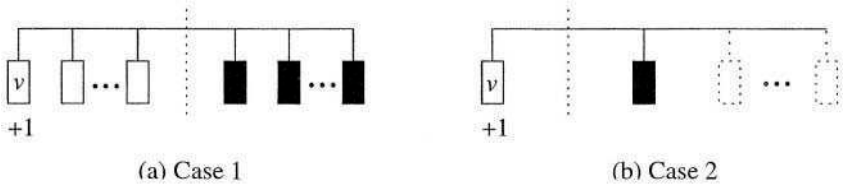


Fig. 2. Positive lock gain

belong to the opposite subset. To differentiate the gain with the lock gain of the next section, we sometimes use the term "general gain."

The node to move is chosen on the basis of its gain and the balance criterion. If we do not consider the balance criterion, all nodes would eventually migrate to one subset except a lowest-degreed node. After a node is moved, it is locked to prevent further movements. Only unlocked nodes are allowed to move. A pass stops when all nodes are locked. Then we undo the sequence of movements beyond the point that maximizes the total gain. All the nodes are now marked unlocked again and another pass starts. This process is repeated until no more improvement is obtained.

2.2 Lock Gain

The general gain of a node only reflects the local information of a node. Hence, it is hard to predict the future state of the node from the gain. FM improves an initial solution through short-sighted moves based on the gain. Kim and Moon [21] introduced lock gain as a primary measure for choosing the nodes to move. It uses the history of search more efficiently. Lock gain showed excellent performance for general graphs.

In general graphs, the lock gain $l(v)$ of a node v is defined to be the gain of the node v only with respect to the locked nodes. Let node v be in the subset V_1 without loss of generality. Formally,

$$\begin{aligned} l(v) \;=\; & |\{w \mid (v,w) \in E, \text{ and node } w \in L(V_2)\}| \\ & - |\{w' \mid (v,w') \in E, \text{ and node } w' \in L(V_1)\}| \end{aligned} \qquad (1)$$

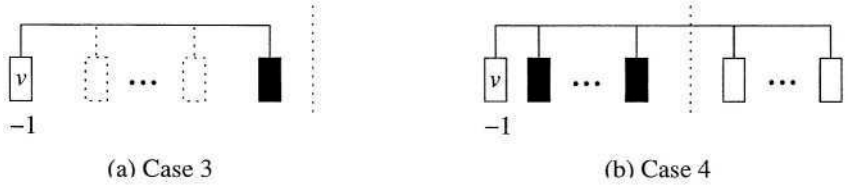
where $L(V_i)$ is the set of locked nodes in the subset V_i.

An example is given in Figure 1(a). In the figure, the gain of the node u is – 1. But when only locked nodes are considered, the lock gain of the node u becomes 1.

3 Lock Gain in Hypergraphs

3.1 Lock Gain in Hypergraphs

To apply the lock gain to the hypergraph bisection, considerable modification is necessary for the lock gain calculation method. In general graphs, nets cannot



(a) Case 3 (b) Case 4

Fig. 3. Negative lock gain

have degrees greater than two. But in hypergraphs, most nets have degrees greater than two. A net may contain both locked nodes and unlocked nodes. The above term (1) does not work for this case. For example, in Figure 1(b), the gain of the node v is 1; but the lock gain of the node cannot be calculated in the same method as in Figure 1(a).

We propose a new lock gain calculation method for the hypergraph bisection. Let's define $l_e(v)$ to be the lock gain of a node v due to the net $e \in E$. $l_e(v)$ is obtained as the following.

We assume that the node v is on the left side without loss of generality. L, R, and L', R' are defined to be the numbers of nodes on the left side and on the right side and the numbers of locked nodes on the left side and on the right side, respectively.

i) Positive Lock Gain
   - Case 1 (e.g., Figure 2(a)) :

$$L > 0, \quad L' = 0, \quad R = R' > 0.$$

     If all nodes on the right side are locked and there is no locked node on the left side, then $l_e(v) = 1$.
   - Case 2 (e.g., Figure 2(b)) :

$$L = 1, \quad L' = 0, \quad R \geq R' > 0.$$

     If there exist locked nodes on the right side and there is one free node on the left side, then $l_e(v) = 1$.

ii) Negative Lock Gain
   - Case 3 (e.g., Figure 3(a)) :

$$L > L' > 0, \quad R = R' = 0.$$

     If all nodes are on the left side and at least one node is locked, then $l_e(v) = -1$.
   - Case 4 (e.g., Figure 3(b)) :

$$L - L' = 1, \quad R > 0, \quad R' = 0.$$

     If all nodes on the left side except v are locked and there is no locked node on the right side, then $l_e(v) = -1$.

```
LFM (G, V1, V2)
    // Vi: a given initial subset, V1 ∪ V2 = V and V1 ∩ V2 = φ
1.  do {
2.      compute g(v) for each v ∈ V;
3.      set l(v) to 0 for each v ∈ V;
4.      Q = φ;
5.      for i = 1 to n {
6.          if (V1 > V2)
7.              choose vi ∈ V1 − Q such that l(v) is maximal over all choices of v ∈ V1;
8.          else
9.              choose vi ∈ V2 − Q such that l(v) is maximal over all choices of v ∈ V2;
10.         Q = Q ∪ {vi};
11.         for each v ∈ V − Q adjacent to vi
12.             adjust l(v) and g(v) if affected by moving vi;
13.     }
14.     choose k ∈ {1, ..., n} to maximize Σ_{i=1}^{k} g(vi);
15.     move nodes in the subsets {v1, ..., vk} to their opposite sides;
16. } until (there is no improvement)
```

Fig. 4. LFM algorithm

iii) Zero Lock Gain
In all the other cases, $l_e(v) = 0$.

Then, the lock gain $l(v)$ of the node v is defined to be

$$l(v) = \sum_{e \in N(v)} l_e(v)$$

where $N(v)$ is a set of nets to which the node v is connected. If all nodes are locked, $l(v)$ is equal to $g(v)$.

Our lock-gain based FM algorithm is given in Figure 4. In the algorithm, lines 2 through 4 compute the gains, set lock gains to 0 for all nodes, and initialize Q (the set of locked nodes). Lines 6 through 12 choose a node, move and lock it, and adjust general gains and lock gains which are affected by the movement. In lines 14 and 15, we undo the sequence of movements beyond the point that maximizes the total gain (i.e., minimizes the cut size). We call this algorithm LFM.

3.2 Implementation

We use the bucket data structure to maintain lock gain lists. This data structure allows constant-time selection of the most attractive node and fast gain update after each move [14]. Since the lock gain is an integer in the range $[-D_{max}, D_{max}]$ where D_{max} is the maximum node degree in the graph, the bucket structure maintains the lock gain efficiently. When a tie occurs in max lock gain, general gains are used for the tie-breaking. For the efficient implementation of this tie-breaking strategy, we enlarge the number of buckets.

Kim and Moon [21] used $(2D_{max}+1)^2$ buckets for tie-breaking. In this paper, we use $(2\lceil \frac{D_{max}}{2} \rceil + 1)^2$ buckets for speed up. In an examination, we observed that

Table 1. The Comparison of Two Buckets (Using LFM)

Circuits	$(2\lceil \frac{D_{max}}{2} \rceil + 1)^2$		$(2D_{max} + 1)^2$	
	Ave[1]	CPU[2]	Ave[1]	CPU[2]
Test03	80.40	0.046	81.36	0.072
Test04	77.03	0.060	78.23	0.108
Test05	107.20	0.117	106.95	0.169
Prim1	66.17	0.009	66.17	0.009

1. The average cut size of 1,000 runs
2. CPU seconds on Pentium III 1 GHz

the number of nodes with a degree greater than $\lceil \frac{D_{max}}{2} \rceil$ is less than 20% of the entire nodes. We set the range of the gain values to $[-\lceil \frac{D_{max}}{2} \rceil, \lceil \frac{D_{max}}{2} \rceil]$. If a gain value is out of the range, we set it to one of the two bounds.

A node v with lock gain $l(v)$ and gain $g(v)$ is stored at the bucket indexed by the value $R(l(v))(2\lceil \frac{D_{max}}{2} \rceil + 1) + R(g(v))$ where

$$R(k) = \begin{cases} -\lceil \frac{D_{max}}{2} \rceil, & \text{if } k < -\lceil \frac{D_{max}}{2} \rceil \\ k, & \text{if } -\lceil \frac{D_{max}}{2} \rceil \leq k \leq \lceil \frac{D_{max}}{2} \rceil \\ \lceil \frac{D_{max}}{2} \rceil, & \text{if } k > \lceil \frac{D_{max}}{2} \rceil \ . \end{cases}$$

Table 1 compares the two strategies with $(2\lceil \frac{D_{max}}{2} \rceil + 1)^2$ and $(2D_{max} + 1)^2$ buckets. One can observe that the former is faster than the latter without notable impact on performance.

4 Genetic Algorithm

We devised a hybrid steady-state genetic algorithm for the problem. A hybrid steady-state genetic algorithm is often called a memetic algorithm [26]. Figure 5 shows the template of the GA.

- Encoding:
  A chromosome is defined to be an n-tuple $\langle c_1, c_2, \ldots, c_n \rangle$ where $c_i \in \{0, 1\}$ for $i = 1, 2, \ldots, n$. If the i^{th} node belongs to the left side, the value of the i^{th} gene is 0; otherwise, it is 1.

- Initialization:
  p chromosomes are created at random. Each chromosome undergoes a balancing process. We set the population size p to be 50.

- Selection:
  The roulette-wheel proportional selection scheme is used. The probability that the best chromosome is chosen was set four times higher than the probability that the worst chromosome is chosen.

```
create initial population of fixed size;
do {
    choose parent1 and parent2 from population;
    offspring ← crossover (parent1, parent2);
    local-improvement(offspring);        // LFM algorithm
    replace (population, offspring);
} until (stopping condition);
return the best member of the population;
```

Fig. 5. A hybrid genetic algorithm for hypergraph partitioning

- Crossover and Mutation:
  We used five-point crossover. After crossover, chromosomes are usually not balanced. We start at a random point on the chromosome and adjust the gene values to the right until the balance is satisfied. This has some mutation effect, so we do not add any explicit mutation.

- Replacement:
  We combine preselection [6] and Genitor-style replacement [31]. If it is superior to the closer parent in Hamming distance, the offspring replaces the closer parent, if not, the inferior parent is replaced if the offspring is better. If not again, the worst in the population is replaced.

- Stopping condition:
  If 70 percent of the population converges with the same cut size as the best solution, we stop the GA.

5 Experimental Results

5.1 Test Set and Test Environment

Before showing the experimental results, we introduce the benchmarks used in this experiment. We tested the proposed algorithm on 9 benchmark circuit graphs, including seven ACM/SIGDA benchmarks [1]. Table 2 shows the numbers of nodes, nets, and pins for each circuit graph. All cut size results are from the strict 50:50% balance criterion. Although some flexibility is allowed in the balancing in real-world circuit partitioning, this bisection model has a strong indication to the performance in other partitioning models.

We first examine the performance of the suggested local optimization heuristic (LFM) against the original FM and a well-known partitioner hMetis [19]. Metis (hMetis) is a recent representative partitioning algorithm which has been a standard for comparison in a number of papers [5] [9] [10]. Then we show the

Table 2. Specification of Circuit Graphs

Circuits	#nodes	#nets	#pins
Test02	1663	1721	6134
Test03	1607	1618	5807
Test04	1515	1658	5975
Test05	2595	2751	10076
Test06	1752	1674	6638
Prim1	833	902	2908
Prim2	3014	3029	11219
19ks	2844	3282	10547
Industry2	12142	12949	47193

experimental results of the hybrid GA. Since the hybrid GA uses the local optimization heuristic as an engine, it is obvious that the hybrid GA would perform better than the local optimization heuristic. We thus examine the effectiveness of the genetic search by comparison with the multi-start local optimization with comparable time budgets.

C language was used on a Pentium III 1 GHz PC with Linux operating system. It was compiled with GNU's *gcc* compiler.

5.2 Experimental Results

We first examine the performance of the local optimization heuristic (LFM) in Table 3. FM, hMetis, and LFM are compared. The statistics of the three algorithms are from 1,000 independent runs. So the average results are fairly stable. The bold-faced numbers indicate the best result among the three algorithms. LFM significantly improved the performance of the original FM at the cost of a bit more CPU time. However, LFM was not comparable to hMetis as its own.

Table 4 shows the performance of the suggested GA. The genetic algorithm (GA) significantly improved the performance of LFM. Because the GA took roughly 200 times more than a single run of LFM, it is not clear how critical the genetic search is to the performance improvement. Thus, we compared the GA with the multi-start version (LFM200) that runs LFM on 200 random initial solutions and returns the best out of them. For the same reason, hMetis200, that is a multi-start version of hMetis with 200 runs, was compared. On the average, the proposed GA performed best in six graphs among nine. The difference between LFM200 and GA is evidently owing to the genetic process.

As mentioned, hMetis outperformed LFM. It constructs a multi-level hierarchy and performs complex declustering and reclustering process called V-cycle. We want to emphasize that LFM, which simply changed the measure for finding nodes to move, obtained high quality performances and the genetic algorithm boosted up the LFM's quality so that it performed better than hMetis.

Table 3. Bipartition Cut Sizes of FM, hMetis, and LFM

Circuits	FM			LFM			hMetis		
	Ave[1]	CPU[2]	Best[3]	Ave[1]	CPU[2]	Best[3]	Ave[1]	CPU[2]	Best[3]
Test02	172.88	0.016	114	109.03	0.059	91	**101.87**	0.061	88
Test03	118.72	0.078	64	80.40	0.046	58	**63.94**	0.059	59
Test04	140.22	0.014	70	77.63	0.060	51	**58.81**	0.050	54
Test05	181.47	0.039	98	107.20	0.117	75	**80.24**	0.096	71
Test06	92.09	0.017	67	75.57	0.026	63	**68.50**	0.061	64
Prim1	80.17	0.007	54	66.17	0.009	54	**56.79**	0.032	53
Prim2	296.96	0.068	176	230.46	0.115	147	**197.00**	0.172	148
19ks	190.88	0.059	136	154.30	0.089	112	**127.52**	0.130	111
Industry2	839.25	0.475	342	311.84	0.757	199	**228.21**	1.159	180

1. The average cut size of 1,000 runs
2. CPU seconds on Pentium III 1 GHz
3. The best cut size of 1,000 runs

Table 4. Bipartition Cut Sizes of LFM200, hMetis200, and GA

Circuits	LFM200			GA			hMetis200		
	Ave[1]	CPU[2]	Best[3]	Ave[4]	CPU[2]	Best[5]	Ave[6]	CPU[2]	Best[7]
Test02	92.08	11.58	89	**88.16**	17.20	88	89.81	12.12	88
Test03	58.93	9.07	58	**58.00**	5.59	58	59.90	11.74	58
Test04	52.41	11.98	51	**51.15**	8.90	51	54.77	10.02	54
Test05	77.37	20.83	72	72.05	27.85	71	**71.33**	19.22	71
Test06	63.53	5.10	63	**63.12**	4.89	63	64.05	12.30	64
Prim1	53.82	1.86	53	53.87	1.14	53	**53.00**	6.33	53
Prim2	152.76	24.78	146	**149.02**	19.45	146	177.76	34.39	146
19ks	116.50	19.72	112	**110.22**	21.44	110	110.81	25.89	110
Industry2	210.48	130.50	195	186.97	211.25	183	**181.15**	233.35	180

1. The average cut size of 100 runs, each of which is the best of 200 runs of LFM
2. CPU seconds on Pentium III 1 GHz
3. The best cut size of 20,000 runs
4. The average cut size of 100 runs
5. The best cut size of 100 runs
6. The average cut size of 100 runs, each of which is the best of 200 runs of hMetis
7. The best cut size of 20,000 runs

6 Conclusions

We proposed a hybrid genetic algorithm for the hypergraph min-cut bisection problem. In order to design a good hybrid GA, we devised a new local optimization heuristic. The suggested local heuristic is a variation of FM algorithm that uses lock gain for the choice of moving nodes.

The hybrid genetic algorithm with the new local search heuristic synergetically achieved good performance. The experimental results showed the effectiveness of the suggested genetic algorithm.

There are stochastic methods such as tabu search [27] [11] [3] and large-step Markov chain [25] [15] that are known to have effective search capabilities. Combination of our suggested local heuristic and other stochastic methods is considered to be a promising topic for further studies.

Acknowledgments. This study was supported by Brain Korea 21 Project. The ICT at Seoul National University provided research facilities for this study.

References

1. Benchmark. http://vlsicad.cs.ucla.edu/~cheese/benchmarks.html.
2. C. J. Alpert and A. B. Kahng. Recent directions in netlist partitioning: A survey. *Integration: the VLSI Journal,* 19(1-2):1–81, 1995.
3. R. Battiti and A. Bertossi. Greedy, prohibition, and reative heuristics for graph partitioning. *IEEE Trans. on Computers,* 48(4):361–385, 1999.
4. T. N. Bui and B. R. Moon. Genetic algorithm and graph partitioning. *IEEE Trans. on Computers,* 45(7):841–855, 1996.
5. A. E. Caldwell, A. B. Kahng, and I. L. Markov. Improved algorithms for hypergraph bipartitioning. In *Asia and South Pacific Design Automation Conference,* pages 661–666, 2000.
6. D. Cavicchio. *Adaptive Search Using Simulated Evolution.* PhD thesis, University of Michigan, Ann Arbor, MI, 1970.
7. J. P. Cohoon, W. N. Martin, and D. S. Richards. A multi-population genetic algorithm for solving the k-part on hyper-cubes. In *Fourth International Conference on Genetic Algorithms,* pages 244–248, 1991.
8. R. Collins and D. Jefferson. Selection in massively parallel genetic algorithms. In *Fourth International Conference on Genetic Algorithms,* pages 249–256, 1991.
9. J. Cong and S. K. Lim. Edge separability based circuit clustering with application to circuit partitioning. In *Asia and South Pacific Design Automation Conference,* pages 429–434, 2000.
10. J. Cong, S. K. Lim, and C. Wu. Performance driven multi-level and multiway partitioning with retiming. In *ACM/IEEE-CAS/EDAC Design Automation Conference,* pages 274–279, 2000.
11. M. Dell'Amico and F. Maffioli. A new tabu search approach to the 0-1 equicut problem. In *Meta-Heuristics 1995: The State of the Art,* pages 361–377. Kluwer Academic Publishers, 1996.
12. S. Dutt and W. Deng. A probability-based approach to VLSI circuit partitioning. In *Proc. Design Automation Conference,* pages 100–105, 1996.
13. S. Dutt and W. Deng. VLSI circuit partitioning by cluster-removal using iterative improvement techniques. In *Proc. International Conference on Computer-Aided Design,* pages 194–200, 1996.
14. C. Fiduccia and R. Mattheyses. A linear time heuristics for improving network partitions. In *19th IEEE/ACM Design Automation Conference,* pages 175–181, 1982.
15. A. S. Fukunaga, J. H. Huang, and A. B. Kahng. On clustered kick moves for iterated-descent netlist partitioning. In *IEEE International Symposium on Circuits and Systems,* volume 4, pages 496–499, 1996.
16. M. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, 1979.
17. M. Harpal, M. Kishan, M. Chilukuri, and R. Sanjay. Genetic algorithms for graph partitioning and incremental graph partitioning. In *IEEE Proceedings of the Supercomputing,* pages 449–457, 1994.
18. D. S. Johnson, C. Aragon, L. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation. *Operations Research,* 37:865–892, 1989.

19. G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: Application in VLSI domain. In *Proc. Design Automation Conference,* pages 526–529, 1997.
20. B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal,* 49:291–307, 1970.
21. Y. H. Kim and B. R. Moon. Lock-gain based graph partitioning. *Journal of Heuristics,* 10(1):37–57, 2004.
22. S. Kirkpatrick, Gelatt C. D. Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science,* 220(4598):671–680, May 1983.
23. B. Krishnamurthy. An improved min-cut algorithm for partitioning VLSI networks. *IEEE Trans. on Computers,* 33:438–446, 1984.
24. G. Laszewski. Intelligent structural operators for the k-way graph partitioning problem. In *Fourth International Conference on Genetic Algorithms,* pages 45–52, 1991.
25. O. Martin, S. W. Otto, and E. W. Felten. Large-step markov chains for the traveling salesman problem. *Complex Systems,* 5:299–326, 1991.
26. Pablo Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical Report C3P 826, Caltech Concurrent Computation Program, 1989.
27. E. Rolland, H. Pirkul, and F. Glover. A tabu search for graph partitioning. *Annals of Operations Research,* 63, 1996.
28. Y. Saab and V. Rao. Stochastic evolution: A fast effective heuristic for some genetic layout problems. In *27th IEEE/ACM Design Automation Conference,* pages 26–31, 1990.
29. D. G. Schweikert and B. W. Kernighan. A proper model for the partitioning of electrical circuits. In *Proc. 9th Design Automation Workshop,* pages 57–62, 1972.
30. A. G. Steenbeek, E. Marchiori, and A. E. Eiben. Finding balanced graph bipartitions using a hybrid genetic algorithm. In *IEEE Conference on Evolutionary Computation,* pages 90–95, 1998.
31. D. Whitley and J. Kauth. Genitor: A different genetic algorithm. In *Proceedings of Rocky Mountain Conference on Artificial Intelligence,* pages 118–130, 1988.

Lagrange Multiplier Method for Multi-campaign Assignment Problem

Yong-Hyuk Kim and Byung-Ro Moon

School of Computer Science & Engineering, Seoul National University
Shillim-dong, Kwanak-gu, Seoul, 151-742 Korea
{yhdfly, moon}@soar.snu.ac.kr

Abstract. It is crucial to maximize marketing efficiency and customer satisfaction in personalized marketing. In this paper, we raise the multiple recommendation problem which occurs when performing several personalized campaigns simultaneously. We formulate the multi-campaign assignment problem to solve this issue and propose methods for solving the problem. The most notable element is the Lagrange multiplier method. Roughly speaking, Lagrange multiplier reduces problem complexity with a minor impact on optimality. However, it is not easy to find Lagrange multipliers in exact accord with problem constraints. We use a genetic algorithm for finding optimal Lagrange multipliers. Through the experiments, we verify the effectiveness of the problem formulation and our genetic approach.

1 Introduction

Customer Relationship Management (CRM) is crucial in acquiring and maintaining loyal customers. To maximize revenue and customer satisfaction, companies try to provide personalized services for customers. A representative effort is one-to-one marketing. The fast development of Internet and mobile communication has enhanced the market for one-to-one marketing. A personalized campaign targets the most attractive customers with respect to the subject of the campaign. So it is important to predict customer preferences for campaigns. Collaborative Filtering (CF) [11] and various data mining techniques including clustering [9] and nearest neighbor algorithm [6] are used to predict customer preferences for campaigns [3]. Especially, since CF is fast and simple, it is widely used for personalization in e-commerce [10] [8]. There have been a number of customer-preference estimation methods based on CF [13] [7] [1]. A survey for recommender systems in e-commerce is given in [12].

As personalized campaigns are frequently performed, several campaigns often happen to run simultaneously. It is often the case that an attractive customer for a specific campaign tends to be attractive for other campaigns. If we perform separate campaigns without considering this problem, some customers may be bombarded by a considerable number of campaigns. We call this the multiple recommendation problem. The larger the number of recommendations for a customer, the lower the customer interest for campaigns. In the long run, the

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1065–1077, 2004.

customer response for campaigns drops. It lowers the marketing efficiency as well as customer satisfaction, which diminishes the customer loyalty and sometimes results in the disastrous "churning." Unfortunately, traditional methods only focused on the effectiveness of individual campaigns and did not consider the problem with respect to the multiple recommendations. In the situation that several campaigns are performed at the same time, it is necessary to find the optimum campaign assignment to customers considering the recommendations in other campaigns.

In this paper, we define the multi-campaign assignment problem (MCAP) considering the multiple recommendation problem and propose three methods for the issue. We show that one can solve the MCAP to optimality by a dynamic programming algorithm. Although the dynamic programming algorithm guarantees optimal solutions, it becomes intractable for large problems. We thus propose a constructive heuristic that not only has practical time complexity but also shows good performance. However, since it is a heuristic, it does not guarantee optimal solutions. Finally, we propose the Lagrange multiplier method with the advantages of both the dynamic programming method and the constructive heuristic. It has linear time complexity for the fixed number of campaigns and sacrifices little on the optimality. Furthermore, the Lagrange multiplier method also provides a good upper bound and can be used to measure the suboptimality of other heuristics. But, randomly generated Lagrange multipliers do not satisfy problem constraints. It is not easy to find Lagrange multipliers in exact accord with problem constraints. Since it is important to find good Lagrange multipliers, we use genetic algorithms to optimize Lagrange multipliers. We also verify the effectiveness of the proposed genetic algorithm with field data.

The remainder of this paper is organized as follows. In Section 2, we describe the multi-campaign assignment problem. The detailed description of response suppression function, the key element for the problem, is given in Section 3. In Section 4, we propose three algorithms for the problem: a dynamic programming algorithm, a heuristic algorithm, and the Lagrange multiplier method. In Section 5, we propose a genetic algorithm for optimizing Lagrange multipliers. We show experimental results in Section 6 and finally make conclusions in Section 7.

2 Problem Formulation

The multi-campaign assignment problem is to find customer-campaign assignments that maximize the effects of campaigns. The main difference with independent campaigns lies in that the customer response for campaigns is influenced by multiple recommendations.

Let N be the number of customers, $C = \{1, 2, ..., N\}$ be the set of customers, and K be the number of campaigns. In the following, we describe the input, output, constraints, and evaluation function for the problem.

Input: For each customer, the preference for each campaign is given. Each campaign has a weight. A response suppression function R related to multiple recommendations is given.

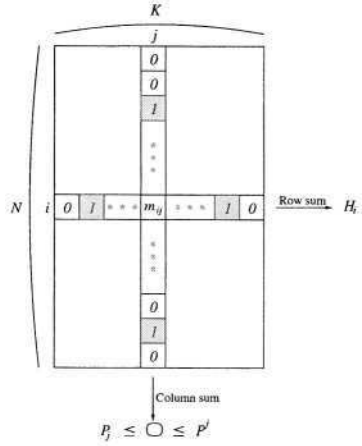


Fig. 1. The campaign assignment matrix $M = (m_{ij})$

– $w_1, w_2, \ldots, w_K$: the weight of each campaign ($w_j > 0$ for each campaign j).
– $f_1, f_2, \ldots, f_K : C \longrightarrow [0, \infty)$: the preference function of each campaign.
– $R : \mathcal{N} \longrightarrow [0, 1]$: the response suppression function with respect to the number of recommendations.

The preferences for a campaign can be gotten from some existing method such as CF. If H_i is the number of multiple recommendations for customer i and $f_j(i)$ is the preference of customer i for campaign j, the actual preference of customer i for campaign j becomes $R(H_i) \cdot f_j(i)$.

Constraints: The maximum and minimum numbers of recommendations for each campaign are enforced. Let P^j be the maximum number of recommendations for campaign j, and P_j be the minimum number of recommendations for campaign j. Then the number of recommendations in campaign j is between P_j and P^j. These constraints can be tight or loose according to the situation of business.

Output: The output is a binary campaign assignment matrix $M = (m_{ij})$ in which m_{ij} indicates whether campaign j is assigned to customer i. Figure 1 shows an example campaign assignment matrix.

Evaluation: The *campaign preference* for campaign j is defined to be the actual preference sum of recommended customers for campaign j as follows: $\sum_{i \in C; m_{ij}=1} R(H_i) \cdot f_j(i)$. The fitness *F(M)* of a campaign assignment matrix M is the weighted sum of campaign preferences.

$$F(M) = \sum_{j=1}^{K} \left(w_j \cdot \sum_{i \in C; m_{ij}=1} R(H_i) \cdot f_j(i) \right)$$

The objective is to find a matrix *M* that maximizes *F*.

Nomenclature: The nomenclature that would be used in the remainder of this paper is given in the below:

$\boldsymbol{w} = (w_1, w_2, \ldots, w_K) \in \mathcal{R}^K$: the campaign weight vector
$f_j : C \longrightarrow [0, \infty)$: the preference function for each campaign j
$R : \mathcal{N} \longrightarrow [0, 1]$: the response suppression function
(for convenience, we assume $R(0) = 0$)
$\boldsymbol{p^*} = (P^1, P^2, \ldots, P^K) \in \mathcal{N}^K$: the upper bound constraint vector
$\boldsymbol{p_*} = (P_1, P_2, \ldots, P_K) \in \mathcal{N}^K$: the lower bound constraint vector
$M = (m_{ij})$: the $N \times K$ binary campaign assignment matrix
$M' = (m'_{ij})$: the $N \times K$ real matrix where $m'_{ij} = f_j(i) \cdot m_{ij}$
$\boldsymbol{m_i} = (m_{i1}, m_{i2}, \ldots, m_{iK})$: the i^{th} row vector of the matrix M
$\boldsymbol{m'_i} = (m'_{i1}, m'_{i2}, \ldots, m'_{iK})$: the i^{th} row vector of the matrix M'
$\mathbf{1}_n$: an n-dimensional vector (1,1,...,1)
$H_i = \boldsymbol{m_i} \cdot \mathbf{1}_K^T$: the number of multiple recommendations for customer i
$\sigma_i = \boldsymbol{m'_i} \cdot \boldsymbol{w}^T$: the weighted sum of preferences of customer i for recommended campaigns

Formal Definition: More formally, we define the multi-campaign assignment problem (MCAP) as follows.

Definition 1 *The* multi-campaign assignment problem *is to find a campaign assignment matrix* $M = (m_{ij})$ *that maximizes* $\langle \boldsymbol{w}, \boldsymbol{R}(\mathbf{1}_K M^T) \cdot M' \rangle$ *subject to* $\boldsymbol{p_*} \leq \mathbf{1}_N M \leq \boldsymbol{p^*}$, *where* $\boldsymbol{R}(x_1, x_2, \ldots, x_n) = (R(x_1), R(x_2), \ldots, R(x_n))$.

3 The Response Suppression Function

In case of multiple campaign recommendations, the customer response rate drops as the number of recommendations grows. We introduce the response suppression function for the response-rate degradation with multiple recommendations. Finding a reasonable response suppression function is the subject to be tuned over abundant field data. The optimal response suppression function depends on situations and it is a longterm research topic. Instead, we devised a number of suppression functions. Definitely, the function should be monotonic nonincreasing.

Figure 2 shows a basic response suppression function. The function is nonnegative monotonic nonincreasing with the maximum value one. It was derived from the Gaussian function. By the function, the preference for a campaign drops to, e.g., one third when four campaigns are performed simultaneously to a customer. In the experiment, we used the function R as the response suppression function.

4 Methods

4.1 Dynamic Programming

We can find the optimal campaign assignment matrix of the MCAP using dynamic programming (DP). DP has been useful for diverse problems [2] [5]. We

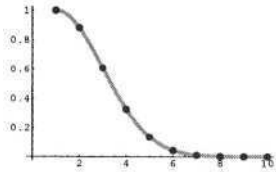


Fig. 2. Basic response suppression function ($R(x) = 0$ for $x \geq 11$)

```
DP(R, P*, P*)
{
    S_0(0_K) = 0;
    for each v such that 0_K ≠ v ≤ P*, S_0(v) = −∞;
    for i = 1 to N
        for each v ≤ P* {
            S_i(v) = max_{m_i; ∀j,m_ij≤v_j} (S_{i−1}(v − m_i) + R(H_i) · σ_i);
            L_i(v) = argmax_{m_i; ∀j,m_ij≤v_j} (S_{i−1}(v − m_i) + R(H_i) · σ_i);
        }
    optimum_value = max_{v; p*≤v≤P*} S_N(v);
    rv = argmax_{v; p*≤v≤P*} S_N(v);
    for i = N to 1 {
        m_i = L_i(rv);
        rv = rv − m_i;
    }
    return optimum_value and M;
}
```

$\mathbf{0}_n$ is an n-dimensional vector $(0, 0, \ldots, 0)$ and $\boldsymbol{v} = (v_1, v_2, \ldots, v_K) \in \mathcal{N}^K$

Fig. 3. A dynamic programming algorithm for MCAP

devised a DP algorithm for MCAP. Figure 3 shows the pseudo-code. In the algorithm, $S_i(\boldsymbol{v})$ means the optimum fitness of the multi-campaign assignment problem with $P_* = P^* = \boldsymbol{v}$ and the customer set $\{1, 2, \ldots, i\}$. The algorithm requires $O(NK \cdot \Pi_{j=1}^{K} P^j)$ space. Since the maximum number of $\boldsymbol{m_i}$ configurations is 2^K, it takes $O(NK2^K \cdot \Pi_{j=1}^{K} P^j)$ time. If K is a fixed number, this is a polynomial-time algorithm. However, when K is not small and almost all P^j's are $\Omega(N)$, it is nearly intractable. An optimum assignment matrix is obtained by backward links $L_i(\boldsymbol{v})$ stored during the process of dynamic programming. The proposed DP algorithm is only applicable to problems with small K, N pairs. Thus we need heuristics for large problems. Since the DP algorithm guarantees optimal solutions, it is useful in evaluating the suboptimality of other heuristics proposed in the next subsection.

```
M = O;
Create an AVL tree;
for each campaign j
        Find topmost α customers with high gain values, and
            for each customer i of them, insert the node (g_ij, (i, j))
            into the AVL tree;
do {
        Choose a maximum gain node (i, j) and delete it from the AVL tree;
        if (g_ij is not positive and every campaign satisfies
            its constraint on the minimum number of recommendations)
                then break;
        if (the campaign j is not full) then {
                Recommend the campaign j to the customer i; // m_ij ← 1
                for each not-full campaign k
                        Update the gain values g_ik in the AVL tree;
        }
} until (the AVL tree is empty or every campaign is full)
```

∗ We set α to be $N/2$ in our experiments.

Fig. 4. The constructive assignment algorithm

4.2 Heuristic Algorithm

Starting at the initial situation that no customer is recommended any campaigns, we iteratively assign campaigns to customers by a greedy method. We call this algorithm Constructive Assignment Algorithm (CAA). Define the *gain* g_{ij} of a pair (customer i, campaign j) to be the amount of fitness gain by assigning campaign j to customer i. Initially, the gain g_{ij} is equal to $w_j f_j(i)$, the product of campaign weight and the preference of customer i for campaign j. Generally the gain is formulated as: $g_{ij} = R(H_i + 1) \cdot (\sigma_i + w_j f_j(i)) - R(H_i) \cdot \sigma_i$. We use an AVL tree for the efficient management of real-valued gains. Figure 4 shows the template of the CAA. It chooses the most attractive α customers for each campaign and inserts them into the AVL tree. Next, it iteratively performs the following. It chooses a pair (customer i, campaign j) with the maximum gain and, if the gain is positive and campaign j does not exceed the maximum number of recommendations, it recommends campaign j to customer i. If a recommendation is done, it updates the gains of customer i for the other campaigns. We naturally assume that the response suppression function is monotonic nonincreasing. In that case, any gain cannot be positive after the maximum gain drops below zero. When the maximum gain is not positive, the algorithm terminates as far as every campaign satisfies the constraint on the minimum number of recommendations. There are *O(NK)* nodes in the AVL tree during a run of the algorithm. Hence, both the insertion and deletion of a node in the AVL tree take *O(log(NK)),* i.e., *O(logN).* The time complexity of the algorithm is $O(NK^2 logN)$. If the algorithm is implemented without an AVL tree, it would take $O(N^2K^3)$.

LM(R, $\boldsymbol{\lambda}$)
{
  for $i = 1$ **to** N {
    $F_i = max_{\boldsymbol{m_i} \in \{0,1\}^K}(R(H_i) \cdot \sigma_i - \langle \boldsymbol{\lambda}, \boldsymbol{m_i} \rangle)$;
    $\tilde{\boldsymbol{m}}_i = argmax_{\boldsymbol{m_i} \in \{0,1\}^K}(R(H_i) \cdot \sigma_i - \langle \boldsymbol{\lambda}, \boldsymbol{m_i} \rangle)$;
  }
  $\boldsymbol{p} = \sum_{i=1}^{N} \tilde{\boldsymbol{m}}_i$;
  optimum_value $= \sum_{i=1}^{N} F_i + \langle \boldsymbol{\lambda}, \boldsymbol{p} \rangle$;
  return *optimum_value*, $\tilde{M} = (\tilde{m_{ij}})$, and $\boldsymbol{p}$;
}

$\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_K) \in \mathcal{R}^K$ and $\boldsymbol{p} = (p_1, p_2, \ldots, p_K) \in \mathcal{N}^K$

Fig. 5. Lagrange multiplier method for MCAP

4.3 Lagrange Multipliers

Since the multi-campaign assignment problem is a constraint optimization problem, it can be solved using Lagrange multipliers as in the following:

$$Max\ \{\langle \boldsymbol{w}, \boldsymbol{R}(\mathbf{1}_K M^T) \cdot M' \rangle - \langle \boldsymbol{\lambda}, \mathbf{1}_N M \rangle\}.$$

However, since it is a discrete problem which is not differentiable, it is formulated to a restricted form. Figure 5 shows the pseudo-code of the Lagrange multiplier method for MCAP. The following proposition guarantees its optimality.

Proposition 1 *(Optimality) Given the constraint vector* $\boldsymbol{p}$*, Lagrange multiplier method outputs an optimum matrix* $\tilde{M} = (\tilde{m_{ij}})$.

Proof: Suppose that $\tilde{M} = (\tilde{m_{ij}})$ is not an optimum matrix. Let $\bar{M} = (\bar{m_{ij}})$ be an optimum matrix with $\boldsymbol{p}$ as the constraint vector. Let $\bar{F_i}$ be $R(H_i) \cdot \sigma_i - \langle \boldsymbol{\lambda}, \bar{\boldsymbol{m}}_i \rangle$ for each i in campaign assignment matrix $\bar{M}$. For each i, by the optimality of F_i, $F_i \geq \bar{F_i}$. Since $\boldsymbol{p}$ is given, $\sum_{i=1}^{N} F_i + \langle \boldsymbol{\lambda}, \boldsymbol{p} \rangle \geq \sum_{i=1}^{N} \bar{F_i} + \langle \boldsymbol{\lambda}, \boldsymbol{p} \rangle$. This contradicts that $\tilde{M} = (\tilde{m_{ij}})$ is not an optimum matrix. Therefore, $\tilde{M} = (\tilde{m_{ij}})$ is an optimum matrix. ■

Given a Lagrange multiplier vector $\boldsymbol{\lambda}$, we can get the constraint vector $\boldsymbol{p}$ and the optimum result with $\boldsymbol{p}$ by the Lagrange multiplier method.

Proposition 2 *Let* $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_K)$ *and* $\boldsymbol{\lambda}' = (\lambda'_1, \lambda'_2, \ldots, \lambda'_K)$. *Suppose that* $\lambda_i = \lambda'_i$ *for* $i \neq k$ *and* $\lambda_k \neq \lambda'_k$. *Then, if* $\lambda_k < \lambda'_k$, $p_k(\boldsymbol{\lambda}) \geq p_k(\boldsymbol{\lambda}')$ *and if* $\lambda_k > \lambda'_k$, $p_k(\boldsymbol{\lambda}) \leq p_k(\boldsymbol{\lambda}')$.

Proof: Suppose that $\boldsymbol{\lambda}$ and $\boldsymbol{\lambda}'$ correspond to $M = (m_{ij})$ and $M' = (m'_{ij})$, respectively. By the optimality of $\boldsymbol{\lambda}$, $\sum_{i=1}^{N} F_i(\boldsymbol{\lambda}) \geq F'_i(\boldsymbol{\lambda})$. By the optimality of $\boldsymbol{\lambda}'$, $\sum_{i=1}^{N} F'_i(\boldsymbol{\lambda}') \geq F_i(\boldsymbol{\lambda}')$. From the summation of above two formulas,

$$\sum_{i=1}^{N} -\langle \boldsymbol{\lambda}, \boldsymbol{m}_i \rangle + \sum_{i=1}^{N} -\langle \boldsymbol{\lambda}', \boldsymbol{m}'_i \rangle \geq \sum_{i=1}^{N} -\langle \boldsymbol{\lambda}, \boldsymbol{m}'_i \rangle + \sum_{i=1}^{N} -\langle \boldsymbol{\lambda}', \boldsymbol{m}_i \rangle$$

$$\Longleftrightarrow \sum_{i=1}^{N} -\lambda_k m_{ik} + \sum_{i=1}^{N} -\lambda'_k m'_{ik} \geq \sum_{i=1}^{N} -\lambda_k m'_{ik} + \sum_{i=1}^{N} -\lambda'_k m_{ik}$$

$$\Longleftrightarrow -\lambda_k p_k(\boldsymbol{\lambda}) - \lambda'_k p_k(\boldsymbol{\lambda}') \geq -\lambda_k p_k(\boldsymbol{\lambda}') - \lambda'_k p_k(\boldsymbol{\lambda})$$

$$\Longleftrightarrow (\lambda'_k - \lambda_k)(p_k(\boldsymbol{\lambda}) - p_k(\boldsymbol{\lambda}')) \geq 0. \quad \blacksquare$$

Proposition 3 *(Sensitivity) Suppose that* $\boldsymbol{\lambda}_0$ *and* $\boldsymbol{\lambda}_1$ *correspond to* $\{M_0, \boldsymbol{p}_0\}$ *and* $\{M_1, \boldsymbol{p}_1\}$*, respectively. Then, the following inequalities are satisfied.*

$$\langle \boldsymbol{\lambda}_0, \boldsymbol{p}_0 - \boldsymbol{p}_1 \rangle \leq F(M_0) - F(M_1) \leq \langle \boldsymbol{\lambda}_1, \boldsymbol{p}_0 - \boldsymbol{p}_1 \rangle.$$

In particular, for any assignment matrix M, $F(M) - F(M_0) \leq \langle \boldsymbol{\lambda}_0, \mathbf{1}_N M - \boldsymbol{p}_0 \rangle$.

Proof: By the optimality of $\boldsymbol{\lambda}_0$, $F(M_0) - \langle \boldsymbol{\lambda}_0, \boldsymbol{p}_0 \rangle \geq F(M_1) - \langle \boldsymbol{\lambda}_0, \boldsymbol{p}_1 \rangle$. Hence, $F(M_0) - F(M_1) \geq \langle \boldsymbol{\lambda}_0, \boldsymbol{p}_0 - \boldsymbol{p}_1 \rangle$. By the optimality of $\boldsymbol{\lambda}_1$, $F(M_1) - \langle \boldsymbol{\lambda}_1, \boldsymbol{p}_1 \rangle \geq F(M_0) - \langle \boldsymbol{\lambda}_1, \boldsymbol{p}_0 \rangle$. Hence, $F(M_0) - F(M_1) \leq \langle \boldsymbol{\lambda}_1, \boldsymbol{p}_0 - \boldsymbol{p}_1 \rangle$. $\blacksquare$

Fact 1 *F is a nondecreasing function of $\boldsymbol{p}$ (i.e., $\boldsymbol{p} < \boldsymbol{p}' \Rightarrow F \leq F'$). This implies $\boldsymbol{\lambda} \geq \mathbf{0}_K$.*

From Proposition 2, p_i inversely grows with λ_i. However, it is not easy to find $\boldsymbol{\lambda}$ satisfying $\boldsymbol{p}_* \leq \boldsymbol{p} \leq \boldsymbol{p}^*$. Proposition 3 shows that our Lagrange multiplier method also gives good upper bounds. By using the Lagrange multiplier method, the problem of finding the optimum campaign assignment matrix becomes that of finding a K-dimensional real vector $\boldsymbol{\lambda}$ with Lagrange multipliers. The Lagrange multiplier method takes $O(NK2^K)$ time. It is more tractable than the original problem. Roughly, for a fixed number K, the problem size is lowered from $O(N^{K+1})$ to $O(N)$.

5 A Genetic Algorithm for Optimizing Lagrange Multipliers

We propose a genetic algorithm (GA) for optimizing Lagrange multipliers. It conducts a search using an evaluation function with penalties for violated constraints. The search space with N customers and K campaigns has $\Pi_{i=0}^{K} \Pi_{j=P_i}^{P^i} \binom{N}{j}$ elements if all possibilities are considered. But, too large a problem size may make the search intractable. Our GA provides an alternative search method to find a good campaign assignment matrix by optimizing K Lagrange multipliers instead of directly dealing with the campaign assignment matrix.

5.1 Genetic Operators

The general framework of a typical steady-state genetic algorithm is used in our GA. In the following, we describe each part of the GA.

- *Encoding:* Each solution in the population is represented by a chromosome. Each chromosome contains K Lagrange multipliers. A real encoding is used for representing the chromosome $\boldsymbol{\lambda}$. From Fact 1, each Lagrange multiplier is nonnegative. So we set a gene corresponding to a Lagrange multiplier to be a real value between 0.0 and 1.0.
- *Initialization:* The GA first creates p real vectors between $\mathbf{0}_K$ and $\mathbf{1}_K$ at random. We set the population size p to be 100.
- *Selection and crossover:* To select two parents, we use a proportional selection scheme where the probability for the best solution to be chosen is four times higher than that for the worst solution. A crossover operator creates a new offspring by combining parts of the parents. We use the uniform crossover [14].
- *Mutation:* After the crossover, mutation operator is applied to the offspring. We use a variant of Gaussian mutation. We devised it considering the relation between λ_i and p_i given in Proposition 2. For each selected gene λ_i, we choose a Gaussian random real number t normally distributed with parameters $\mu = 0$ and $\sigma^2 = 1$ (i.e., $t \sim N(0,1)$). If the corresponding constraint p_i is less than P^i, we set λ_i to $\lambda_i + (1-\lambda_i) \cdot |t|/\gamma$. Otherwise, we set λ_i to $\lambda_i - \lambda_i \cdot |t|/\gamma$. Mutation rate is 0.05 and the constant γ is 2.58.
- *Replacement and stopping condition:* After generating an offspring, our GA replaces the worse of the two parents with the offspring. It is called *preselection* replacement [4]. Our GA stops when one of the following two conditions is satisfied: i) the number of generations reaches 15,000, ii) when the fitness of the worst chromosome is equal to the fitness of the best one.

5.2 Evaluation Function

Our evaluation function is to find a Lagrange multiplier vector $\boldsymbol{\lambda}$ that has high fitness satisfying the constraints as *much* as possible. In our GA, the following evaluation function is used: $F(M) - \sum_{j=1}^{K} c_j \cdot excess(j)$ where c_j is the campaign penalty and $excess(j)$ is the number of exceeded recommendations for campaign j. In this paper, we used K times the weighted average preference of campaign j as the campaign penalty c_j (i.e., $c_j = K \cdot w_j \cdot \frac{1}{N} \sum_{i=1}^{N} f_j(i)$).

6 Experimental Results

6.1 Inputs and Parameters

We used the preference values estimated by CF from a set of field data with 48,559 customers and 10 campaigns. We used the function R which was derived from Gaussian function as the response suppression function. The weight for each campaign was given equally 0.1. The maximum number of recommendations for each campaign was set to 7,284, equal to 15% of the total number of customers. The minimum number of recommendations was set to 0.

The average preference of customers for a campaign was 4.74. We examined the Pearson correlation coefficient of preferences for each pair of campaigns.

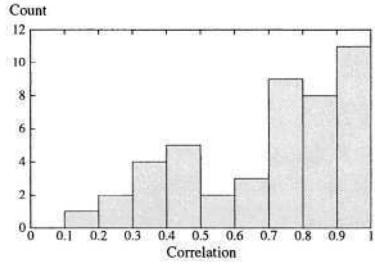


Fig. 6. Histogram for Pearson correlation coefficient of each campaign pair

Table 1. Comparison for Small Data Sets ($K = 3$)

Method	$N = 200$	$N = 400$	$N = 600$	$N = 800$	$N = 1000$
CAA	733.61	1303.98	2030.10	2725.59	3479.30
(Time*)	(0.0010)	(0.0021)	(0.0032)	(0.0045)	(0.0058)
DP†	734.64	1304.43	2031.22	2726.99	3483.02
(Time*)	(145.82)	(2336.41)	(11553.92)	(28838.84)	(71479.06)

Maximum 50% and minimum 0% recommendation for each campaign.
Equally weighted campaigns; i.e., $w_j = 0.33$ for each campaign j.
† The optimum value.
* CPU seconds on Pentium III 1 GHz.

Figure 6 shows its histogram. Thirty three pairs (about 73%) out of totally 45 pairs showed higher correlation coefficient than 0.5. This property of field data provides a good reason for the need of MCAP modeling.

6.2 Analysis of Results

First, we compare the proposed heuristic algorithm with the DP algorithm which guarantees optimality. Due to the running time of dynamic programming, we restricted the instances with up to 1,000 customers and three campaigns. Table 1 shows their performance. We chose three campaigns among the 10 campaigns, and the sets of customers were sampled at random to prepare the sets with sizes from 200 to 1,000. The maximum number of recommended customers was set to be a half the total number of customers. The minimum number of recommendations was set to 0. The CAA was much faster than the dynamic programming algorithm while its results reached fairly close to the optimal solutions. This implies that CAA is an attractive practical heuristic.

Table 2 shows the performance of the independent campaign and various multi-campaign algorithms in the multi-campaign formulation. We use all the 48,559 customers and 10 campaigns here. The figures in the table represent the fitness values *F(M)* described in Section 2. The results of "Independent" campaign are from K independent campaigns without considering their relationships with others. Although the independent campaign was better than the "Random"

Table 2. Comparison of Algorithms

Method	$K = 7$	$K = 8$	$K = 9$	$K = 10$
Independent	38546.25	36779.68	30259.65	26452.75
Random	32487.36	29385.14	27187.26	25951.19
CAA	85565.05	79885.40	71863.76	66473.63
LM-Random†	45401.90	53638.58	52351.99	46990.46
LM-GA†	**86474.30**	**80287.72**	**72518.22**	**66932.79**
LM Upper Bound*	87776.38	81428.48	73446.60	67654.77

† Average over 10 runs.
* Upper bound by Proposition 3 and the results of LM-GA.

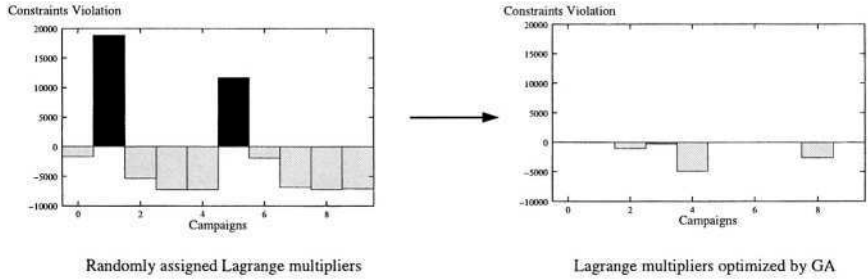


Fig. 7. An example of optimized Lagrange multipliers

assignment in multi-campaign formulation, it was not comparable to the other multi-campaign algorithms. The results of Random assignment are the average over 100 runs. The solution fitnesses of CAA heuristic were overall more than two times higher than those of the independent campaign.

Next, we compare the proposed heuristic algorithm with the Lagrange multiplier method. Table 2 shows their performance, too. LM-Random means the best result among randomly generated 15,000 Lagrange multiplier vectors. Even in the best result, there were constraints violations for some campaigns. However, when Lagrange multipliers are optimized by a genetic algorithm (LM-GA), we always found the best-quality solutions satisfying all constraints. Moreover, LM-GA performed better than the proposed deterministic heuristic, CAA. Figure 7 shows an example of optimized 10 Lagrange multipliers. For a typical randomly generated Lagrange multiplier vector, we can observe that excessive constraints violations for some campaigns (the left figure of Fig. 7). However, with the Lagrange multipliers optimized by GA, we can see that all constraints were satisfied (the right figure of Fig. 7).

As a final solution of the GA with $K = 10$ and fitness 66980.76, we got the following constraint vector: $\boldsymbol{p} = \{7254, 7269, 6238, 6982, 2422, 7284, 7237, 7282, 4675, 7260\}$ (on the right side of Fig. 7). When the vector $\boldsymbol{p}$ is used as the upper bound constraint vector $\boldsymbol{p}^*$ of the MCAP, CAA produced the campaign assignment matrix with a fitness value 65819.80. This gives another usage of the

Lagrange multiplier method. The suboptimality of heuristic algorithms can be measured by using the optimality of the Lagrange multiplier method.

7 Conclusions

The representative contributions of this paper are as follows. First, we proposed and formulated the multi-campaign assignment problem (MCAP). Second, we presented a dynamic programming algorithm and a constructive heuristic algorithm for MCAP. Finally, we proposed Lagrange multiplier method for the MCAP and optimized it using a genetic algorithm. Their performance was examined with experiments.

Our Lagrange multiplier method is fast and outputs optimal solutions. But, it is not easy to find Lagrange multipliers satisfying all constraints. When combined with the genetic algorithm, we could find high-quality Lagrange multipliers. The Lagrange multiplier method may be combined with other metaheuristics as well such as evolutionary strategy, simulated annealing, tabu search, and large-step Markov chains. Experimentation with respect to these methods is left for future study.

Acknowledgments. This study was supported by Brain Korea 21 Project. The ICT at Seoul National University provided research facilities for this study.

References

1. C. C. Aggarwal, J. L. Wolf, K. L. Wu, and P. S. Yu. Horting hatches an egg: A new graph-theoretic approach to collaborative filtering. In *Knowledge Discovery and Data Mining,* pages 201–212, 1999.
2. R. Bellman. *Dynamic Programming.* Princeton University Press, 1957.
3. M. Berry and G. Linoff. *Data Mining Techniques For Marketing, Sales, and Customer Support.* John Wiley & Sons, Inc, 1997.
4. D. Cavicchio. *Adaptive Search Using Simulated Evolution.* PhD thesis, University of Michigan, Ann Arbor, MI, 1970.
5. S. E. Dreyfus and A. M. Law. *The Art and Theory of Dynamic Programming.* Academic Press, 1977.
6. C. Feustel and L. Shapiro. The nearest neighbor problem in an abstract metric space. *Pattern Recognition Letters,* 1:125–128, 1982.
7. D. Greening. Building consumer trust with accurate product recommendations. Technical Report LMWSWP-210-6966, LikeMinds White Paper, 1997.
8. J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval,* pages 230–237, 1999.
9. A. K. Jain and R. C. Dubes. *Algorithms For Clustering Data.* Prentice Hall, 1988.
10. J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordan, and J. Riedl. GroupLens: applying collaborative filtering to Usenet news. *Communications of the ACM,* 40:77–87, 1997.

11. P. Resnick, N. Iacovou, M. Sushak, P. Bergstrom, and J. Riedl. GroupLens: An open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 Computer Supported Collaborative Work Conference,* pages 175–186, 1994.
12. J. B. Schafer, J. A. Konstan, and J. Riedi. Recommender systems in e-commerce. In *ACM Conference on Electronic Commerce,* pages 158–166, 1999.
13. U. Shardanand and P. Maes. Social information filtering: Algorithms for automating "word of mouth". In *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems,* volume 1, pages 210–217, 1995.
14. G. Syswerda. Uniform crossover in genetic algorithms. In *Third International Conference on Genetic Algorithms,* pages 2–9, 1989.

Biomass Inferential Sensor Based on Ensemble of Models Generated by Genetic Programming

Arthur Kordon[1*], Elsa Jordaan[2], Lawrence Chew[3], Guido Smits[2], Torben Bruck[3], Keith Haney[3], and Annika Jenings[3]

[1]The Dow Chemical Company, Corporate R&D, 2301 N Brazosport Blvd, Freeport, TX, 77541, USA
[2] Dow Benelux BV, Corporate R&D, 5 Herbert H Dowweg, Terneuzen, The Netherlands
[3]The Dow Chemical Company, Biotechnology, 5501 Oberlin Dr., San Diego, CA 92121, USA
* corresponding author akordon@dow.com

Abstract. A successful industrial application of a novel type biomass estimator based on Genetic Programming (GP) is described in the paper. The biomass is inferred from other available measurements via an ensemble of nonlinear functions, generated by GP. The models are selected on the Pareto front of performance-complexity plane. The advantages of the proposed inferential sensor are: direct implementation into almost any process control system, rudimentary self-assessment capabilities, better robustness toward batch variations, and more effective maintenance. The biomass inferential sensor has been applied in high cell density microbial fermentations at The Dow Chemical Company.

1 Introduction

Soft (or inferential) sensors infer important process variables (called outputs) from available hardware sensors (called inputs). Usually the outputs are measured infrequently by lab analysis, material property tests, expensive gas chromatograph analysis, etc. Furthermore, the output measurement is very often performed off-line and then introduced into the on-line process monitoring and control system. Soft sensors, on the other hand, can predict these outputs online and frequently using either data supplied from standard, and frequently cheap, hardware sensors or from other soft sensors.

Different inference mechanisms can be used for soft sensor development. If there is a clear understanding of the physics and chemistry of the process, the inferred value can be derived from a fundamental model. Another option is to estimate the parameters of the fundamental model via Kalman Filter or Extended Kalman Filter. There are cases when the input/output relationship is linear and can be represented either by linear regression or by a multivariate model. The most general representation of the inferential sensor, however, is as a nonlinear empirical model. The first breakthrough technology for soft sensor development was neural networks because of their ability to capture nonlinear relationships and their adequate

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1078–1089, 2004.

framework for industrial use (i.e. no additional cost for fundamental model building and data collection based on design of experiments).

The common methodology of building neural net soft sensors and the practical issues of their implementation have been discussed in detail in [2]. Thousands of soft sensor applications in the chemical, petro-chemical, pharmaceutical, power generation, and other industries have been claimed by the key software vendors. However, despite their successes, most commercially available neural net packages are still based on a classical back-propagation algorithm. Some other recent approaches, like Bayesian neural networks [18] are still in the research domain and have not been accepted for industrial applications. As a result, those commercial neural networks generally exhibit poor generalization capability outside the range of training data [1]. This can result in poor performance of the model and unreliable prediction in new operating conditions. Another drawback is that such packages usually yield neural net structures with unnecessarily high complexity. Selection of the neural net structure is still an *ad hoc* process and very often leads to inefficient and complex solutions. As a result of this inefficient structure and reduced robustness, there is a necessity of frequent re-training of the empirical model. The final effect of all of these problems is an increased maintenance cost and gradually decreased performance and credibility [3].

The need for robustness toward process variability, the ability to handle industrial data (e.g., missing data, measurement noise, operator intervention on data, etc.) and ease of model maintenance are key issues for mass-scale application of reliable inferential sensors. Several machine-learning approaches have the potential to contribute to the solution of this important problem. Stacked analytical neural networks (internally developed in The Dow Chemical Company) allow very fast model development of parsimonious black-box models with confidence limits [4]. Genetic Programming (GP) can generate explicit functional solutions that are very convenient for direct on-line implementation in the existing process information and control systems [5]. Recently, Support Vector Machines (SVM) give tremendous opportunities for building empirical models with very good generalization capability due to the use of the structural risk minimization principle [6]. At the same time, each approach has its own weaknesses, which reduces the implementation space and which make it difficult to design the robust soft sensor based on separate computational intelligence techniques. An alternative, more integrated approach for a "second generation" soft sensor development is described in [4]. It combines a nonlinear sensitivity and time-delay analysis based on Stacked Analytical Neural Nets with outlier detection and condensed data selection driven by the Support Vector Machines. The derived soft sensor is generated by GP as an analytical function. The integrated methodology amplifies the advantages of the individual techniques, significantly reduces the development time, and delivers robust soft sensors with low maintenance cost. The methodology has been successfully applied in The Dow Chemical Company for critical parameter prediction in a chemical reactor [7], for interface level estimation [8], and for emission estimation [9]. However, the increased requirements for robustness towards batch-to-batch variation in inferential sensors applied on batch processes need to be addressed. A potential solution of using an ensemble of symbolic regression predictors, instead of one model, is proposed in this paper. The selected models of the ensemble occupy the Pareto front on the performance-model complexity plane. The experience from implementing the

proposed methodology for biomass estimation during the growth phase of fed-batch fermentation is described in this paper.

2 The Need for Biomass Estimation in Batch Fermentation Processes

Biomass monitoring is fundamental to tracking cell growth and performance in bacterial fermentation processes. In standard fed-batch fermentations of recombinant microorganisms, biomass determination over time allows for calculation of growth rates during the growth phase. Slow growth rates can indicate non-optimal fermentation conditions which can then be a basis for further optimization of growth medium, conditions or substrate feeding. Biomass monitoring is also needed to determine the most optimum time to induce recombinant product formation. During the production phase, biomass decline can forecast onset of cell lysis which, if allowed to continue, can result in decreased product yields. Biomass monitoring therefore can aid in determination of the appropriate time for process termination. In fed-batch fermentations, biomass data can also be used to determine feed rates when yield coefficients are known. Certain high cell density fermentations require growth rates to be controlled in order to prevent accumulation of by-products such as acetic acid or ethanol, which if left unchecked can be detrimental. Biomass monitoring also acts as an indicator of batch-to-batch variability.

There are several inferential sensors implementations for biomass estimation in different continuous and fed-batch bioprocesses [10], [11], [12]. Usually the biomass concentration is determined off-line by lab analysis every 2-4 hours. However, these infrequent measurements can lead to poor control and on-line estimates would be more preferred. In [10], a soft sensor for biomass estimation based on two process variables – fermenter dilution rate and carbon dioxide evolution rate (CER) successfully estimated biomass in continuous *mycelial* fermentation. The neural net model included six inputs that incorporated process dynamics for three consecutive sampling periods, two hidden layers with 4 neurons, and one output, the biomass estimate. Another successful implementation of a biomass soft sensor for a penicillin fed-batch process is described in [12]. The topology of the neural net in this case is (2-3-1), where the two inputs are the oxygen uptake rate (OUR) and the batch time, and the output is penicillin biomass estimates. Good surveys of the current state of the art of the soft sensors and data driven approaches to bioprocess modeling are given in [13] and [14]. However, all applied neural net-based soft sensors expressed the common problems of poor extrapolation, high batch-to-batch sensitivity, and frequent re-training. As a result, long term maintenance becomes the Achilles heel of neural-net-based biomass estimators that significantly reduces their credibility.

3 Robust Inferential Sensors Based on Ensemble of Predictors

Robustness toward process changes is the key requirement for industrial application of inferential sensors. It is crucial especially for batch processes where the batch-to-batch changes are guaranteed even when applying the most consistent operating

discipline [14]. One of the possible ways to improve robustness toward process changes is by using explicit nonlinear functions derived by symbolic regression. The advantages of this first approach to increase robustness in comparison to black-box models, like neural networks, are as follows: potential for physical interpretation, the ability to examine the behavior of the model outside the training range in an easy and direct way, the ability to impose external constraints in the modeling process and to relieve the extrapolation level of the final model toward process changes, and last, but not least, process engineers are more open to take the risk to implement such type of models. The applicability of symbolic-regression-based inferential sensors has already been demonstrated in several industrial applications of continuous processes [7], [8], and [9]. However, an open issue that influences robustness toward process changes is the control of empirical model complexity. As is well-known, Statistical Learning Theory in general and Support Vector Machines in particular, give explicit control over model complexity by the selection of the number of support vectors [6]. There is a defined optimal complexity for the available data, called the Vapnik-Chervonenkis (VC) dimension [6]. Unfortunately, direct application of these theoretical results to symbolic regression-based models faces difficulties. However, the idea of balancing modeling performance and complexity will be explored by selecting models on the Pareto front only and this is the second approach to increase robustness that will be tested.

The third approach that could improve robustness toward process changes is to use an ensemble of predictors. By combining diverse symbolic regression models it is possible to use the statistical characteristics of the ensemble for more reliable prediction and for model self-assessment [15].

3.1 Integrated Methodology for Robust Inferential Sensors Development

The objective of the integrated methodology is to deliver successful industrial inferential sensors with reduced development time, better generalization capability, and minimal implementation and maintenance cost. The main blocks of the methodology (data pre-processing, nonlinear variable selection based on analytic neural networks, data condensation based on SVM, automatic model generation based on GP, model selection, on-line implementation and maintenance) are described in detail in [4]. The key idea is to optimize the synergy between three methods for empirical model building: neural networks, Support Vector Machines, and Genetic Programming. Special types of neural networks, called analytic neural networks, are used for nonlinear sensitivity analysis and variable selection [4]. The data set is further reduced by using SVM with selected level of complexity (% support vectors). As a result, only relevant variables and information-rich data points are used for GP model generation. In this way the efficiency of symbolic regression, which is computationally-intensive, is increased significantly. In addition, the probability for finding parsimonious solutions increases due to the reduced search space. The integrated methodology has been successfully implemented for several soft sensors on continuous industrial processes [4]. However, the increased requirements for robustness in batch processes require some improvements in the areas of proper model selection and using ensemble of predictors.

3.2 Pareto-Front Based Model Selection

Several thousand empirical models are generated in a typical GP run with at least 20 simulated evolutionary processes of 200 generations. Most of the generated models are with similar performance and proper model selection is non-trivial. The direct approach is to use the R^2-statistic as model selection criterion and to select the "best" model based on the fitness measure at the end of the run.

However, the fitness measure does not take complexity or smoothness of the function into account. Furthermore, it is possible that for a slight decrease in the measure a far less complex function may be obtained that may have higher robustness. For this the experience of the analyst is needed. Therefore it is necessary to extract a manageable number of models to inspect.

One indicator of the complexity of the models in a GP-run is the number of nodes used to define the model. The measure may be misleading for it does not discern between the types of operators used in each node. For example, no distinction is made between an operator that is additive and an operator that is an exponential function. Clearly there is a huge difference in complexity. However, using the number of nodes as an indicative measure can help reduce the number of models to inspect to a reasonable size.

In order to find the right trade-off between complexity and accuracy, the Pareto-front is constructed. The Pareto-front is a concept commonly used in multi-objective optimization [16]. In multi-objective optimization, apart from the solution space, which is constructed by the constraints in terms of the input variables, there is also an objective space. The objective space is a mapping of the solution space onto the objectives. In classical multi-objective optimization, the problem is cast into a single objective optimization problem by defining an *a priori* weighted sum. The solution to the single objective optimization problem is one point in the objective space. However, as the optimal weighted sum is seldom known *a priori,* it is often better to make the final decision from a set of solutions which is independent of the weights. This set of solutions is given by the Pareto-front. The Pareto-front thus represents a surface in the objective space of all possible weighted combinations of the different objectives that optimally satisfy the constraints.

Since the model selection task is in principle a multi-objective problem (i.e. accuracy vs. complexity), the fundamentals of the Pareto-front can be applied. Using the Pareto-front for GP-generated models has many advantages [17]. Firstly, the structural risk minimization principle [6] can be easily applied to GP-generated models. Secondly, it effectively displays the trade-off between the measures, which enables the analyst to make an unbiased decision. Thirdly, as only a small fraction of the generated models in GP will end up on the Pareto-front, the number of models that need to be inspected individually is decreased tremendously. Finally, additional considerations such as variety in input variables used for ensemble construction can be taken into account. For example, if a Pareto-optimal model uses an undesirable transformation or input variable, one could look for an alternative model among the models close to the Pareto-front.

In Fig. 1 the Pareto-front is displayed for a set of GP-generated models in terms of two objectives, ratio of nodes and R^2. The ratio of nodes is a measure of complexity and needs to be minimized. The second objective, R^2 is a measure of the performance of the models. Using $1\text{-}R^2$ instead of R^2 allows easier interpretation as both objectives

are minimized. The Pareto-front models are models for which no improvement on one objective can be obtained without deteriorating another objective. The optimal model will therefore lie somewhere on the Pareto-front. Its position will depend on the problem at hand. For example, if the complexity and performance have equal importance then the optimal model would lie in the lower left corner of the Pareto front. This example shows how the structural risk minimization is used in finding the optimal model as well as the trade-off between complexity and accuracy.

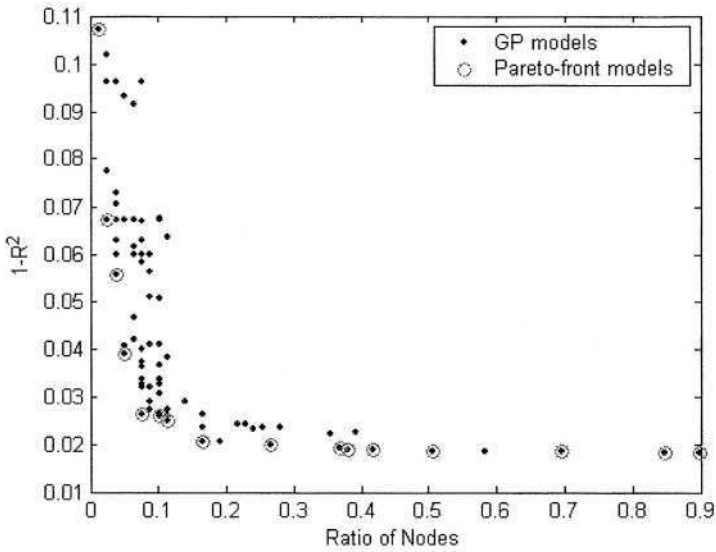


Fig. 1. Pareto-front based on the performance of the training data

Recall that another advantage of using the Pareto front is the reduced number of models to inspect. This is clearly seen in Fig. 1. In total there are 88 models depicted in the figure. However, only 18 of them are identified as Pareto-front models. Furthermore, one can clearly see that using models with a ratio of nodes higher than 0.3 does not result in a significant improvement of R^2. Therefore, the number of models to consider may even be less.

Finally, if the analyst is interested in a set of models to be used in the ensemble, the pool of interesting models can be easily extracted. It is possible to perform model selection in an interactive way such that the analyst can request a model's functional form, error statistics, and response surfaces by clicking on the particular model depicted in the figure.

3.3 Ensemble Design

It is often preferred to not to develop a single-model soft sensor, but a soft sensor that uses an ensemble of models. An ensemble consists of several models that will be used to predict future measurements. The average of the various models will be used as the final prediction.

One advantage of using an ensemble soft sensor is that the standard deviation of the different models in the ensemble can be used as model disagreement measure. This measure indicates how much the performance of the models differs from each other within a given estimate. The smaller the difference, the more certain the operators can be of the prediction made by the soft sensor. This indicator is of special importance for automatically detecting that the inferential sensor goes outside the training range. It is assumed that in the extrapolation mode the different models will diverge more strongly than in interpolation mode, i.e. the standard deviation of their predictions will significantly increase. There are several methods to design an ensemble of predictors [15]. The key issue is that one should be careful not to use models in the ensemble that are too similar in performance, because then a false sense of trust can be created as the standard deviation will be very small. On the other hand, the models should not be too different in performance, because then the predictive power of the soft sensor will be lost. Ideally, the selected models are diverse enough to capture uncertainties, but similar enough to predict well. The final selection of the models to be used in the ensemble depends mainly on the expertise of the analyst.

Another advantage of using an ensemble of models is that it enables redundancy. Since soft sensors are mainly used in processing conditions it often occurs that one or more of the instruments measuring the input variables can fail. If the ensemble consists of models that have different input variables, there will be another model available in the ensemble that still can predict. This prediction may not be the most accurate one, but at least there is a prediction instead of nothing.

In order to select the final models for the ensemble, we inspect the models on the Pareto-fronts based on their performance with both the training and test data. Model selection of the ensemble is based on the following error statistics: correlation coefficient, standard deviation, relative error, R^2-statistic, Root mean square error prediction (RMSEP) and the ratio of nodes. A matrix of variables used by the models is also taken into account. This is needed to identify models that don't use certain variables when ensembles with redundancy are constructed.

4 Development of Biomass Inferential Sensor for the Growth Phase in a Batch Fermentation Process

The described methodology for design of inferential (soft) sensors based on an ensemble of symbolic regression-type predictors will be illustrated with an industrial application of biomass estimator for the growth phase in a batch fermentation process at the Dow Biotech facilities in San Diego.

4.1 Data Collection and Pre-processing

The growth phase data were selected from eleven repeat fermentation runs on different 20L fermentors. Each experiment produced a batch of time series data that included seven input variables (air, pressure, OUR, time elapsed, agitation, nutrient, and total volume) and one output – Optical Density (OD), a measurement for

biomass. The selected inputs have been recommended by the process experts and were sampled every 15 min. The output measurement, which is highly correlated to the biomass, was sampled every two hours. The data from each batch was preprocessed in order to remove outliers, faults and missing values. From the eleven batches eight batches were chosen for training purposes and three batches were set aside for testing purposes.

4.2 GP Models Generation and Selection

The symbolic regression-type models have been derived on a MATLAB toolbox, developed internally in The Dow Chemical Company. The key parameters of a typical Genetic Programming run are given in Table 1. Several GP simulated evolution processes of 20 runs were made varying the values for the number of generations and the parsimony pressure.

Table 1. Genetic Programming Parameter Settings

Parameter	Value/Setting
Random subset selection [%]	100
Number of runs	20
Population size	100
Number of generations	30, 100
Probability for function as next node	0.6
Optimization function	Correlation Coefficient
Parsimony pressure	0.1, 0.08, 0.05
Probability for random vs. guided crossover	0.5
Probability for mutation of terminals	0.3
Probability for mutation of functions	0.3

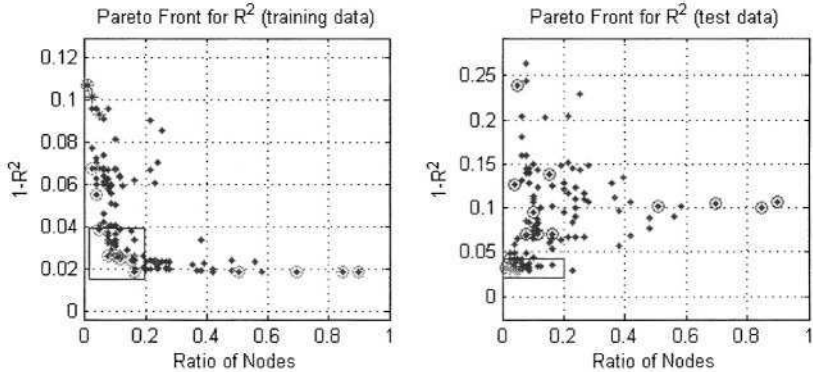


Fig. 2. Performance and Pareto-fronts for combined results of all GP runs

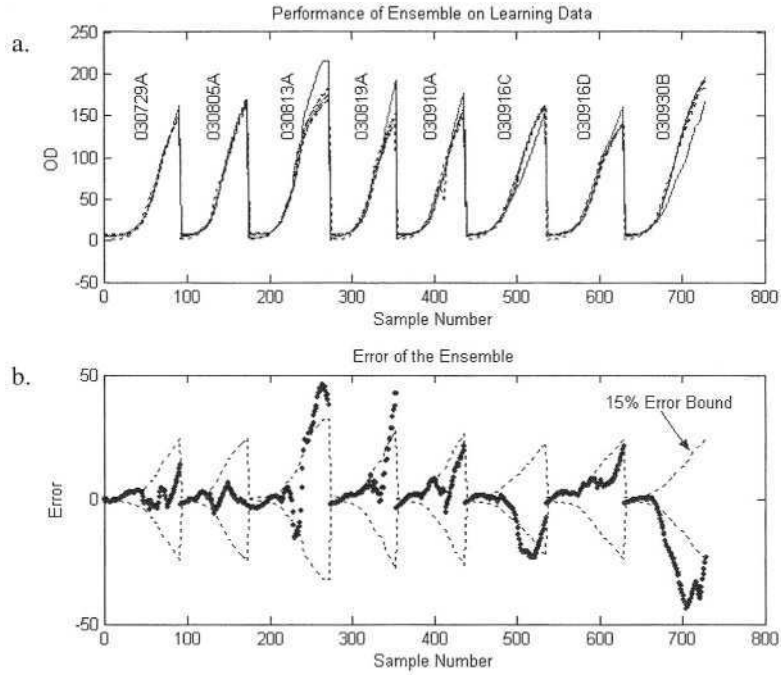


Fig. 3. Performance of the ensemble on the training data

The results from the different GP runs were combined and all duplicate models with equal symbolic strings were removed. In Fig. 2, the performance of the various models on the training and testing data are shown as well as the Pareto-fronts based on both data sets.

For the biomass inferential sensor, it was important that the models not be too complex and have an R^2–performance above 0.94. The models of interest were therefore those in the lower left corner of the Pareto-fronts of both training and test data. Furthermore, it was desirable that all the models have similar performance on the test data. The set of models that satisfy these conditions is within the compounding boxes of the training and test data, as shown in Fig. 2. The first set of the potential members of the ensemble were identified and inspected from the models that appear within both the boxes. Several models turned out to be duplicate models and were consequently removed. The final set of models for the ensemble is as follows:

$$f_1 = -116.471 + 329.8129\, e^{-e^{(-x_6)}}$$

$$f_2 = 7.4698 + 6.296 \frac{x_5 x_6}{{x_7}^2}$$

$$f_3 = 7.6537 + 343.9166\frac{x_2 x_6}{{x_7}^2}$$

$$f_4 = -37.7896 + 647.9714\, e^{-e^{\left(\exp(-x_6)+\exp\left(-\exp\left(x_7^{1/4}\right)\right)\right)}}$$

$$f_5 = -1.3671 + 0.12025\sqrt{x_6}\,(x_5 - x_7)$$

All of them are simple enough and provide significant diversity (in terms of selected inputs and nonlinear relationships) to allow a proper design of the ensemble of predictors. There is no direct physical or biological interpretation of the models but the experts agree that the most often used variable in all models x_6 is of critical importance to the cell growth.

4.3 Ensemble Performance

The prediction of the ensemble is defined as the average of the predictions of the five individual models obtained in the previous section. The accuracy requirements for the ensemble were to predict OD within 15% of the observed OD-level at the end of the growth phase. The performance of the ensemble on the training data used by GP can be seen in Fig. 3.

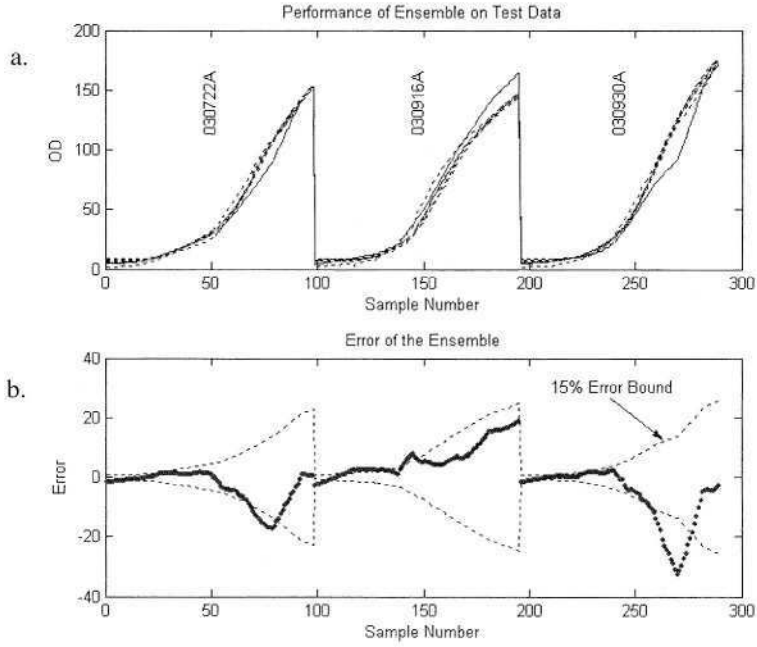


Fig. 4. Performance of the ensemble on the test data

In Fig. 3 (a) the OD-level is plotted against the sample number, which corresponds to the time from the beginning of the fermentation. The solid (blue) line indicates the observed OD-level. The dashed lines represent predictions of the individual models. The solid (red) line corresponds to the prediction of the ensemble. In Fig. 3(b) the residuals of the ensemble's prediction with respect to observed OD is shown. The 15% error bound is also shown. For the training data one sees that for three batches (030813A, 030819A and 030930B), the ensemble predicts outside the required accuracy. For batch 030813A it was known that the run was not consistent with the rest of the batches. However, this batch was added in order to increase the range of operating conditions captured in the training set.

The performance of the ensemble on the test data can be seen in Fig. 4. Again, in Fig. 4(a) the OD-level is plotted against the sample number for the observed data, the individual model predictions and the ensemble. In Fig. 4(b), the error with respect to the test data can be seen. We see that the performance of the ensemble at the end of the run for all the batches of the test data is within the required error bound.

5 Conclusions

In this paper we have shown a successful application of a novel type of biomass estimator based on GP. Furthermore, it is one of the rare applications of inferential sensors to batch processes.

We have also improved the model selection by using the Pareto-front approach. The main advantages of this approach are that the complexity of the generated models is taken into account and an unbiased decision is made. Furthermore, the number of interesting models to inspect manually is decreased to a manageable number.

Finally we have successfully implemented an ensemble-based inferential sensor of symbolic regression-generated functions for a notoriously difficult batch process.

References

1. Haykin, S.: Neural Networks: A Comprehensive Foundation, 2nd edn. Prentice Hall, New York (1998)
2. Qin, S.: Neural Networks for Intelligent Sensors and Control - Practical Issues and Some Solutions. In: Neural Systems for Control, Academic Press, New York (1996)
3. Lennox B. G. Montague, A. Frith, C. Gent, V. Bevan: Industrial Applications of Neural Networks – An Investigation, Journal of Process Control, **11**, (2001) 497 – 507
4. A. Kordon, G. Smits, A. Kalos, E. Jordaan: Robust Soft Sensor Development Using Genetic Programming, In: R. Leardi (ed): Nature-Inspired Methods in Chemometrics, Elsevier,Amsterdam (2003)
5. Koza, J.: Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, Cambridge, MA (1992)
6. Vapnik, V.: Statistical Learning Theory, Wiley, New York (1998)
7. Kordon A., G. Smits: Soft Sensor Development Using Genetic Programming, Proceedings of GECCO'2001, San Francisco, (2001) 1346–1351
8. Kalos A., A. Kordon, G. Smits, S. Werkmeister: Hybrid Model Development Methodology for Industrial Soft Sensors, Proc. of the ACC 2003, Denver, CO, (2003) 5417-5422

9. Kordon A.K, G.F. Smits, E. Jordaan, E. Rightor, Robust Soft Sensors Based on Integration of Genetic Programming, Analytical Neural Networks, and Support Vector Machines: Proceedings of WCCI 2002, Honolulu (2002) 896 – 901
10. Di Massimo C. et al.: Bioprocess Model Building Using Artificial Neural Networks, Bioprocess Engineering **7** (1991) 77-82
11. Tham M, A Morris, G Montague, P Lant: Soft Sensors For Process Estimation and Inferential Control, J. Process Control **1** (1991) 3-14
12. Willis M et al.: Solving Process Engineering Problems Using Artificial Neural Networks, In: Mc Ghel, M. Grimble, and P. Mowforth (eds): Knowledge-Based Systems for Industrial Control, Peter Peregrinus, London, (1992) 123-142
13. Cheruy A.: Software Sensors in Bioprocess Engineering, Journal of Biotechnology **52** (1997) 193-199
14. Hodge D., L. Simon, M. Karim: Data Driven Approaches to Modeling and Analysis of Bioprocesses: Some Industrial Examples, Proc. of the ACC2003, Denver (2003) 2062-2076
15. Sharkey A. (Editor): Combining Artificial Neural Nets, Springer-Verlag, London (1999)
16. Deb K.: Multi-Objective Optimization Using Evolutionary Algorithms, Wiley, Chichester UK (2001)
17. Bleuer S., M. Brack, L. Thiele, E. Zitzler: Multi-Objective Genetic Programming: Reducing Bloat bt Using SPEA-2, In Proceedings of CEC 2001, (2001) 536-543
18. Lampinen J. and A. Vehtari: Bayesian Approach for Neural Networks – Review and Case Studies, Neural Networks, **14** (2001 7-14

CellNet Co-Ev: Evolving Better Pattern Recognizers Using Competitive Co-evolution

Taras Kowaliw[1], Nawwaf Kharma[2], Chris Jensen[2], Hussein Moghnieh[2], and Jie Yao[2]

[1] Computer Science Dept., Concordia University, 1455 de Maisonneuve Blvd. W. Montreal, Quebec, Canada H3G 1M8
taras.kowaliw@utoronto.ca
[2] Electrical & Computer Eng. Dept., Concordia University, 1455 de Maisonneuve Blvd. W. Montreal, Quebec, Canada H3G 1M8
kharma@ece.Concordia.ca

Abstract. A model for the co-evolution of patterns and classifiers is presented. The CellNet system for generating binary classifiers is used as a base for experimentation. The CellNet system is extended to include a competitive co-evolutionary Genetic Algorithm, where patterns evolve as well as classifiers; This is facilitated by the addition of a set of topologically-invariant camouflage functions, through which images may disguise themselves. This allows for the creation of a larger and more varied image database, and also artificially increases the difficulty of the classification problem. Application to the CEDAR database of hand-written characters yields both an increase in reliability and an elimination of over-fitting relative to the original CellNet project.

1 Introduction and Review

Our goal is to build a system for evolving recognizers for arbitrary problem spaces utilizing as little expert input as possible. This is an ambitious goal, requiring much additional work. This paper describes a step toward that goal: Using the CellNet system [Kharma et al], we implement co-evolution utilizing a set of camouflage functions for the given pattern database. Our results will be shown to contribute towards this overall goal.

A Pattern Recognition System is almost always defined in terms of two functionalities: the description of patterns and the identification of those patterns. The first functionality is called feature extraction and the second, pattern classification. Since there are many types of patterns in this world ranging from the images of fruit flies to those of signatures and thumb prints, the focus of most research endeavours in pattern recognition has rightfully been directed towards the invention of features that can capture what is most distinctive about a pattern. This leads to two overlapping areas of research associated with features: feature selection and feature creation.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1090–1101, 2004.

1.1 Feature Selection

Siedlecki in [12] is the first paper to suggest the use of GA for feature selection. Ten years later, Kudo [6] demonstrates the superiority of GA over a number of traditional search & optimization methods, for sets with a large number of features. Vafaie [13] shows that a GA is better than Backward Elimination in some respects, including robustness. Guo [4] uses a GA for selecting the best set of inputs for a neural network used for fault diagnosis of nuclear power plant problems. Moser [8] proposes two new architectures of his own: VeGA, which uses parallelism to simultaneously evaluate different sets of features (on one computer), and DveGA, which utilizes an adaptive load balancing algorithm to use a network of heterogeneous computers to carry out the various evaluations. Shi [10] applies a GA to the selection of relevant features in hand-written Chinese character recognition. Fung [2] uses a GA to minimize the number of features required to achieve a minimum acceptable hit-rate, in signature recognition. Yeung [14] succeeds in using a GA in tuning four selectivity parameters of a Neocognitron, which is used to recognize images of hand-written Arabic Numerals.

1.2 Feature Creation

The first paper to speak about feature creation appears to be [Stentiford, 1985]. In his work, Stentiford uses evolution to create and combine a set of features (into a vector), which is compared to reference vectors, using nearest-neighbour classifiers. A single feature is a set of black- or white-expecting pixel locations. The results were good; the error rate, was 1.01%, and was obtained using 316 nearest-neighbour classifiers automatically generated by the evolutionary system. More recently, a group of researchers including Melanie Mitchell used evolution to create features for use in analyzing satellite and other remote-sensing captured images [1]. Specifically, the researchers defined a genetic programming scheme for evolving complete image processing algorithms for, for example, identifying areas of open water. Their system called GENIE successfully evolved algorithms that were able to identify the intended ground features with 98% accuracy.

1.3 Evolvable Pattern Recognition Systems

It was natural for researchers to move from feature creation to attempt to create whole pattern recognition systems. After all, the hardest part of successful pattern recognition is the selection/creation of the right set of features. Indeed, the first such systems seem to have focused on the evolution of features or feature complexes, which are useful for classification. Two systems described below do this. CellNet [5] blurs the line between feature selection and the construction of binary classifiers out of these features. HELPR [9] also evolves feature detectors, but the classification module is completely separate from the feature extraction module. Other differences exist, but, both attempts are the only systems, that we know of, that aim at using artificial evolu-

tion to synthesize complete recognition systems (though currently for different application domains), with minimum human intervention.

CellNet is an ongoing research project aiming at the production of an autonomous pattern recognition software system, for a large selection of pattern types. Ideally, a CellNet operator would need little to no specialized knowledge to operate the system. CellNet will eventually achieve this through the inclusion of methodologies for the simultaneous evolution of features and selectors (Cells and Nets); However, at present, a set of hard-coded features are used. Some interesting perspectives are offered as to how an evolvable feature creation framework may be structured; The most interesting of these suggestions is the use of a pre-processing routine deconstructing images using techniques inspired by Pre-Attentive Vision.

However, at present, CellNet is an evolvable (existing) feature selector and classifier synthesizer, which uses a specialized genetic operator, Merger (a.k.a Cooperative Co-Evolution). Merger is an operator, somewhat similar to that used in MessyGAs [3], designed to allow the algorithm to search a larger space than a regular GA. CellNet is cast as a general system, capable of self-adapting to many hand-written alphabets, currently having been applied to both Latin and Indian alphabets – as such, a more powerful engine is required than in preceding approaches.

HELPR is composed of a feature extraction module and a classification module. The classification system is not evolvable; only the feature extractor is. The feature extraction module is made of a set of feature detectors. Its input is the raw input pattern and its output is a feature vector. The current system is designed to handle signals (not visual patterns or patterns in general).

Each feature extractor has two stages: a transformation network followed by a capping mechanism. The transformation network utilizes a set of morphological operations with structuring elements in addition to a small number of arithmetic operations. As such, a transformation network (tn) represents a mixed expression which transforms an n-element digitized input signal into an n-element output signal. The purpose of the tn is to highlight those features of the input signal that are most distinctive of the target classes. On the other hand, the purpose of the capping mechanism is to transform the n-element signal coming out of the tn into a k-element array of scalars, where k is the number of target classes defined by the user. Ideally, every element will return a +1 for a single class and -1 for the rest.

Both HELPR and CellNet only evolve part of the system: other parts of the system are hard-coded. In addition, both systems have a large number of parameters that are manually set by an expert user. Finally, both systems only work on specific input signals/images; the promise of extension is there but not yet realized. This is why the following set of problems must be dealt with the:

1. Provision of a Feature Creation-Selection mechanism that is suited to a large class of classification problems. This is to eliminate the need for re-engineering, every time a different application domain is targeted.
2. Elimination of the many parameters that need to be set by the user before the system is run. These include items such as: probability distributions, probability values, size of population, number of detectors (feature extractors) and so on.

3. Testing any resulting system against a diverse set of pattern databases (and not just Radar signatures or Arabic Numerals), and doing so, without subjecting the system to the slightest amount of re-configuration.

This paper may be viewed as a step towards the realization of points 2 and 3; Co-Evolution is demonstrated to help eliminate the problem of over-fitting in the creation of a classifier, hence eliminating the need for an expert to manually determine the correct parameters; Additionally, the camouflage routines presented aid in the diversification of a given pattern database, allowing for greater variance in any given set of data.

2 Hunters and Prey

2.1 Hunters

CellNet hunters were first introduced in [5], a reader interested in their formal representation is urged to consult that source – our implementation is identical. However, given they are an entirely new and rather complicated representation for pattern recognizers, we offer an informal explanation of their structure, along with some illustrative examples.

2.1.1 Feature Functions

The basis on which Hunters are built is a set of normalized feature functions; The functions (all applied to thinned figures) used by CellNet CoEv are {parameterized histograms, central moments, Fourier descriptors, Zernike moments, normalized width (of a bounding box), normalized height, normalized length, number of terminators, numer of intersections}, as in [5]. However, for the purposes of explanation, we shall assume that our examples use only two: F_1 and F_2. This assumption is made for ease of visual representation of the Feature Space, and is easily generalized.

2.1.2 Hunters

A Hunter is a binary classifier – a structure which accepts an image as input, and outputs a classification. The fitness function which is used to drive the Genetic Algorithm determines which digit the agent classifies; For example, assuming our fitness function specifies that we are evolving Hunters to recognize the digit “ONE”, a Hunter which returns “yes” given an image will implicitly be stating that the image is a “ONE”, as opposed to “NOT-ONE”, i.e. any other digit. We will refer to these classes as the primary class and the non-primary class. Hence, a Hunter outputs a value from {PRIMARY, NON-PRIMARY, UNCERTAIN} when presented with an image.

A Hunter consists of Cells, organized in a Net. A Cell is a logical statement – it consists of the index of a Feature Function, along with bounds. Every cell is represented by the following format:

Feature Function F_i	Bound b_1	Bound b_2

Provided with an image I, a cell returns TRUE, if $b_1 < F_i(I) < b_2$, and FALSE otherwise.

A Net is an overall structure which organizes cells into a larger tri-valued logical statement. That is, a Net is a logical structure which combines the TRUE / FALSE values of its constituent Cells to return a value from {PRIMARY, NON-PRIMARY, UNCERTAIN}.

The structure of a Net is as follows: A Net is a tree, with a voting mechanism as its root node. At depth 1, there are a set of one or more Chromosomes.

Chromosomes consist of trees which begin with a Class Bit – this bit determines whether or not the Chromosome votes for "PRIMARY" or "NON-PRIMARY". Following the Class Bit is a tree of one or more Cells, connected into a logical structure by AND and NOT nodes. A Chromosome may be represented as a string as follows:

Class Bit C	[NOT]	$Cell_1$	AND	[NOT]	$Cell_2$	AND	...

Hence, a chromosome is a logical statement, which returns TRUE or FALSE. A Chromosome will return C if the logical statement returns TRUE – otherwise it will remain silent.

A Net is a collection of such Chromosomes, connected via a Vote mechanism. The Vote mechanism collects input from each Chromosome (although some Chromosomes will remain silent), consisting of a series of zero or more values of "PRIMARY" or "NON-PRIMARY". The Vote mechanism will tally the number of each, and output the majority, or "UNCERTAIN" in the case of no input or a tie.

For example, consider the following Example Agent, specified by the two Chromosomes chromosome 1 and chromosome2:

chromosome1:	C_1	$Cell_1$			
chromosome2:	C_2	$Cell_2$	AND	NOT	$Cell_3$

A Hunter is a Net – that is, a Hunter is an organized collection of one or more Cells, which when presented with an image, will return one of "PRIMARY", "NON-PRIMARY" or "UNCERTAIN". The complexity of a Hunter is the number of cells it contains, regardless of organization.

2.1.3 Examples of Hunters of Complexity One

The following are some examples of Hunters with complexity one, and interpretations in a two-dimensional feature space. Assume the Primary class is "ONE", and the non-primary class is "NOT-ONE".

Our first Hunter, A_1, consist of a single cell in a single chromosome – It is illustrated in Fig. 2.1.3.1. It is instructive to consider the Feature Space of all images on the basis of Feature F_1 and F_2 – every image maps to an <x,y> coordinate in this space, and hence may be drawn in a unit square. Agent A_1 may be viewed as a statement which partitions Feature Space into three disjoint sets – this is also illustrated in

Fig. 2.1.3.1. A second Hunter, A2, is illustrated in Fig. 2.1.3.2; This agent's partition of the same feature space is also illustrated.

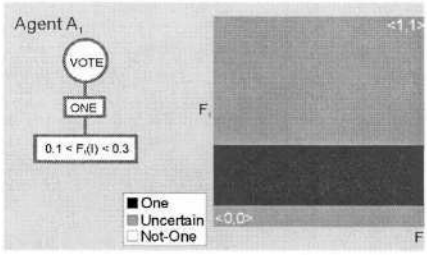


Fig. 2.1.3.1. Agent A_1 and its partition of Feature Space

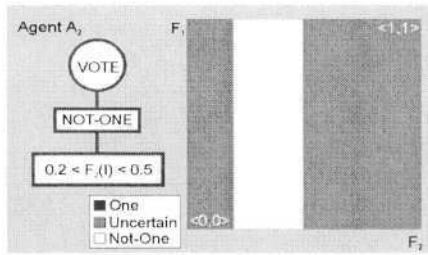


Fig. 2.1.3.2. Agent A_2 and its partition of Feature Space

2.1.4 Merger

Thus far, we have given examples only of Hunters with complexity one – this is the state of the CellNet Co-Ev system when initialized. What follows is a system of cooperative co-evolution which generates agents of increasing complexity.

Cooperative Co-evolution is achieved through the inclusion of a new Genetic Operator, augmenting the typical choices of Crossover, Mutation and Elitism. This new operator, Merger, serves to accept two Hunters and produce a single new Hunter of greater complexity; The complexity of the merged Hunter will be the sum of the complexities of the parents.

Merger operates at the level of Chromosomes – when merging two Hunters, Chromosomes are paired randomly, and merged either Horizontally or Vertically. Vertical Merger simply places both Chromosomes in parallel under the Vote mechanism – they are now in direct competition to determine the outcome of the Vote. Horizontal Merger, on the other hand, combines the two Chromosomes to produce a single and more complex Chromosome, where the two original Chromosomes are connected via a AND or AND-NOT connective. Hence, Horizontal Merger serves to refine a particular statement in the vote mechanism.

There are several decisions made when selecting which type of Merger is undertaken, and how connections are made in the Horizontal case – these decisions are under the control of two Affinity bits which are associated with each Chromosome. These Affinity bits ensure that all decisions are under genetic control, and may be selected for.

2.1.5 Horizontal and Vertical Merger

The Cooperative Co-evolution of Hunters, as realised through Merger is a technical process, more easily explained visually. We re-consider agents A_1 and A_2 of Section 3.1.3, considering their children through the Merger operator.

Consider the Horizontal Merger of Hunters A_1 and A_2 – here, we produce agent A_3 by combining the Chromosomes of A_1 and A_2 into one new one, linked via

an AND connective. As is visible in Fig. 2.1.5.1, Horizontal Merger may be viewed as the refinement of a partition created by two Chromosomes.

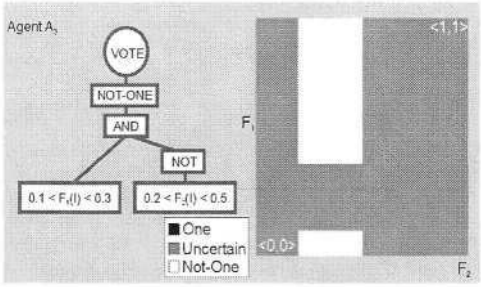


Fig. 2.1.5.1. Agent A_3 and its partition of Feature Space

In contrast, consider the Vertical Merger of these same two Hunters, producing agent A_4 – in this case, the Chromosomes are combined directly under the Vote mechanism. As shown in Fig. 2.1.5.2, Vertical Merger may be loosely be viewed as the union of the partitions generated by two Chromosomes.

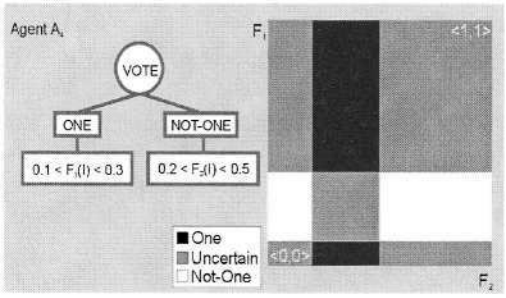


Fig. 2.1.5.2. Agent A4 and its partition of Feature Space

2.2 Prey

In a divergence from the methodology in [5], the second population in the CellNet Co-Ev system consist of Prey; Prey are primarily images, drawn from the CEDAR database of hand-written digits. In addition to the image, Prey disguise themselves via a set of camouflage functions, controlled genetically.

A Prey consists of a simple genome – an image index and a series of bits:

Image Index I	Bit b_1	Bit b_2	…	Bit b_k

The Image Index points to a particular image in the database. The Bits are Boolean parameters, indicating whether a particular camouflage function is to be applied or not.

Prey exist to be passed to Hunters for classification – prior to this, however, all camouflage functions specified by the series of bits in a Preys genome are applied

– hence, a Hunter views a transformed version of the original Image specified by the Image Index.

Camouflage functions used by the CellNet Co-Ev system consist of {Salt-Pepper, Scaling, Translation, Rotation}. These particular functions were chosen as they were topologically invariant (save Salt-Pepper, unlikely to have an effect). Parameters for the functions were chosen such that simultaneous application of all to an image would still yield a human-readable image.

Crossover for a Prey is single-point and fixed-length, occurring within the series of bits. Mutation flips a single bit in the series.

2.3 Agent Fitness and Competitive Co-evolution

The relationship between Hunters and Prey in the CellNet Co-Ev system is defined through their interaction in the Genetic Algorithm. Both populations are initially spawned randomly. For each generation, each agent is evaluated against the entirety of the opposing population. Explicitly:

Let h be a member of the Hunter population H, p a member of the Prey population P. For each generation, each Hunter h attempts to classify each Prey p – let

$$classAttempt(h,p)=\begin{cases}1;\ \text{h correctly classifies p}\\ 0.5;\ \text{h responds uncertain}\\ 0;\ \text{h incrorrectly classifies p}\end{cases} \quad (1)$$

Then the $\text{accuracy}_{\text{train}}$ of a hunter h is

$$accuracy_{train}(h)=\frac{1}{P}\sum_{p\in P} classAttempt(h,p) \quad (2)$$

Note that $\text{accuracy}_{\text{train}}$ differs from the traditional definition of the accuracy of a classifier. Later, when discussing the Validation Accuracy of a Hunter, we shall use a different measure on an independent validation set of (non-camouflaged) images, *im* in *I*.

$$classAttempt_{validation}(h,im)=\begin{cases}1;\ \text{h correctly classifies } im\\ 0;\ \text{otherwise}\end{cases} \quad (3)$$

Leading to the familiar measure of accuracy, which we shall call $\text{accuracy}_{\text{validation}}$.

$$accuracy_{validation}(h)=\frac{1}{I}\sum_{im\in I} classAttempt_{validation}(h,im) \quad (4)$$

Fitness of a hunter is defined as

$$fitness(h)=accuracy_{train}^{2}(h)-\alpha\cdot complexity(h) \quad (5)$$

where alpha is a system parameter designed to limit Hunter complexity, and complexity(h) is the number of cells in Hunter h.

In contrast, the fitness of a Prey p is defined as

$$fitness(p)=\frac{1}{H}\sum_{h\in H}(1-classAttempt(h,p)) \quad (6)$$

which is proportional to the inverse of fitness for Hunters.

As is evident from the relation between Hunters and Prey, the two populations exist in a state of Competitive Co-Evolution. The purpose of the addition of Camouflage functions to the database of images is two-fold:

1. It artificially increases the size of the database of images provided, by creating subtle changes in existing images. Hence, the system has a broader palette of training data.
2. The dynamics of the populations creates a more difficult problem for the Hunter population – not only do Hunters have to deal with an artificially large number of agents, it is precisely the transformation which they most often fail to recognize which will comprise the bulk of the next population of Prey. Hence, a Hunter population's weak points are emphasized.

3 Data and Analysis

The CellNet system was tested using the CEDAR database of handwritten numbers. Unlike previous experiments, only one pattern recognition task was attempted, although it is expected that scale-up to other handwritten languages, such as Indian or Chinese digits, is still possible. The system was configured to generate five binary hunters for each digit – these are labeled ***h.x.y,*** where x is the digit number, and y an index from 0 to 4. Each hunter was trained using a base set of 250 training images, and tested via an independent set of 150 validation images.

All hunters were developed using identical system parameters, although training and validation images for each run were chosen randomly from a pool of 1000 images. The hunters were placed in a genetic algorithm, using fitness-proportional selection and elitism. Parameters were a rate of crossover of 0.8, a rate of mutation of 0.02, a rate of merger of 0.02, and a rate of elitism of 0.1. The complexity penalty used in fitness was set to 0.0002. Prey were evaluated similarly – fitness-proportional selection, elitism of 0.1, crossover of 0.8 and mutation of 0.02.

Each run was executed for a maximum of 250 generations, outputting data regarding validation accuracy each 10 generations. A typical run may be seen in the evolution of the h.0 hunters, as illustrated in Fig. 3.1. Training and validation accuracies are very close, although validation accuracy tends to achieve slightly higher levels – this behaviour is typical of all digits. This is in contrast to previous experiments involving CellNet on hand-written characters, where overfitting of approximately 2-3% was reported consistently [5]. It is also noted that in initial generations of the runs, overfitting is common, as it can clearly be seen that the training plots are more accurate than the validation plots; This initial bonus, however, disappears by generation 60, where the validation plots overtake. However, also in contrast to previous experiments, complexity is vastly increased – in the case of the zero digit, mean complexity jumps from approximately 35 cells to approximately 65 cells, while the complexity of the most accurate agent jumps from 40 cells to seemingly random oscillations in the range of 50 cells to 350 cells. Fig. 3.2 shows the complexities of the most accurate agents and the mean for the h.0 runs.

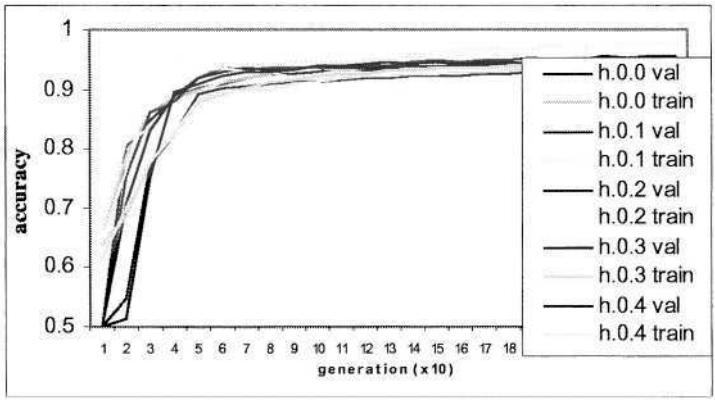


Fig. 3.1. Maximum training *(light lines)* and validation *(dark lines)* accuracies for the h.0 hunters.

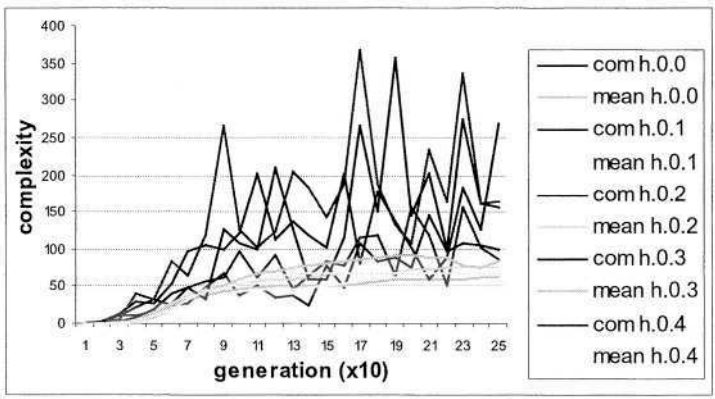


Fig. 3.2. Complexity of most fit agents *(dark lines),* and mean complexity *(light lines)* for the h.0 runs.

Table 3.1 shows the maximum training and validation accuracies for each binary hunter. The final columns compute the means for the validation and training accuracies for each class of hunter, and compare the difference. It is shown that the mean difference between training and validation data is –0.006, implying slight ***underfitting*** of the classifiers to the training data.

Finally, a series of experiments was undertaken regarding the classifications of the evolved binary classifiers; The scope of these experiments was the determination of the relative independence of the errors made by the classifiers when classifying images. Hence, our goal was a measure of the variance found between the errors of the hunters for each particular digit.

Table 3.1 Maximum Training and Validation Accuracies for the Binary Classifiers

	h.dig.0		h.dig.1		h.dig.2		h.dig.3		h.dig.4		mean		
dig	train	valid	train	valid	train	valid	train	valid	train	valid	train	valid	diff
0	.951	.955	.977	.955	.946	.945	.951	.944	.941	.946	.953	.949	+.004
1	.992	.981	.984	.971	.992	.982	.987	.977	.990	.984	.981	.979	+.002
2	.906	.906	.935	.932	.895	.904	.881	.896	.906	.920	.905	.912	-.007
3	.922	.910	.894	.919	.894	.910	.895	.908	.906	.926	.902	.915	-.013
4	.944	.941	.972	.967	.957	.962	.956	.962	.957	.952	.957	.957	+.000
5	.890	.919	.935	.937	.899	.919	.919	.922	.894	.914	.907	.922	-.015
6	.914	.941	.925	.940	.923	.953	.945	.917	.931	.923	.928	.935	-.007
7	.937	.934	.937	.954	.954	.954	.946	.940	.961	.939	.947	.944	+.003
8	.900	.914	.933	.918	.932	.939	.875	.905	.914	.931	.911	.921	-.010
9	.882	.911	.938	.944	.915	.917	.918	.926	.924	.939	.915	.927	-.012
											mean		-.006

Each hunter evaluated a set of 300 previously unseen images – a note was made for each error. Each classifier for any particular digit then had an associated error list of images, These lists were contrasted, computing the total number of errors (for all 5 hunters) and the percentage of the list shared by two or more hunters. These results are shown in Table 4.2; It is evident that there is much variance between the errors made by the various hunters.

Table 3.2. Percentage agreement in errors made by classifiers by digit

Digit	zero	one	two	three	four	five	six	seven	eight	nine	mean
number of errors	38	22	62	58	34	80	58	55	73	129	60.9
agree-ment	0.26	0.05	0.40	0.52	0.47	0.19	0.19	0.35	0.25	0.25	0.29

5 Conclusions

Relative to the original CellNet experiments, our augmented system performs admirably. The creation of binary classifiers is accomplished for each digit, showing little variance between results; This is contrasted against the original CellNet use of 30 runs to find a good classifier.

Additionally, the problem of over-fitting has been eliminated through the inclusion of competitive co-evolution. The inclusion of a genome for patterns and camouflage functions for diversification has resulted in a more difficult problem for the classifiers, increasing overall performance.

Finally, it has been demonstrated that although the reliability of the system's ability to generate classifiers has been improved, the produced classifiers' error sets are largely independent; This matter is crucial for the creation of multi-classifiers (combinations of the binary classifiers to form a single multiple-class recognizer), a step which a practitioner may wish to take. The independence of the error rates of the classifiers implies that several hunters for each class may be used in a bagging or bootstrapping technique, methods which are expected to improve the accuracy of the overall multi-classifier.

These results represent a significant step forward for the goal of an autonomous pattern recognizer: competitive co-evolution and camouflage is expected to aid in the problem of over-fitting and reliability without expert tuning, and also in the generation of a larger and more diverse data set.

References

1. Brumby, S.P., Theiler, J., Perkins, S.J., Harvey, N.R., Szymanski, J.J., Bloch J.J., Mitchell, M. Investigation of Image Feature Extraction by a Genetic Algorithm. In: Proc. SPIE 3812 (1999) 24—31.
2. Fung, G., Liu, J., & Lau, R. Feature Selection in Automatic Signature Verification Based on Genetic Algorithms, ICONIP'96, (1996) pp. 811-815
3. Goldberg, D. E., Korb, B. and Deb, K., Messy genetic algorithms: Motivation, analysis, and first results. In Complex Syst., vol. 3, no. 5, pp. 493-530, (1989).
4. Guo, Z., & Uhrig, R. Using Genetic Algorithms to Select Inputs for Neural Networks, Proceedings of COGANN'92. (1992) pp. 223-234
5. Kharma, N., Kowaliw, T., Clement, E., Jensen, C., Youssef, A., Yao, J. Project CellNet: Evolving an Autonomous Pattern Recognizer, Int. Journal of Pattern Recognition and Artificial Intelligence (In publication). http://www.ece.concordia.ca/~kharma/ResearchWeb/CellNet/cn.pdf
6. Kudo, M., & Sklansky, J. A Comparative Evaluation of Medium- and Large-Scale Features Selectors for Pattern Recognition in Kybernetika, V.34, No. 4, (1998) pp. 429-434.
7. Mitchell, M. An Introduction to Genetic Algorithms, MIT Press (1998)
8. Moser, A. A Distributed Vertical Genetic Algorithm for Feature Selection, ICDAR '99. pp1-9 (late submissions' supplement) (1999).
9. Rizki, M., Zmuda, M., Tamburino, L., Evolving pattern recognition systems, IEEE Transactions on Evolutionary Computation, volume 6, issue 6 , (2002) pp. 594-609
10. Shi, D., Shu, W., & Liu, H. Feature Selection for Handwritten Chinese Character Recognition Based on Genetic Algorithms, IEEE Int. Conference on Systems, Man, and Cybernetics, V.5, (1998) pp. 4201-6.
11. Siedlicki, W., & Sklansky, J. On Automatic Feature Selection, International Journal of Pattern Recognition, 2:197-220 (1998).
12. Stentiford, F. W. M. Automatic Feature Design for Optical Character Recognition Using an Evolutionary Search Procedure, IEEE Trans, on Pattern Analysis and Machine Intelligence. Vol. PAMI-7, No. 3, (1985) pp. 349- 355.
13. Vafaie, H., & De Jong, K., Robust Feature Selection Algorithms, Proceedings of the IEEE International Conference on Tools with Artificial Intelligence. (1993) pp. 356-363.
14. Yeung, D., Cheng, Y., Fong, H., & Chung, F., Neocognitron Based Handwriting Recognition System Performance Tuning Using Genetic Algorithms, IEEE Int. Conference on Systems, Man, and Cybernetics, V.5, (1998) pp. 4228-4233.

Evolutionary Ensemble for Stock Prediction

Yung-Keun Kwon and Byung-Ro Moon

School of Computer Science & Engineering, Seoul National University
Shilim-dong, Kwanak-gu, Seoul, 151-742 Korea
{kwon, moon}@soar.snu.ac.kr

Abstract. We propose a genetic ensemble of recurrent neural networks for stock prediction model. The genetic algorithm tunes neural networks in a two-dimensional and parallel framework. The ensemble makes the decision of buying or selling more conservative. It showed notable improvement on the average over not only the buy-and-hold strategy but also other traditional ensemble approaches.

1 Introduction

Stock prediction is a historically hot topic. There were a variety of studies on this topic [10] [14] [18] [19] [24] [26] [28] [29]. Early studies were mostly about deterministic measures to help the prediction [1] [11] [16] [20]. Since about the early nineties many approaches based on stochastic or heuristic models have been proposed. They include artificial neural networks [26] [29], decision trees [4], rule induction [7], Bayesian belief networks [30], evolutionary algorithms [14] [18], classifier systems [25], fuzzy sets [3] [28], and association rules [27]. Hybrid models combining a few approaches are also popular [10] [24].

Kwon and Moon [19] proposed neuro-genetic hybrids for the stock prediction and showed notable success. However, since they did not consider the cost of trading, the performance can be overestimated when the trading occur too often. Although the trading cost is not very high these days, their results more or less took benefit from the zero trading cost. To overcome the problem, we need a longer term trading model.

In this paper, we propose a new neuro-genetic approach. It is an evolutionary ensemble of recurrent neural networks which is an extended model of the system in [19]. An ensemble learning is to aggregate multiple subsystems to solve a complex problem and expect stable performance, and since genetic algorithms produce many solutions it is natural to make an ensemble model in GAs. The basic evolutionary ensemble is one that chooses some best solutions and makes a decision by the opinion of the majority or average output of the ensemble. In this paper, we apply a different ensemble model. It does not use the same members of ensemble for each test data. It dynamically chooses the members that perform well for a set of training data with similar to each test data. In this model, the ensemble consists of the members performing best for the days with the most similar contexts to today's.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1102–1113, 2004.

The rest of this paper is organized as follows. In Section 2, we explain the problem and present the objective. In Section 3, we describe our hybrid genetic algorithm for predicting the stock price. In Section 4, we provide our experimental results. Finally, conclusions are given in Section 5.

2 Objective

We have a database with years of daily trading data. Each record includes daily information which consists of the closing price, the highest price, the lowest price, and the trading volume. We name those at day t as $x(t)$, $x_h(t)$, $x_l(t)$, and $v(t)$, respectively. If we expect $x(t+1)$ is considerably higher than $x(t)$, we buy the stocks; if lower, we sell them; otherwise, we do not take any action. The problem is a kind of time-series data prediction as follows:

$$x(t+1) = f(x(t), x(t-1), x(t-2), \ldots).$$

As in [19], we transform the original time series to another that is more suitable for neural networks. Instead of $x(t+1)$, $\frac{x(t+1)-x(t)}{x(t)}$ is used as the target variable as follows:

$$\frac{x(t+1)-x(t)}{x(t)} = f(g_1, g_2, \ldots, g_m),$$

where g_k's $(k = 1, \ldots, m)$ are technical indicators or signals that were developed in previous studies.

We have four daily data, x, x_h, x_l, and v, but we do not use them for the input variables as they are. We utilize a number of technical indicators being used by financial experts such as moving average, golden-cross, dead-cross, relative strength index, and so on [15].

We describe some of them which were not considered in [19] in the following:

- Rate of change (ROC)
  - A ratio of price difference between the current price and the price a period of time ago.
  - $ROC = \frac{x(t)-x(t-K)}{x(t-K)}$
- Money flow index (MFI)
  - A momentum indicator that measures the strength of money flowing in and out of a security.
  - $MFI = MFI^+/MFI^-$
  - MFI^+, MFI^- : the sum of MF of days when TP is greater or smaller than that of the previous day over a period of time, respectively.
  - $MF = TP \times v$
  - $TP = (x_h + x_l + x)/3$
- Ease of movement (EOM)
  - An indicator that explains a relationship between price and volume.
  - $EOM = (MP(t) - MP(t-1))/BR$
  - $MP = (x_h + x_l)/2$
  - $BR = v/(x_h - x_l)$

$$X_1 = \frac{MA(t) - MA(t-1)}{MA(t-1)}$$
$$X_2 = \frac{MA_S(t) - MA_L(t)}{MA_L(t)}$$
$$X_3 = \frac{x(t) - x(t-1)}{x(t-1)}$$
$X_4 =$ the profit while the stock has risen or fallen continuously
$X_5 =$ # of days for which the stock has risen or fallen continuously
$$X_6 = \frac{x(t) - x_l(t)}{x_h(t) - x_l(t)}$$
$X_7 = ROC(t) - ROC(t-1)$
$$X_8 = \frac{x(t) - MA(t)}{MA(t)}$$
$X_9 =$ whether $MFI(t)$ crosses $MFI(t-1)$ or not
$X_{10} =$ whether $EOM(t)$ crosses zero value or not

Fig. 1. Some examples of input variables

In [19], 64 input variables were generated using the technical indicators. We add 11 input variables so totally generate 75 input variables. Figure 1 shows some representative variables including the added variables. In the figure, MA means a numerical average value of the stock prices over a period of time. MA_S and MA_L are short-term and long-term moving average, respectively. After generating the new variables, we normalize them by dividing by the maximum value of each variable. It helps the neural network to learn efficiently.

There can be a number of measures to evaluate the performance of the trading system. In our problem, we use almost the same measure as the one in [19]. The only difference in them is that we consider the transaction cost in this work. Figure 2 shows the investing strategy and change of property at day $t+1$ according to the signal at day t of the trading system. In the figure, C_t and S_t mean the cash and stock balances at day t $(t = 1, \ldots, N)$, respectively. We start with C, i.e, $C_1 = C$ and $S_1 = 0$. In the strategy, the constant B is the upper bound of stock trade per day and T is the transaction cost. The transaction cost was set to 0.3% in this work. We have the final property ratio P as follows:

$$P = \frac{C_N + S_N}{C_1 + S_1}.$$

3 Evolutionary Ensemble

3.1 Artificial Neural Networks

We use a recurrent neural network architecture which is a variant of Elman's network [6]. It consists of input, hidden, and output layers as shown in Figure 3.

if (signal is $SELL$) {
$C_{t+1} \leftarrow C_t + \min(B, S_t) \times (1 - T)$
$S_{t+1} \leftarrow S_t - \min(B, S_t)$
}
if (signal is BUY) {
$C_{t+1} \leftarrow C_t - \min(B, C_t)$
$S_{t+1} \leftarrow S_t + \min(B, C_t)$
}
$S_{t+1} \leftarrow S_t \times \frac{x_{t+1}}{x_t}$

Fig. 2. Investing strategy and change of the property

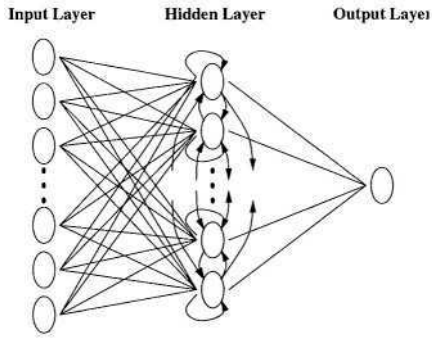


Fig. 3. The recurrent neural network architecture

Each hidden unit is connected to itself and also connected to all the other hidden units. The network is trained by a backpropagation-based algorithm.

It has 75 nodes in the input layer corresponding to the variables described in Section 2. Only one node exists in the output layer for $\frac{x(t+1)-x(t)}{x(t)}$.

3.2 Parallel Genetic Algorithm

We use a parallel GA to optimize the weights. It is a global single-population master-slave [2] and the structure is shown in Figure 4.

In this neuro-genetic hybrid approach, the fitness evaluation is dominant in running time. To evaluate an offspring (a network) the backpropagation-based algorithm trains the network with a set of training data. We distribute the load of evaluation to the clients (slaves) of a Linux cluster system. The main genetic parts locate in the server (master). When a new ANN is created by crossover and mutation, the GA passes it to one of the clients. When the evaluation is completed in the client, the result is sent back to the server. The server communicates with the clients in an asynchronous mode. This eliminates the need to synchronize every generation and it can maintain a high level of processor utilization, even if the slave processors operate at different speeds. This is possible because we use a steady-state GA which does not wait until a set of offspring

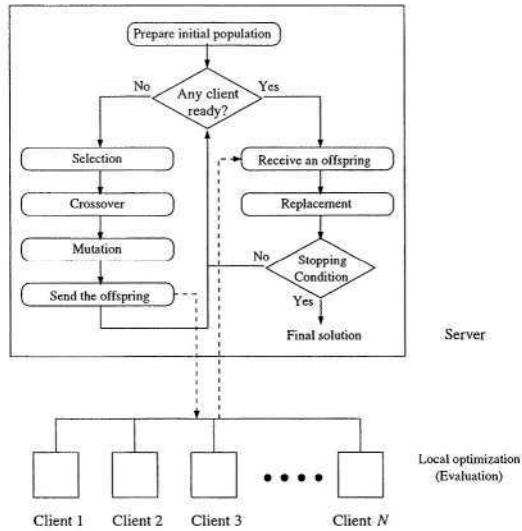


Fig. 4. The framework of the parallel genetic algorithm

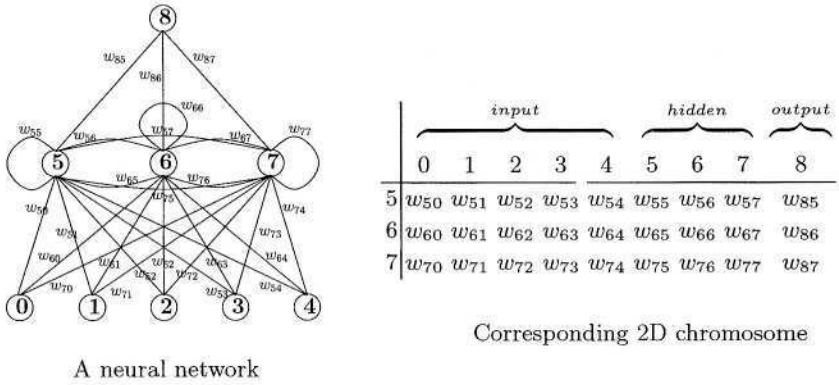


Fig. 5. Encoding in the GA

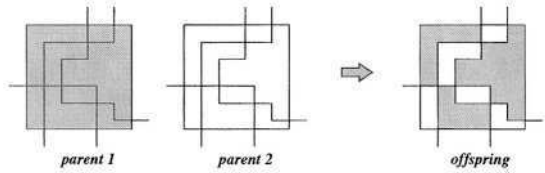


Fig. 6. An example of 2D geographical crossover

is generated. All these are achieved with the help of MPI (Message Passing Interface), a popular interface specification for programming distributed memory systems.

As shown in Figure 4, the process in the server is a parallel variant of traditional steady-state GA. In the following, we describe each part of the GA.

- *Representation:* We represent a chromosome by a two-dimensional weight matrix. In the matrix, each row corresponds to a hidden unit and each column corresponds to an input, hidden, or output unit. A chromosome is represented by a 2D matrix of $p \times (n+p+q)$ where n, p, and q are the numbers of input, hidden, output units, respectively. In this work, the matrix size is $20 \times (75+20+1)$. We should note that most GAs for ANN optimization used linear encodings [9] [21]. We take the 2D encoding suggested in [17]. Figure 5 shows an example neural network and the corresponding chromosome.
- *Selection, crossover, and mutation:* Roulette-wheel selection is used for parent selection. The offspring is produced by geographic 2D crossover [13]. It is known to create diverse new schemata and reflect well the geographical relationships among genes. It chooses a number of lines, divides the chromosomal domain into two equivalent classes, and alternately copies the genes from the two parents as shown in Figure 6. The mutation operator replaces each weight in the matrix with a probability 0.1. All these three operators are performed in the server.
- *Local optimization:* After crossover and mutation, the offspring undergoes local optimization by backpropagation which helps the GA fine-tune around local optima. The result of local optimization provides the quality of the offspring. As mentioned, it is performed in the client and the result is sent back to the server.
- *Replacement and stopping criterion:* The offspring first attempts to replace the more similar parent to it. If it fails, it attempts to replace the other parent and the most inferior member of the population in order. Replacement is done only when the offspring is better than the replacee. The GA stops if it does not find an improved solution for a fixed number of generations.

3.3 Instance-Based Ensemble Model

An ensemble learning is to aggregate multiple subsystems to solve a complex problem. A number of approaches have been developed for ensemble learning [5] [8] [12] [23] [31]. The method is based on the fact that a solution with the smallest training error does not necessarily guarantee the most generalized one.

It is usual to select the best individual as the final solution in genetic algorithms. However, there is room for improving the performance with the help of other individuals in the population. Evolutionary ensemble approaches select a subset of the population as ensemble. It consists of some best individuals or representative ones from the whole population. In the latter, a clustering algorithm such as k-means algorithm [22] is used and a representative solution for each cluster is selected. In this paper, we devised an instance-based ensemble which is different from traditional ensembles. Traditional ensemble models do not consider the relationship or difference between data; the members of the ensemble are chosen with respect to a fixed set of instances. The basic idea of

our instance-based ensemble is that it does not fix the members of ensemble but dynamically chooses the members that perform well for the days with similar contexts to today's. The instance-based ensembles are determined as follows:

- Obtain a set of NNs by the genetic algorithm described in Section 3.2.
- For each test day, select a set K of instances among the training days that are the most similar to the test day in terms of Euclidean distance.
- Construct a subset of NNs, as ensemble, that predict relatively well on the days in K.

The trading decision depends on the majority decision in the ensemble. The final decision is one of the following three signals: *BUY, SELL,* and *KEEP*. The signal *KEEP* means no action. In the course, we extract three opinions from the ensemble, D1, D2, and D3. D1 is about the direction of the price at the next day. D2 and D3 are about the directions of the price at the day after tomorrow in the cases that tomorrow's price goes up or down, respectively. The ensemble gives the signal of *BUY* when both D1 and D2 are *"up"* and gives the signal of *SELL* when both D1 and D3 are *"down."* Otherwise, it gives the signal of *KEEP*. By this strategy, the trading becomes more conservative and too light an action can be avoided.

4 Experimental Results

We tested our approaches with the stocks of 36 companies in NYSE and NASDAQ. We evaluated the performance for 11 years from 1992 to 2002. We got the entire data from YAHOO (http://quote.yahoo.com). The GA was trained with two consecutive years of data and validated with the third year's. The solution was tested with the fourth year's data. This process was shifted year by year. Thus, totally 14 years of data were used for this work.

Table 1 shows the experimental results. The values mean the final property ratio P defined in Section 2. *I-Ensemble* is the instance-based ensemble described in Section 3.3 and *Winner* is the version that uses the best solution. *A-Ensemble* is the version that selects a set of best NNs in the population and makes the decision from the average output of them, and *M-Ensemble* is the version that selects a set of best NNs in the same way as *A-Ensemble* and makes the decision by the the majority opinion of them. They are average results over 10 trials.

For quantitative comparison, we summarized the relative performance in Table 2. It represents the relative performance of each approach over the *buy-and-hold* strategy which buys the stock at the first day and holds it all through the year. Since there are 36 companies tested for 11 years, we have 394 cases except two cases with deficient data. In the table, *Up, Down,* and *NC* represent the situation of the stock market in each year. The *Up* and *Down* mean that the closing price has risen or fallen, respectively, over the year's starting price by 5% or more. *NC* means no notable difference. *Better* and *Worse* mean the number of cases where the P value of the learned strategy was at least 5% higher or lower than that of the *buy-and-hold,* respectively. The *I-Ensemble* performed better

Table 1. P values

Symbols	Strategies	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002
AA	*I-Ensemble*	1.206	1.093	1.091	1.211	1.239	1.181	1.081	1.577	0.797	1.190	0.674
	Winner	0.992	0.993	1.079	1.108	1.045	1.002	1.280	1.203	0.678	1.030	0.715
	A-Ensemble	1.141	1.115	1.282	1.272	1.305	1.214	1.279	1.359	0.797	1.155	0.754
	M-Ensemble	1.003	1.180	1.227	1.093	1.227	1.116	1.066	1.476	0.797	1.200	0.794
AXP	*I-Ensemble*	0.999	1.021	1.120	1.388	1.365	1.674	1.196	1.629	1.060	0.690	0.973
	Winner	0.995	1.050	0.983	1.330	1.245	1.583	1.147	1.355	1.084	0.690	0.807
	A-Ensemble	0.965	1.169	1.114	1.419	1.428	1.564	1.240	1.711	1.169	0.690	1.020
	M-Ensemble	1.026	1.030	1.114	1.456	1.237	1.599	1.176	1.558	1.034	0.690	0.883
AYP	*I-Ensemble*	1.074	1.156	0.825	1.325	1.041	1.065	1.060	0.765	1.330	0.850	0.192
	Winner	1.001	1.102	0.810	1.298	1.075	0.943	1.012	0.723	1.242	0.993	0.217
	A-Ensemble	1.070	1.113	0.825	1.384	1.039	0.991	1.062	0.740	1.282	1.050	0.157
	M-Ensemble	1.087	1.074	0.813	1.265	1.035	1.011	1.070	0.729	1.347	0.920	0.201
BA	*I-Ensemble*	0.783	1.102	1.114	1.668	1.300	0.950	0.713	1.260	1.018	0.614	0.864
	Winner	0.830	1.026	1.019	1.599	1.458	0.948	0.684	1.257	0.988	0.519	1.056
	A-Ensemble	0.860	1.012	0.983	1.709	1.300	0.973	0.720	1.329	1.097	0.615	0.856
	M-Ensemble	0.832	0.998	1.089	1.605	1.294	0.953	0.731	1.279	1.129	0.615	0.864
C	*I-Ensemble*	1.233	1.778	0.817	1.709	1.440	1.841	0.934	1.634	1.245	1.010	0.749
	Winner	0.824	1.672	0.817	1.211	1.387	1.698	0.960	1.861	1.367	0.979	0.775
	A-Ensemble	1.238	1.878	0.817	1.639	1.441	1.848	0.983	1.695	1.406	1.010	0.803
	M-Ensemble	1.210	1.833	0.817	1.435	1.441	1.773	0.980	1.636	1.317	1.010	0.748
CAT	*I-Ensemble*	1.330	1.327	1.244	1.181	1.487	1.135	0.967	1.228	0.883	1.009	0.928
	Winner	1.219	1.159	1.302	1.185	1.549	0.994	0.967	1.098	0.795	0.959	0.904
	A-Ensemble	1.060	1.644	1.231	1.280	1.245	1.351	0.967	1.499	0.829	1.115	0.962
	M-Ensemble	1.355	1.386	1.174	1.223	1.377	1.234	0.967	1.232	0.780	1.002	0.914
DD	*I-Ensemble*	1.037	1.031	1.053	1.270	1.330	1.349	1.097	1.269	0.900	0.966	1.369
	Winner	1.039	1.005	1.019	1.095	0.853	1.276	1.110	1.366	0.758	0.982	1.004
	A-Ensemble	1.038	0.989	0.996	1.233	1.075	1.577	1.205	1.331	1.005	0.996	1.334
	M-Ensemble	1.065	0.994	1.103	1.178	1.272	1.262	1.119	1.431	0.827	0.927	1.207
DELL	*I-Ensemble*	3.047	0.667	1.838	1.540	2.016	3.304	3.461	1.568	0.343	1.633	1.147
	Winner	3.440	0.917	1.992	1.717	2.799	1.882	3.461	1.331	0.341	1.259	0.871
	A-Ensemble	3.415	0.958	1.796	1.483	2.007	3.346	3.461	1.253	0.352	2.111	1.122
	M-Ensemble	2.235	0.513	1.713	1.660	2.013	3.346	3.461	1.455	0.344	1.868	1.424
DIS	*I-Ensemble*	1.286	1.024	1.113	1.106	1.164	1.462	0.877	1.002	1.024	0.828	1.125
	Winner	1.397	0.956	1.145	1.036	1.136	1.330	0.892	1.018	1.089	0.971	1.184
	A-Ensemble	1.345	1.009	1.024	0.931	1.107	1.149	0.843	1.050	1.073	0.908	1.301
	M-Ensemble	1.246	1.019	1.096	1.046	1.141	1.466	0.923	1.026	0.807	0.948	1.196
EK	*I-Ensemble*	0.891	1.173	1.078	1.548	1.193	0.842	1.140	0.978	0.655	0.777	1.232
	Winner	0.861	1.287	0.918	1.392	0.947	0.840	1.187	0.976	0.588	0.702	1.022
	A-Ensemble	0.889	1.187	1.076	1.227	1.149	0.810	1.096	1.023	0.694	0.761	1.097
	M-Ensemble	0.893	1.279	1.054	1.353	1.115	0.826	1.134	1.111	0.608	0.762	1.198
GE	*I-Ensemble*	1.119	1.179	1.011	1.312	1.376	1.448	1.350	1.600	0.954	0.934	1.152
	Winner	1.161	1.201	0.982	1.162	1.075	1.484	1.315	1.606	0.912	0.895	0.906
	A-Ensemble	1.116	1.141	1.006	1.234	1.328	1.429	1.359	1.316	0.982	0.874	0.989
	M-Ensemble	1.156	1.136	0.996	1.178	1.266	1.487	1.386	1.597	0.981	0.853	1.075
GM	*I-Ensemble*	1.065	1.244	0.755	1.283	0.957	1.309	1.150	1.304	0.840	0.963	0.993
	Winner	0.970	1.024	0.708	1.251	0.940	1.085	1.088	1.376	0.630	0.971	0.850
	A-Ensemble	0.969	1.030	0.754	1.251	0.995	1.526	1.284	1.419	0.809	1.324	1.040
	M-Ensemble	1.122	1.050	0.741	1.251	1.023	1.183	1.203	1.316	0.708	0.929	0.905
HD	*I-Ensemble*	1.437	0.787	1.167	0.874	1.036	1.211	1.425	1.661	0.697	1.110	0.485
	Winner	1.250	0.831	1.074	1.150	1.164	1.411	1.850	1.703	0.683	1.139	0.493
	A-Ensemble	1.437	0.825	1.214	1.049	1.037	1.210	1.640	1.549	0.615	1.106	0.516
	M-Ensemble	1.437	0.817	1.204	0.882	1.038	1.382	1.476	1.679	0.716	1.106	0.471
HON	*I-Ensemble*	1.347	1.303	0.886	1.287	1.378	1.238	1.103	1.138	0.817	0.799	0.672
	Winner	0.958	1.171	1.014	1.038	1.149	1.158	1.148	1.232	0.859	0.682	0.682
	A-Ensemble	1.289	1.289	0.990	1.189	1.365	1.312	1.103	1.164	0.878	0.809	0.642
	M-Ensemble	1.391	1.243	0.912	1.222	1.286	1.249	1.090	1.142	0.791	0.763	0.637
HWP	*I-Ensemble*	1.299	1.334	1.387	1.207	1.235	1.337	1.145	1.755	0.887	0.716	1.040
	Winner	1.004	1.203	1.305	1.276	1.334	1.353	1.215	1.293	0.724	0.716	0.807
	A-Ensemble	1.297	1.078	1.407	0.990	1.358	1.351	1.235	1.220	0.883	0.716	0.945
	M-Ensemble	1.577	1.133	1.389	1.193	1.320	1.390	1.088	2.013	0.806	0.716	0.933

Table 1. Continued

Symbols	Strategies	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002
IBM	*I-Ensemble*	0.566	1.010	0.994	1.150	1.738	1.378	1.644	1.276	0.734	1.459	0.841
	Winner	0.598	0.935	0.969	1.103	1.567	1.202	1.567	1.420	0.734	1.439	0.695
	A-Ensemble	0.622	1.150	0.988	1.063	1.759	1.378	1.668	1.341	0.734	1.543	0.836
	M-Ensemble	0.631	1.109	1.001	1.232	1.686	1.378	1.515	1.361	0.734	1.518	1.081
INTC	*I-Ensemble*	1.436	1.443	1.041	1.573	2.200	1.116	1.668	1.432	0.757	1.066	0.985
	Winner	1.068	1.358	1.078	1.564	2.015	0.985	1.623	1.084	0.678	1.008	0.975
	A-Ensemble	1.370	1.304	1.041	1.640	2.224	1.114	1.707	1.120	0.609	1.111	1.079
	M-Ensemble	1.142	1.275	1.041	1.353	2.228	1.114	1.729	1.425	0.710	1.113	1.000
IP	*I-Ensemble*	0.945	1.018	1.148	1.136	1.135	1.034	0.932	1.333	0.734	1.114	0.969
	Winner	0.910	0.967	1.130	1.164	1.222	0.938	0.880	1.024	0.831	1.057	0.882
	A-Ensemble	0.946	1.026	1.376	1.210	1.013	0.988	0.932	1.251	0.711	1.077	0.937
	M-Ensemble	0.946	1.049	1.223	1.163	1.087	1.013	0.932	1.124	0.718	1.145	1.053
JNJ	*I-Ensemble*	0.882	0.929	1.117	1.562	1.164	1.333	1.318	1.184	1.282	1.110	0.882
	Winner	0.888	0.933	0.977	1.556	1.016	1.388	1.431	1.436	1.288	1.043	0.912
	A-Ensemble	0.882	0.914	1.095	1.546	1.192	1.199	1.156	0.999	1.307	0.957	0.934
	M-Ensemble	0.882	0.950	1.164	1.599	1.227	1.311	1.299	1.110	1.226	1.217	0.927
JPM	*I-Ensemble*	0.944	0.976	0.814	1.442	1.173	1.207	0.939	1.173	1.220		
	Winner	0.942	0.977	0.815	1.384	1.099	1.011	0.987	1.164	1.359	N/A	N/A
	A-Ensemble	0.959	0.987	0.814	1.163	0.959	1.184	1.012	1.133	1.118		
	M-Ensemble	0.940	0.974	0.814	1.570	1.248	1.153	0.905	1.133	1.267		
KO	*I-Ensemble*	1.054	1.060	1.064	1.457	1.401	1.297	0.948	0.836	1.035	0.880	0.943
	Winner	1.040	1.060	1.032	1.503	1.329	1.222	0.899	0.834	0.816	0.838	0.921
	A-Ensemble	1.047	1.060	1.042	1.449	1.381	1.279	0.940	0.762	1.056	0.781	0.974
	M-Ensemble	1.037	1.060	1.073	1.413	1.386	1.251	1.029	0.839	1.061	0.993	0.997
MCD	*I-Ensemble*	1.248	1.160	1.048	1.642	1.024	1.070	1.719	1.029	0.845	0.943	0.619
	Winner	1.166	1.031	1.132	1.465	1.113	1.044	0.994	1.042	0.845	0.566	0.644
	A-Ensemble	1.248	1.220	1.051	1.348	0.992	1.117	1.246	1.030	0.845	0.791	0.592
	M-Ensemble	1.292	1.244	1.059	1.498	1.049	1.092	1.662	1.034	0.845	0.946	0.579
MMM	*I-Ensemble*	1.056	1.157	1.053	1.308	1.227	1.021	0.894	1.176	1.125	0.938	1.049
	Winner	1.050	1.022	1.089	1.176	1.071	1.034	0.894	0.996	1.283	0.808	1.003
	A-Ensemble	1.102	1.093	1.011	1.254	1.179	0.936	0.894	1.133	1.062	0.982	1.050
	M-Ensemble	1.092	1.100	1.073	1.210	1.150	0.961	0.894	1.221	1.235	0.884	1.050
MO	*I-Ensemble*	0.959	0.755	1.030	1.448	1.214	1.337	1.157	0.564	1.281	0.969	0.802
	Winner	0.959	0.726	0.777	1.221	1.198	1.300	1.139	0.595	1.494	0.905	0.827
	A-Ensemble	0.959	0.755	1.000	1.277	1.214	1.299	1.136	0.629	1.382	0.974	0.796
	M-Ensemble	0.959	0.756	0.985	1.328	1.214	1.383	1.151	0.652	1.396	0.921	0.839
MRK	*I-Ensemble*	0.790	0.800	1.206	1.636	1.243	1.316	1.310	0.958	1.333	0.817	0.999
	Winner	0.856	0.803	1.063	1.124	1.251	1.163	1.417	0.875	1.374	0.928	0.970
	A-Ensemble	0.791	0.812	0.986	1.680	1.243	1.220	1.245	0.905	1.265	1.065	1.058
	M-Ensemble	0.787	0.803	1.150	1.489	1.267	1.264	1.495	0.892	1.346	0.813	0.987
MSFT	*I-Ensemble*	1.130	0.965	1.241	1.350	1.819	1.594	2.012	1.653	0.372	1.540	1.258
	Winner	1.150	1.027	1.268	1.243	1.914	1.377	1.926	1.837	0.381	1.431	0.912
	A-Ensemble	1.115	0.998	1.150	1.345	1.819	1.142	1.829	1.653	0.394	1.762	1.238
	M-Ensemble	1.120	0.974	1.213	1.186	1.817	1.536	1.972	1.690	0.372	1.510	1.135
NMSB	*I-Ensemble*	4.926	4.054	1.291	3.055	1.561	1.963	1.230	1.108	1.344	1.315	1.249
	Winner	6.671	4.190	4.315	4.395	3.051	2.876	1.861	1.407	2.403	1.398	1.210
	A-Ensemble	7.167	4.481	4.447	4.235	2.772	3.180	2.067	1.540	2.166	1.442	1.112
	M-Ensemble	6.822	4.199	3.875	4.745	2.932	2.909	1.927	1.470	2.255	1.390	1.270
ORCL	*I-Ensemble*	1.597	1.821	1.504	1.519	1.687	0.855	1.820	1.431	0.893	0.557	0.820
	Winner	1.520	1.546	1.579	1.151	1.686	0.949	1.490	1.386	1.021	0.611	0.904
	A-Ensemble	0.978	2.019	1.515	1.652	1.797	0.869	1.373	1.346	0.946	0.530	0.981
	M-Ensemble	1.317	1.887	1.537	1.513	1.700	0.989	1.870	1.723	0.895	0.530	0.633
PG	*I-Ensemble*	1.180	1.056	1.133	1.362	1.258	1.446	1.112	1.285	0.722	1.025	1.021
	Winner	1.066	0.908	1.131	1.215	1.182	1.315	1.146	1.177	0.958	0.837	1.014
	A-Ensemble	1.115	1.027	1.208	1.370	1.265	1.483	1.112	1.358	0.681	1.049	1.036
	M-Ensemble	1.111	1.048	1.129	1.294	1.082	1.496	1.112	1.301	0.672	1.019	1.002
RYFL	*I-Ensemble*	5.341	8.777	13.169	5.974	6.145	2.837	1.417	1.925	2.137	3.067	1.569
	Winner	6.780	11.990	15.244	5.312	7.467	4.640	1.824	2.945	3.854	5.264	2.055
	A-Ensemble	9.031	12.164	15.898	6.318	7.424	4.747	1.554	3.690	3.952	4.807	2.033
	M-Ensemble	9.035	12.399	13.660	6.269	6.834	4.348	1.550	2.679	3.678	5.111	1.789

Table 1. Continued

Symbols	Strategies	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002
SBC	*I-Ensemble*	1.115	1.217	0.979	1.280	0.901	1.432	1.603	1.056	1.077	0.956	0.729
	Winner	1.083	1.246	0.973	1.174	0.834	1.571	1.584	0.949	1.027	0.981	0.905
	A-Ensemble	1.184	1.301	0.979	1.167	0.901	1.432	1.404	1.151	0.926	0.792	0.783
	M-Ensemble	1.108	1.233	0.957	1.131	0.943	1.524	1.517	0.913	0.977	1.104	0.737
SUNW	*I-Ensemble*	1.176	0.913	1.193	1.445	1.241	1.595	1.754	3.269	0.672	0.514	0.918
	Winner	1.072	0.595	1.158	1.499	1.680	2.084	1.022	3.101	0.614	0.574	1.073
	A-Ensemble	1.350	0.995	1.146	1.108	1.366	1.722	1.858	3.398	0.671	0.505	1.083
	M-Ensemble	1.501	0.863	1.261	1.858	1.340	1.686	1.643	3.277	0.665	0.492	0.864
T	*I-Ensemble*	0.970	1.014	0.950	1.180	1.012	1.123	1.311	1.179	0.351	1.246	0.935
	Winner	0.972	1.044	0.950	0.968	0.847	0.949	1.190	0.825	0.440	0.786	1.003
	A-Ensemble	0.931	0.990	0.950	1.347	0.891	1.126	1.324	1.067	0.345	1.599	1.196
	M-Ensemble	0.937	0.932	0.950	1.102	0.965	1.287	1.296	1.073	0.342	1.238	1.001
UTX	*I-Ensemble*	1.034	1.139	1.079	1.474	1.346	1.120	1.627	1.173	1.204	0.812	0.995
	Winner	0.989	1.082	1.134	1.300	1.321	1.107	1.350	1.362	1.222	0.692	0.825
	A-Ensemble	0.920	1.007	1.048	1.225	1.400	1.110	1.461	1.157	1.204	0.744	0.847
	M-Ensemble	0.942	1.008	1.178	1.358	1.347	1.131	1.524	1.140	1.204	0.734	1.009
WMT	*I-Ensemble*	1.063	0.811	0.809	1.140	1.062	1.258	1.316	1.824	0.843	1.126	0.942
	Winner	1.072	0.801	0.723	1.092	0.947	1.147	1.180	1.755	0.875	0.998	0.963
	A-Ensemble	1.063	0.811	0.833	1.114	1.048	1.401	1.089	1.857	0.932	1.108	0.937
	M-Ensemble	1.063	0.811	0.782	1.159	1.098	1.220	1.148	1.834	0.806	1.143	0.961
XOM	*I-Ensemble*	1.118	1.119	0.951	1.114	1.244	1.303	1.207	1.110	1.454	0.939	0.922
	Winner	1.178	1.088	1.020	1.175	1.273	1.296	1.203	1.101	1.431	0.986	1.022
	A-Ensemble	1.130	1.108	1.024	1.039	1.220	1.380	1.294	1.075	1.460	1.063	0.963
	M-Ensemble	1.143	1.095	1.007	1.178	1.275	1.263	1.178	1.079	1.372	0.916	0.960

Table 2. Relative performance over *buy-and-hold* Strategy

(1) *I-Ensemble*

	Better	*Worse*	*Even*	Total
Up	45	60	130	235
Down	53	3	56	112
NC	14	3	30	47
Total	**112**	**66**	**216**	394

(2) *Winner*

	Better	*Worse*	*Even*	Total
Up	45	125	65	235
Down	57	16	39	112
NC	14	12	21	47
Total	**116**	**153**	**125**	394

(3) *A-Ensemble*

	Better	*Worse*	*Even*	Total
Up	56	87	92	235
Down	58	9	45	112
NC	15	3	29	47
Total	**129**	**99**	**166**	394

(4) *M-Ensemble*

	Better	*Worse*	*Even*	Total
Up	47	68	120	235
Down	42	7	63	112
NC	15	7	25	47
Total	**104**	**82**	**208**	394

than the *buy-and-hold* in 119 cases, worse in 60 cases, and comparable in 216 cases. *Winner* uses the best NN and it is an approach with no ensemble. We note that *Winner* is the same model as the one in [19] except that it was evaluated under consideration of the transaction cost in this work. It did not show good performance when considering the transaction cost, primarily due to too often trades. *I-Ensemble* showed a significant performance improvement over not only *Winner* but also the other ensemble models on average.

5 Conclusion

In this paper, we proposed a GA-based evolutionary ensemble of recurrent neural networks for the stock trading. It showed significantly better performance than the "buy-and-hold" strategy and traditional ensemble models with a variety of companies on the data for the recent 11 years. In addition to the ensemble, we tried to make a conservative system not to trade too often. We have satisfiable profits after considering the transaction cost.

In the experiments, the proposed GA predicted better in some companies than in others. It implies that this work can be useful in portfolio optimization. Future study will include finding the stock trading strategy combined with portfolio. In addition, we believe this approach is not just restricted to the stock market.

Acknowledgment. This work was supported by Brain Korea 21 Project. The ICT at Seoul National University provided research facilities for this study.

References

1. W. Blau. *Momentum, Direction, and Divergence.* John Wiley & Sons, 1995.
2. E. Cantu-Paz. A survey of parallel genetic algorithms. *Calculateurs Parallels,* 10(2):141-171, 1998.
3. O. Castillo and P. Melin. Simulation and forecasting complex financial time series using neural networks and fuzzy logic. In *Proceedings of IEEE Conference on Systems, Man, and Cybernetics,* pages 2664–2669, 2001.
4. Y. M. Chae, S. H. Ho, K. W. Cho, D. H. Lee, and S. H. Ji. Data mining approach to policy analysis in a health insurance domain. *International Journal of Medical Informatics,* 62(2):103–111, 2001.
5. H. Drucker, C. Cortes, L. D. Jackel, Y. LeCun, and V. Vapnik. Boosting and other ensemble methods. *Neural Computation,* 6:1289–1301, 1994.
6. J. L. Elman. Finding structure in time. *Cognitive Science,* 14:179–211, 1990.
7. J. A. Gentry, M. J. Shaw, A. C. Tessmer, and D. T. Whitford. Using inductive learning to predict bankruptcy. *Journal of Organizational Computing and Electronic Commerce,* 12(l):39–57, 2002.
8. L. K. Hansen and P. Salamon. Neural network ensembles. 12:993–1001, 1990.
9. S. A. Harp, T. Samad, and A. Guha. Towards the genetic synthesis of neural networks. In *International Conference on Genetic Algorithm,* pages 360–369, 1989.
10. P. G. Harrald and M. Kamstra. Evolving artificial neural networks to combine financial forecasts. *IEEE Transactions on Evolutionary Computation,* 1(1):40–52, 1997.
11. F. Hochheimer. *Computerized Trading Techniques.* Merrill Lynch Commodities, 1982.
12. R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural Computation,* 3:79–87, 1991.
13. A. B. Kahng and B. R. Moon. Toward more powerful recombinations. In *International Conference on Genetic Algorithms,* pages 96–103, 1995.
14. M. A. Kanoudan. Genetic programming prediction of stock prices. *Computational Economics,* 16:207–236, 2000.

15. P. J. Kaufman. *Trading Systems and Methods.* John Wiley & Sons, 1998.
16. J. F. Kepka. Trading with ARIMA forecasts. *Technical Analysis of Stocks & Commodities,* 3:142–144, 1985.
17. J. H. Kim and B. R. Moon. Neuron reordering for better neuro-genetic hybrids. In *Proceedings of the Genetic and Evolutionary Computation Conference,* pages 407–414, 2002.
18. K. J. Kim. Genetic algorithms approach to feature discretization in artificial neural networks for the prediction of stock price index. *Expert Systems with Applications,* 19(2):125–132, 2000.
19. Y. K. Kwon and B. R. Moon. Daily stock prediction using neuro-genetic hybrids. In *Proceedings of the Genetic and Evolutionary Computation Conference,* pages 2203–2214, 2003.
20. P. E. Lafferty. The end point moving average. *Technical Analysis of Stocks & Commodities,* 13:413–417, 1995.
21. C. T. Lin and C. P. Jou. Controlling chaos by ga-based reinforcement learning neural network. *IEEE Transactions on Neural Networks,* 10(4):846–869, 1999.
22. J. MacQueen. *Some methods for classification and analysis of multivariate observation.* Berkely, University of California Press, 1967.
23. N. J. Nilsson. *Learning Machines: Foundations of Trainable pattern-classifying systems.* New Yourk: McGraw Hill, 1965.
24. K. N. Pantazopoulos, L. H. Tsoukalas, N. G. Bourbakis, Brün, and E. N. Houstis. Financial prediction and trading strategies using neurofuzzy approaches. *IEEE Transactions on Systems, Man, and Cybernetics–Part:B,* 28(4):520–531, 1998.
25. S. Schulenburg and P. Ross. Explorations in LCS models of stock trading. In *Advances in Learning Classifier Systems,* pages 151–180, 2001.
26. P. Tiňo, C. Schittenkopf, and G. Dorffner. Financial volatility trading using recurrent neural networks. *IEEE Transactions on Neural Networks,* 12:865–874, 2001.
27. R. Veliev, A. Rubinov, and A. Stranieri. The use of an association rules matrix for economic modelling. In *International conference on neural information processing,* pages 836–841, 1999.
28. Y. F. Wang. Predicting stock price using fuzzy gray prediction system. *Expert Systems with Applications,* 22(1):33–38, 2002.
29. I. D. Wilson, S. D. Paris, J. A. Ware, and D. H. Jenkins. Residential property price time series forecasting with neural networks. *Knowledge-Based Systems,* 15(5) :335–341, 2002.
30. R. K. Wolfe. Turning point identification and Bayesian forecasting of a volatile time series. *Computers and Industrial Engineering,* 15:378–386, 1988.
31. X. Yao and Y. Liu. Making use of population information in evolutionary artificial neural networks. *IEEE Transactions on Systems, Man, and Cybernetics–Part:B,* 28:417–425, 1998.

Discovery of Human-Competitive Image Texture Feature Extraction Programs Using Genetic Programming

Brian Lam and Vic Ciesielski

School of Computer Science and Information Technology
RMIT University
GPO Box 2476V, Melbourne, Vic Australia 3001
{blam, vc}@cs.rmit.edu.au

Abstract. In this paper we show how genetic programming can be used to discover useful texture feature extraction algorithms. Grey level histograms of different textures are used as inputs to the evolved programs. One dimensional K-means clustering is applied to the outputs and the tightness of the clusters is used as the fitness measure. To test generality, textures from the Brodatz library were used in learning phase and the evolved features were used on classification problems based on the Vistex library. Using the evolved features gave a test accuracy of 74.8% while using Haralick features, the most commonly used method in texture classification, gave an accuracy of 75.5% on the same problem. Thus, the evolved features are competitive with those derived by human intuition and analysis. Furthermore, when the evolved features are combined with the Haralick features the accuracy increases to 83.2%, indicating that the evolved features are finding texture regularities not used in the Haralick approach.

1 Introduction

Human-competitive methods can be defined as automatically generated methods that perform equal or better than those derived by human intuition, that require simple input from humans in their generation and are general in that they can be readily applied to new problems in the same domain [1]. Koza et al [1] have identified a number of instances where genetic programming has produced human-competitive solutions in such domains as analog electrical circuits and controllers, synthesis of networks of chemical reactions and antenna design. In this paper we describe an approach to the evolution of texture features that are human-competitive.

Apart from size, shape and colour, texture is an important property used in image classification. Texture can be defined as a function of the spatial variation in pixel intensities [2]. Typically repetition of some basic pattern is involved. Examples of textures are given in figures 8 and 9. The usual approach to texture classification involves extracting texture features from two or more classes of

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1114–1125, 2004.

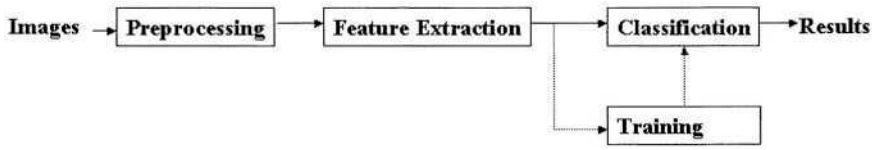


Fig. 1. Image Texture Classification

texture images to train a classifier. The features and classifier can then be used on new images. This process is shown in Figure 1. Currently, algorithms for texture features are developed by human intuition and analysis and there is a large number of approaches and theories. Texture classification is a significant economic problem and has been applied in many domains, for example, remote sensing [3], automatic inspection [4], medical image processing [5] and document processing [6].

There are two texture libraries that are used in most of the research in texture analysis - the Brodatz album and the Vistex data set. The Brodatz album consists of homogeneous categories of naturally occurring textures. The Vistex set consists of heterogeneous categories of texture images, that is, each class may have more than one type of texture. For example, the flower category may have flower images at three different resolutions, thus making the Vistex set more difficult to classify.

Our conjecture is that it may be possible to discover general feature extraction algorithms using an evolutionary search technique such as genetic programming if suitable fitness evaluation is provided. The overall process is shown in figure 2. Our specific research questions are:

1. What inputs, pixels or histograms, should be used as inputs to the genetic programs?
2. How can feature extraction algorithms be evolved from two textures?
3. Will features learnt from the Brodatz dataset generalize to the Vistex data set?
4. Have the evolved features detected any texture regularities not used in human developed methods.

In this paper we use the following terminology. A *learning set* is the set of images used by the evolutionary process to evolve feature extraction algorithms. These algorithms are then used on a different *training set* of images to get a nearest neighbour classifier which is evaluated against a different *test set.*

2 Related Work

2.1 Genetic Programming and Computer Vision

Genetic programming has been applied to a variety of image classification problems. The work so far can be grouped into three approaches. The first approach

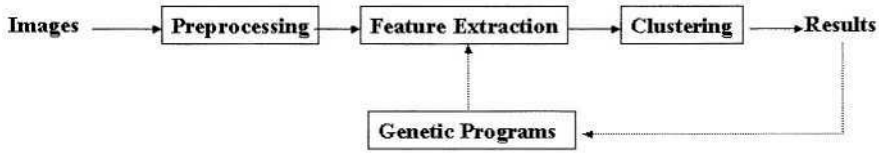


Fig. 2. Feature Extraction Discovery Using Genetic Programming

involves pre-processing images with low-level feature extraction algorithms followed by genetic programming designed to discover the classification rule. Tackett [7] used statistical features such as mean and standard deviation of grey levels within a window to classify military targets in infrared images. Poli [8] used pixel grey levels and moving averages of different size convolution masks to segment medical images. Agnelli [9] used statistical features to classify documents. Ross et al [10] used statistical features to classify microscopic images of minerals. Lin and Bhanu [11] used a co-evolutionary approach to build composite features from primitive features. This approach involves the steps of training and classification shown in Figure 1.

The second approach involves not using features and working directly from image pixels. Koza [12] discovered a detector for two alphabetic characters by moving a 3 x 3 matrix over two 6 x 4 binary pixel grids. Building on Koza's work, Andre [13] used moving templates to identify digits. Song et al [14] developed a pixel based approach to classify grey scale texture images. Martin [15] evolved image classification processes for robot navigation. This approach combines the feature extraction and classification steps shown in Figure 1.

The algorithms mentioned in the two previous approaches are problem-specific, that is, the derived algorithms only work with the types of images they were trained on. The learning process needs to be repeated for new types of images.

In the third approach, genetic programming is used to discover general algorithms that can be applied to new types of images without prior training. Harris [16] evolved edge detectors that performed well on real and synthetic data. This approach involves only the feature extraction step in Figure 1.

2.2 Conventional Texture Features

There has been considerable work on texture features since 1960 and many theoretical models have been proposed. A list of some of these is in table 1. The most well known texture feature extraction algorithm is the Grey Level Co-occurrence Matrix method developed by Haralick [17]. Assuming we are working with images that have 256 grey levels, this method involves first generating a grey level co-occurrence matrix with $256(i)$ columns and $256(j)$ rows. An entry in the matrix is the frequency of occurrence of pixels with grey level i and j level separated by a displacement d in a particular orientation. Most work uses a

displacement of 1. There are four principal orientations namely 0°,45°,90° and 135° so four matrices are generated. Thirteen second order statistical features are then calculated from each matrix. The features for the four principal orientations are then averaged giving a total of 13 Haralick features. Most new texture feature extraction methods are benchmarked against the GLCM method.

In our experiments we have followed the methodology of Wagner [18] who compared a large number of conventional methods on classification problems based on the Vistex dataset. This permits direct comparison of classification accuracy with the other methods. The image size used was 64 × 64.

3 Configuration of Genetic Programming

3.1 Inputs

Determining the most appropriate inputs to use for texture feature discovery is a major issue. Using grey levels of individual pixels will result in a very large number of terminals which can lead to memory and search problems. For example a 64×64 image would have 4096 terminals. We have determined empirically that 256 terminals is about the limit for reasonable performance in our system. For images larger than 16×16 some kind of aggregation of pixel values needs to be used. We have experimented with two forms of inputs – pixel values and histograms. In the case of pixel values, the grey level of each pixel is a terminal. Image size was 16 × 16 making a total of 256 terminals. In the case of histograms, the image size was 64×64 and the number of pixels at each grey level is a terminal in the program, making a total of 256 terminals. It is important to note that image texture has important spatial characteristics. These are preserved in the pixel representation, but lost in the histogram representation.

3.2 Functions

Originally we used the function set $\{+, -, *, /\}$. However we found that using just + gave feature extraction programs that were just as accurate as those using all four operators but were considerably easier to understand. Thus all subsequent work was carried out using just the + function.

3.3 Fitness Evaluation

A feature extraction algorithm is considered useful if the feature values result in high classification accuracy. This will occur if the feature values computed for each class are well separated in feature space. Thus, to evolve feature extraction algorithms, we need a way to implement the intuition that "the better the separation, the better the fitness". We have done this by computing the overlap between clusters generated by the K-means clustering algorithm. An example of this, for the case where there are two texture classes in the learning set is shown in figure 3. To get the data shown in the figure a program in the population

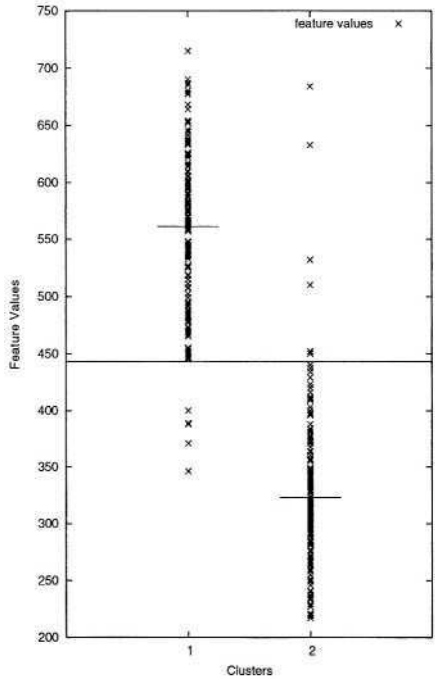


Fig. 3. Feature Space for Two Texture Classes

has been evaluated on a learning set of 160 images which consist of 80 examples of texture 1 from figure 8 and 80 examples from texture 2. The averages of the feature values for each class give cluster centroids at 561 and 323. The mid point of the two centroids is the cluster boundary, that is 443. There are 4 cluster1 feature values below the boundary and 6 cluster2 features above it, thus 10 points are incorrectly clustered. Equivalently, it can be considered that there are 10 errors.

3.4 Parameters

The RMIT-GP package [19] was modified to suit the problem. Clustering was performed using the Weka machine learning tool [20]. Default genetic programming parameters for the RMIT-GP package were used, namely a mutation rate of 0.28, a cross-over rate of 0.78 and an elitism rate of 0.02. Each run consisted of a population of 100 individuals evolved for 200 generations. The first generation of programs was generated randomly.

4 Learning from Two Classes of Textures

The goal of the experiments described in this section was to determine whether useful feature extraction algorithms could be learnt from two textures. The same

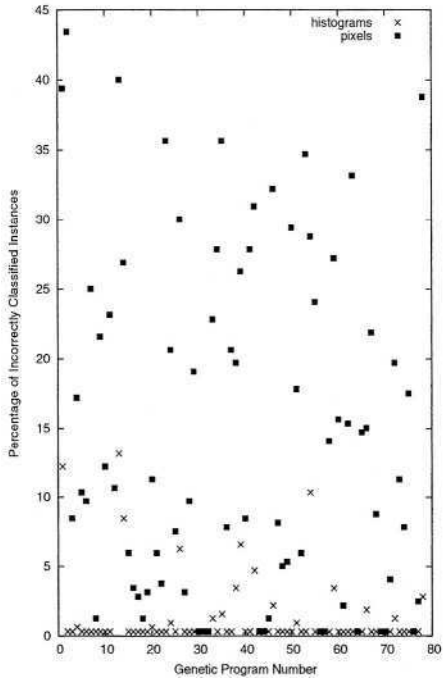


Fig. 4. Error for pixel and histogram approach

13 textures as selected in [18] were used. These are shown in figure 8. Each of the 78 different pairs were used as the learning set in 78 different runs of the genetic programming system, giving 78 features that could be used by a classifier.

4.1 Pixel Inputs

We first investigated the use of pixels of 16×16 sub images as inputs to the evolutionary process. Our intent was to preserve spatial relationships between pixels. However, the number of incorrectly clustered instances was very high and, despite considerable experimentation with the genetic algorithm parameters, we could not get this number down to reasonable levels. Figure 4 shows a comparison of the clustering results for the pixel approach and the histogram approach for each of the 78 combinations of Brodatz textures. The percentage of incorrectly clustered instances is shown for each program. For example, the error was 39% using pixels compared with 12% using histograms for programs which use Brodatz texture classes 1 and 2. It is clear that histogram inputs give much better clustering. In figure 5, the average and best fitness values are plotted for both approaches. The top two curves are those for the pixel approach and the bottom curves for histograms. The histogram approach converged faster to a better solution than the pixel approach.

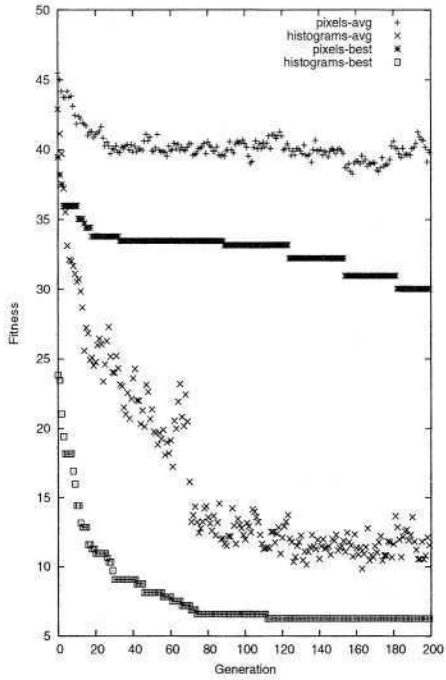


Fig. 5. Comparing runs on using pixels and histograms of image class 13 and class 2

While using pixels preserves spatial relationships, this approach has a number of disadvantages. First, the clustering is not as good. Second, the image size was limited to 16×16 pixels. Thus in our further experiments we have used the histogram representation. This has the further advantages that images of arbitrary size can be used as the histograms will always have 256 grey levels and are rotation invariant.

4.2 Histogram Inputs

To assess the quality and generality of the 78 evolved feature extraction algorithms we carried out three texture classification experiments using the standard train-a-classifier-and-test methodology. Experiment 1 was a 4 class problem involving Brodatz textures that were not in the learning set of 13. These are shown in figure 7. The training set consisted of 33 images and the test set 67. The 78 evolved feature extraction programs were applied to each image to generate a feature vector of 78 features. The feature vectors of the training set were then used to train a nearest neighbourhood classifier followed by calculation of the error rate on the feature vectors of the test set. The 78 evolved features gave a test accuracy of 100% while Haralick features gave a test accuracy of 95.5%.

Experiments 2 and 3 were carried out using the same methodology as [18], in which a large number of feature extraction algorithms were compared on a

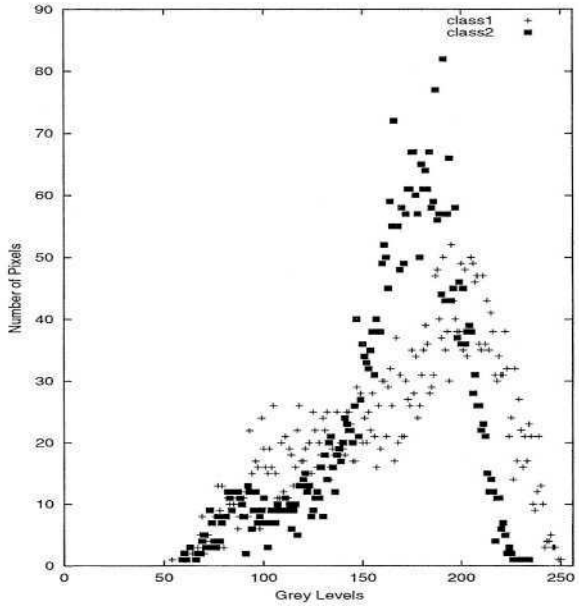


Fig. 6. Histograms of class1 and class2 Brodatz textures

number of texture classification problems. We did this in order to enable direct comparison with our evolved feature extraction algorithms. In experiment 2, the 13 Brodatz textures used in the learning set were used to give a 13 class problem. There were 32 images (64×64) per class in the training set and 64 images per class in the test set. The results are shown in the table 1. The evolved features outperform 5 of the 18 methods tested. However, this is not a true test of generality since the features are being tested on the same data they were evolved from.

In experiment 3, texture images from the Vistex set were used. In this problem there are 15 classes of textures (figure 9), each presented as a 64×64 image. There are 32 images per class in the training set and 64 images per class in the test set. The results are shown in the last column of table 1. Using the features from the evolved programs (GP Features) gave an accuracy of 74.8% which is

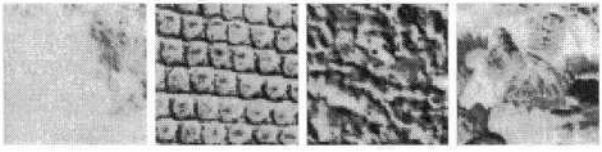


Fig. 7. Four Brodatz textures used for testing

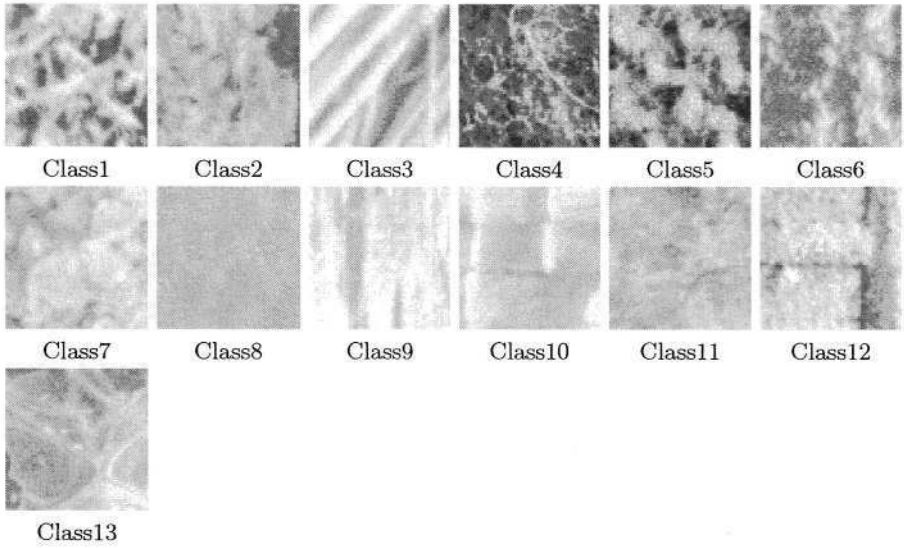


Fig. 8. Brodatz Texture Images

Table 1. Performance of Various Feature Extraction Algorithms All results, except for the last 3 are from [18].

Feature Set	Brodatz	Vistex
Unser	92.6%	81.4%
Galloway	84.7%	70.4%
Laine	92.4%	75.6%
Local features	61.1%	47.1%
Fractal(1)	62.6%	54.5%
Fractal(2)	66.5%	48.5%
Laws	89.7%	79.8%
Fourier coeff.	92.7%	80.1%
Chen	93.1%	84.5%
Sun & Wee	63.9%	58.4%
Pikaz & Averbuch	79.4%	74.4%
Gabor	92.2%	75.4%
Markov	83.1%	69.6%
Dapeng	85.8%	74.6%
Amadasun	83.4%	65.6%
Mao & Jain	86.3%	73.0%
Amelung	93.0%	82.1%
Haralick	86.1%	75.5%
GP Features	*81.5%*	*74.8%*
GP Features + Haralick	88.2%	83.2%

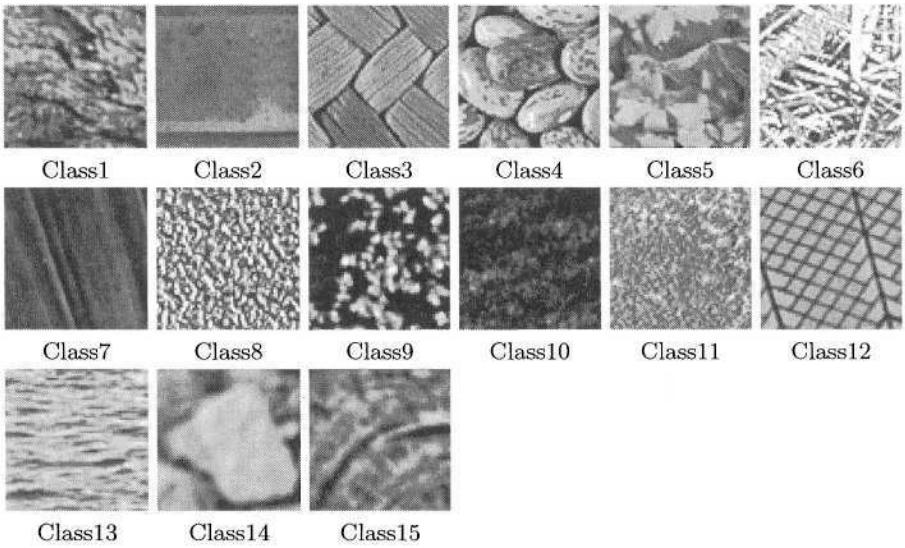


Fig. 9. Vistex Texture Images

very competitive with the Haralick accuracy of 75.5% and better than 8 of the 18 methods tested. The learning time is about 7 hours on a Xeon 2.8 Ghz computer.

5 Analysis of the Evolved Algorithms

Since + is the only function, all of the evolved algorithms are sums of the number of pixels at certain grey levels. For example, the feature extraction program evolved from class1 and class2 Brodatz textures is X109 + 2*X116 + 2*X117 + X126 + 2*X132+ X133 + 2*143 + X151 + X206 + X238 + 3*X242 + X254, where X*nnn* represents the value of the histogram at grey level *nnn.* If we examine the histograms of two images, shown in figure 6, we can see the program has make use of the points between grey level 100 and grey level 150 and those above grey level 200 where class1 is significantly different from class 2.

5.1 New Texture Regularities

A major question about the evolved feature extraction algorithms is whether any previously unknown texture regularities have been discovered. In general this is a very difficult question to answer, however, if the accuracy achieved by using Haralick and GP features together is significantly higher than the accuracy achieved by using Haralick features alone, then a reasonable case can be made that some new texture regularities, not captured by the Haralick approach, have been discovered. When the 78 GP features were added to the Haralick features, the accuracy on the Brodatz problem improved from 86.1% to 88.2% and the

accuracy on the Vistex problem improved from 75.5% to 83.2%. We can conclude that some new regularities have been discovered. However, we have not been able to determine the nature of these regularities.

6 Conclusions and Future Work

We have established that evolutionary search with genetic programming can be used to generate image texture feature extraction algorithms that are competitive with human developed algorithms on a difficult texture classification problem involving the Vistex library. Histogram inputs, rather than pixel values, should be used as inputs to the algorithms. The algorithms have captured some texture regularities, but it is very difficult to determine what they are.

Since the learning set contained only examples from the Brodatz set, but training and testing of the classifier using the evolved algorithms was performed on the Vistex set, there is some evidence that the feature extraction algorithms are general. However, more work needs to be done with other texture classification problems to verify this. We also plan to repeat the experiments with different selections from the Brodatz library and more than just pairs of textures in the learning sets.

The performance of the current feature extraction algorithms is limited by the fact that there is no spatial information in the histograms. We need to look at alternative representations that keep the number of inputs to a maximum of 256 but which capture more of the spatial information in a picture.

Acknowledgments. This work was funded by Australian Research Council SPIRT grant number C00107351 to RMIT University and Carlton and United Breweries. We acknowledge the support of Peter Rogers of Carlton and United Breweries. The evolutionary runs were performed on the Linux cluster hosted by the Victorian Partnership for Advanced Computing.

References

1. Koza JR, Keane MA, Streeter MJ, Mydlowec W, Yu J, Lanza G "Genetic Programming IV Routine Human-Competitive Machine Intelligence" Kluwer, 2003, 1-10
2. Tuceryan, M. and Jain, A.K."Texture Analysis" in Handbook of Pattern Recognition and Image processing, World Scientific, 1993, Chapter 2, 235-276
3. Rignot E, Kwok R, "Extraction of Textural Features in SAR images: Statistical Model and Sensitivity" in Proceedings of International Geoscience and Remote Sensing Symposium, Washing DC , 1990, 1979-1982
4. Jain AK, Farrokhnia, Alman DH, "Texture Analysis of Automotive Finishes" in Proceedings of SME Machine Vision Applications Conference, Detroit, 1990, 1-16
5. Chen CC, Daponte JS, Fox MD, "Fractal Feature Analysis and Classification in Medical Imaging" IEEE Transactions on Medical Imaging, 1989, 8, 133-142

6. Jain AK, Bhattacharjee SK, Chen Y "On Texture in Document Images" in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Champaign, Il, 1992
7. Tackkett WA , "Genetic Programming for Feature Discovery and Image Discrimination", in Proceedings of the 5th International Conference on genetic Algorithms, ICGA-93, University of Illinois at Urbana-Champaign, 17-21 July 1993, 303-309
8. Poli R, "Genetic Programming for Image Analysis" in Genetic Programming 1996: Proceedings of the First Annual Conference, MIT Press, 1996 363-368
9. Agnelli D, Bollini A, Lombardi L, "Image Classification: An Evolutionary Approach", in Pattern Recognition Letters 2002 ,Volume 23, 303-309.
10. Ross BJ, Fueten F and Yashkir DY, "Automatic Mineral Identification using Genetic Programming", in Technical Report CS-99-04, Brock University, December 1999
11. Yingqiang Lin and Bir Bhanu, "Learning Features for Object Recognition", GEC2003, LNCS2724, 2003, 2227-2239
12. Koza J R, "Simultaneous Discovery of Detectors and A Way of Using The Detectors Via Genetic Programming" in 1993 IEEE International Conference on Neural Networks, San Francesco, Piscataway, NJ: IEEE 1993. Volume 3, 1794 - 1801.
13. Andre D, "Automatically Defined Features: The Simultaneous Evolution of 2-dimensional Feature Detectors and An Algorithm for Using Them" in Advances in Genetic Programming, 1993 Chapter 23, 477-494
14. Song A, Loveard T and Ciesielski V, Towards Genetic Programming for Texture Classification; in AI 2001 Advances in Artificial Intelligence, Proceedings of the 14th Australian Joint Conference on Artificial Intelligence. Adelaide, December 2001, Springer-Verlag, Lecture Notes in Artificial Intelligence 2256, 461-472.
15. Martin C M, "Genetic Programming for Real World Robot Vision", in Proceedings of 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems, Lausanne, Switzerland, Sept-Oct 2002, 62-72.
16. Harris C and Buxton B, "Evolving Edge Detectors with Genetic Programming" in Genetic Programming 1996: Proceedings of the First Annual Conference, Stanford University, CA, USA, 28-31 July 1996, MIT Press, 309-315
17. Haralick M., Shanmugam K., and Distein I., "Texture Features for Image Classification", IEEE Transactions on Systems, Man, and Cybernetics, SMC-3(6):610-621
18. Wagner T , "Texture Analysis" in Handbook of Computer Vision and Applications, 1999, Volume 2, Chapter 12, 276-308
19. Dylan Mawhinney, RMIT-GP version 1.3.1, the RMIT University, 2002.
20. Len Trigg, Mark Hall and Eibe Frank, Weka Knowledge Explorer version 3-3-4, The University of Waikato 1999

Evolutionary Drug Scheduling Model for Cancer Chemotherapy

Yong Liang[1], Kwong-Sak Leung[1], and Tony Shu Kam Mok[2]

[1] Department of Computer Science & Engineering
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
{yliang, ksleung}@cse.cuhk.edu.hk
[2] Department of Clinical Oncology
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
tony@clo.cuhk.edu.hk

Abstract. This paper presents a modified optimal control model of drug scheduling in cancer chemotherapy and a new adaptive elitist-population based genetic algorithm (AEGA) to solve it. Working closely with an oncologist, we firstly modify the existing model, because the existing equation of the cumulative drug toxicity is not consistent with the clinical experience and the medicine knowledge. For exploring multiple efficient drug scheduling policies, we propose the novel variable representation – the cycle-wise representation; and adjust the elitist genetic search operators in the AEGA. The results obtained by the new model match well with the clinical treatment experience, and can provide much more realistic solutions than that by the previous model. Moreover, it has been shown that the evolutionary drug scheduling approach is simple and capable of solving complex cancer chemotherapy problems by adapting the suitable coding and the multimodal versions of EAs.

1 Introduction

An important target for cancer chemotherapy is to maximally kill tumor cells for a fixed treatment period. So drug scheduling is essential in cancer chemotherapy. Martin [6] have proposed the optimal drug scheduling model by the following differential equations:

$$\frac{dx_1}{dt} = -\lambda x_1 + k(x_2 - \beta)H(x_2 - \beta) \tag{1}$$

$$\frac{dx_2}{dt} = u - \gamma x_2 \tag{2}$$

$$\frac{dx_3}{dt} = x_2 \tag{3}$$

with the initial state $x^T(0) = [ln(100),\ 0,\ 0]$, the parameters $\lambda = 9.9 \times 10^{-4}$, $k = 8.4 \times 10^{-3}$, $\beta = 10$, $\gamma = 0.27$, $\eta = 0.4$, and:

$$H(x_2 - \beta) = \begin{cases} 1, & \text{if} \quad x_2 \geq \beta \\ 0, & \text{if} \quad x_2 \leq \beta \end{cases} \tag{4}$$

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1126–1137, 2004.

where x_1 is a transformed variable that is inversely related to the mass of the tumor. The tumor mass is given by $N = 10^{12} \times exp(-x_1)$ cells, and the initial tumor cell population is set at 10^{10} cells [6]. The variable x_2 is the drug concentration in the body in drug units *(D)* and x_3 is the cumulative drug toxicity in the body. Equation (1) describes the net change in the tumor cell population per unit time. The first term on the right-hand side of Equation (1) describes the increase in cells due to cell proliferation and the second term describes the decrease in cells due to the drug. The parameter λ is a positive constant related to the growth speed of the cancer cells, and k is the proportion of tumor cells killed per unit time per unit drug concentration which is assumed to be a positive constant. The implication of the function described in Equation (4) is that there is a threshold drug concentration level, β below which the number of the killed tumor cells is smaller than the number of the reproduced tumor cells, and the drug is not efficient. Equation (2) describes the net increase in the drug concentration at the cancer site. The variable u is the rate of delivery of the drug, and the half-life of the drug is $ln(2)/\gamma$, where γ is the biochemical character parameter of the drug. It is assumed that the drug is delivered by infusion, and there is an instantaneous mixing of the drug with plasma, as well as an immediate delivery of the drug to the cancer site. These assumptions represent approximations based on the relative amount of time. It takes for the aforementioned activities to occur with respect to the total amount of time over which the treatment is administered. Equation (3) relates the cumulative drug toxicity to the drug concentration, e.g., the cumulative effect is the integral of the drug concentration over the period of exposure.

The performance index [6] to be maximized is:

$$I = x_1(t_f) \tag{5}$$

where the final time $t_f = 84$ days. The control optimization is performed subject to constraints on the drug delivery: $u \geq 0$, and on the state variables: $x_2 \leq 50$, $x_3 \leq 2.1 \times 10^3$.

Cancer chemotherapy is a systemic treatment, so the action of the chemotherapeutic agent is not restricted to the tumor site. Any of the body organs are liable to injury. This is on contrast to the localized treatments, such as surgery or radiotherapy. Therefore, the constraints on the drug concentration x_2 and the cumulative drug toxicity x_3 are to ensure that the patient can tolerate the toxic side effects of the drug. Drug resistance is considered to be a significant factor in chemotherapeutic failure [3] [7] [9] and it has been shown that the drug resistant cells are likely to increase as the tumor burden increases [2]. In order to reduce the likelihood of the emergence of drug resistant cells, the tumor size is forced to reduce by at least 50% every 3 weeks, so that: $x_1(21) \geq ln(200)$, $x_1(42) \geq ln(400)$, $x_1(63) \geq ln(800)$.

Many researchers have applied different optimization methods to improve the results of the drug scheduling model [1] [5] [6] [7] [10] [11]. Among of them, Tan et al. [10] have proposed the "Paladin-Distributed Evolutionary Algorithms" approach to solve this problem and got the best-known solutions.

Through analyzing the experimental results from the existing model, there are two obvious unreasonable outcomes in the optimal drug scheduling policies: (*i*) unreasonable timing for the first treatment; and (*ii*) three point constraints cannot improve the efficiency of the cancer treatment. We analyze the reasons causing these problems and modify the existing model. The newly modified model is consistent with the clinical experience. The drug scheduling models are multimodal optimization problems and their feasible solution spaces consist of several discontinuous subregions. Here we select our novel genetic algorithm (GA), called an adaptive elitist-population based genetic algorithm (AEGA) [4], which is an efficient algorithm for multimodal optimization problems, to solve the modified drug scheduling model. Simulation results obtained show that our multimodal optimization algorithm AEGA produces excellent drug scheduling policies in cancer chemotherapy, which match well with results from clinical treatment experience. The power of the AEGA in obtaining multimodal solutions for this problem is also demonstrated.

This paper is organized as follows. Section 2 analyzes the problems that exist in the best-known solutions obtained by the existing drug scheduling model. Section 3 presents the newly modified model of the drug scheduling for cancer chemotherapy. Section 4 introduces the automation of the drug scheduling for cancer chemotherapy through the AEGA. The adaptation and modelling of the modified model in the AEGA are detailed including the new chromosome representation and genetic operators. The experimental results and discussion are given in Section 4. The paper conclusion is drawn in Section 5.

2 Analysis of the Experimental Results of the Existing Model

Fig.1-(a) and (b) show the best-known drug scheduling policies without and with three point constraints respectively, which are explored by Tan et al. [10] using distributed evolutionary computing software. Through observing the experimental results obtained by the existing drug scheduling model, there are two obvious unreasonable problems:

- unreasonable timing for the first treatment; and
- three point constraints cannot improve the efficiency of the cancer treatment.

In the 84 days treatment, the two best-known control policies for the drug are that it first gives drug on the 41*th* day under the model without the three point constraints, and on the 18*th* day under the model with the three point constraints. This is the first unreasonable problem because these drug policies obviously do not correspond with the clinical experience. In the clinical treatment, the general policy for efficiently reducing the tumor cells is that we should give a multi-dose of the drug rather than a normal dose on the first day. Since in the early days of the treatment, the patient's body has the strongest metabolism capability of the drug, and also the drug resistance of the tumor cells is the weakest at this time. So giving the multi-dose drug at this time can get the best

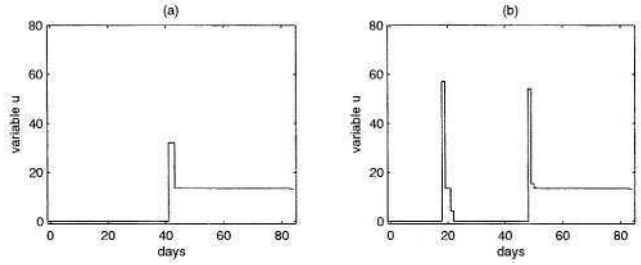


Fig. 1. The best-known solutions obtained by the existing drug scheduling model. (a): without the three point constrains; (b): with the three point constrains.

efficiency of the cancer treatment: to kill maximal tumor cells with minimal drug toxicity. According to this clinical experience, we can guess that an efficient drug scheduling policy should include the scheme that gives the multi-dose drug on the first day in the cancer treatment.

The second problem is that three point constraints cannot improve the efficiency of the cancer treatment. In the previous research works, the best-known performance index under the model with the three point constraints is 17.476 (corresponded to a final tumor size of $N = 2.57 \times 10^4$ cells). It is not better than the best-known performance index under the model without the three point constraints 17.993 (corresponded to a final tumor size of $N = 1.53 \times 10^4$ cells). This means that the three point constraints cannot improve the efficiency of the cancer treatment, but contrary to expectation, they reduce the overall efficiency of the drug chemotherapy. However, as described above, the aim of the three point constrains is to get more efficient drug scheduling policies. Because the drug resistance of the tumor cells increase with time and the emergence of drug resistant cells is thought to be a significant factor in chemotherapeutic failure [3] [7] [9]. In order to reduce the likelihood of the emergence of drug resistance cells, the tumor cells are forced to reduce by at least 50% every 3 weeks [5]. So these three point constraints should help to get more efficient drug scheduling policies for the cancer chemotherapy.

We believe the reason causing these problems is that the existing drug scheduling model is not consistent with the clinical experience. So we modify the existing model in the next section. The newly modified model can overcome the above two problems and its solutions are consistent with the clinical experience.

3 The Modified Model of the Drug Scheduling for Cancer Chemotherapy

The existing model [6] of cancer drug scheduling consists of three equations. Equation (1) can accurately describe the drug efficiency in the treatment period. Equation (2) also correctly describes the change process of the drug concentra-

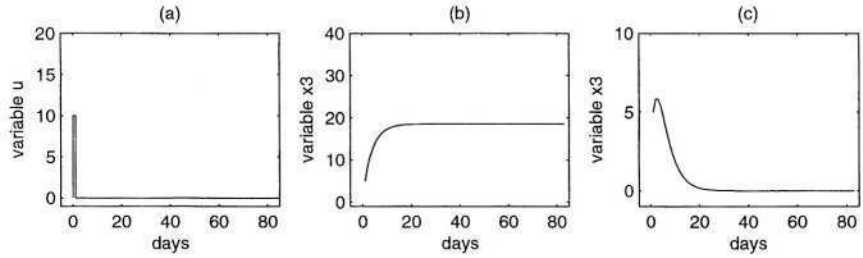


Fig. 2. The change processes of x_3 under the existing and modified models respectively.

tion in the body. But for Equation (3), the variable x_3 does not decrease in the whole cancer treatment, because on the right-side of Equation (3), the drug concentration x_2 is always not smaller than 0. Equation (3) may not be appropriate since it has not considered the metabolism process of the drug in the body. To clearly describe our analysis, we will use some experimental results to explain this problem.

In Fig.2-(a), we consider an additional test drug scheduling policy, which is giving $10D$ dose drug on the first day and not giving any drug in the later days. Fig.2-(b) shows the change process of the cumulative drug toxicity, x_3, in the fixed treatment period under the existing model. We can see that x_3 increases for the first few days due to the drug given on the first day, then x_3 almost remains constant in the later days. This means when the patient does not take the drug anymore, the drug toxicity still remains a constant value in the patient's body for the rest of the period. This does not correspond with the clinical experience. Because when the drug concentration x_2 decreased and tended to 0, generally the cumulative drug toxicity x_3 will be decreased through the metabolism and clearance by the liver and kidney in the body. To correct the above unreasonable model and accurately describe the change process of the cumulative drug toxicity x_3 in the body, with the help of an oncologist we have modified the third equation in the previous differential equation system as follows:

$$\frac{dx_3}{dt} = x_2 - \eta x_3 \tag{6}$$

Equation (6) describes the net change of the cumulative drug toxicity x_3 per unit time. In the right-hand side of this equation, the first term x_2 describes the increase of the cumulative drug toxicity x_3 due to the drug concentration x_2, and the second term $-\eta x_3$ describes the decrease in the drug toxicity due to the metabolism in the body. It is assumed that the metabolism speed of the drug is directly proportion to the amount of the current cumulative drug toxicity x_3. The parameter η is a positive constant related to the metabolism speed of the drug in the body.

Here we combine Equation (6) with Equations (1) and (2) to construct a new drug scheduling model. To demonstrate the difference of both models, we still use the additional test drug scheduling policy (Fig.2-(a)) to test the model and

analyze its results. The change processes of the variables x_1 and x_2 in the fixed treatment period (84 days) under the two models are the same. This means the new model also can correctly describe the change processes of the variables x_1 and x_2. Fig.2-(c) shows the change process of the cumulative drug toxicity x_3 in the fixed treatment period under the new model. The variable x_3 increases in the first few days after giving the drug on the first day, then it decreases to 0 along with the drug concentration x_2 decreasing to 0. The new drug scheduling model can correctly describe the metabolism process of the drug toxicity in the body.

Because we proposed the new Equation (6) to control the variable of the cumulative drug toxicity, x_3, the previous constraint $x_3 < 2100$ is not suitable. We use the two best-known solutions under the previous model without and with the three point constraints, to evaluate the new constraint about the variable x_3. Here we set the parameter $\eta = 0.4$ in Equation (6) and the maximal cumulative drug toxicity of these two best-known solutions are 99.851 and 99.999. These two maximal cumulative toxicities are too closed and all smaller than 100. According to this fact the new constraint for x_3 can be given as follows: $x_3 < 100$.

4 Evolutionary Drug Scheduling Model Via AEGA

In this section we use our adaptive elitist-population based genetic algorithm (AEGA), which is an efficient multimodal optimization algorithm [4], to implement the automation of the drug scheduling model for exploring multiply efficient drug scheduling policies. Why do we select a multimodal algorithm to solve this optimization problem? Because this problem includes some constraints, the feasible solution space consists of many subregions and these subregions are discontinuous. It is difficult for global optimization algorithms to search all subregions and explore a global optimum. We want to use a multimodal optimization algorithm to get multiple optima from all the subregions. On the other hand, in the clinical treatment, a doctor expects to select a different drug scheduling policy for a different patient. So we can use the multimodal optimization algorithm to solve the drug scheduling model, and get multiply efficient drug scheduling policies of the clinical treatment for the doctor to choose depending on the particular conditions of the patient under the treatment.

4.1 Variable Representation

For the drug scheduling problem in cancer chemotherapy as described in Section 1, there are 84 control variables to be optimized, which represent the dosage levels for the 84 days. The drug scheduling model is a high dimensional and multimodal optimization problem. Due to the large number of variables and fine accuracy involved, several representation schemes of variables in the evolutionary optimization were investigated. For example, some researchers used a pair-wise variable representation to reduce the complexity of variables. The information of dosage level and start-day are coded as variable pairs in such representation,

e.g., (30.5,28) meaning the starting of drug schedule from the $28th$ day with the dosage level of $30.5D$. However, the pair-wise variable representation is only useful when the given drug doses are not changed many times in the treatment period. For example, the best-known solution obtained by the previous model without the three point constraints is {(0,0); (32.1,41); (13.484,43); (13.21,83)}. The changes of the drug doses occur only 3 times in this solution. But in the clinical treatment, generally the drug schedule is a repeated policy (e.g., giving drug every two days). For such a case, the pair-wise variable representation will become more complex. Moreover, in EAs with the pair-wise variable representation, the existing evolutionary operators can only be implemented on the pair-wise variable representations, which consist of a constant number of variable pairs [10]. This will reduce the scheduling freedom and the efficiencies of EAs.

Here we propose a new variable representation—cycle-wise variable representation to accurately and efficiently describe the drug scheduling policy in a chromosome.

Definition 1: Due to the large number of variables and fine accuracy involved, the drug scheduling variable in the fixed treatment period defined by

$$\begin{aligned} \text{Variable Representation} &:= \{[C|D]^*|[D(DC)]^*\} \\ C &:= [c_i]^* \\ D &:= k_i \times \overline{d_i, \cdots, d_j} \end{aligned}$$

where

c_i is the drug dose on each day;
k_i is the number of cycles;
$\overline{d_i, \cdots, d_j}$ is the repetend;
$*$ represents the repetition of the structure that is located in the front square bracket, but the values can be different;

is called a cycle-wise variable representation of the drug scheduling model.

By Definition 1, the cycle-wise variable representation includes two parts: a front and a cyclic parts. The front part is $[C|D]^*$. It describes the drug doses in the initial treatment days. The cyclic part is $[D(DC)]^*$. It consists of the number of cycles k_i and the repetend $(\overline{d_i, \cdots, d_j})$. The cycle-wise variable representation is very suitable for the drug scheduling problem. Because in the first few days of the treatment period, the patient's body may not have adapted to the drug, but it is important to kill as much tumor cells as possible, the drug doses will be adjusted day by day. We use the front part to represent the drug doses in this initial period. Then when the patient's body gradually gets used to the drug, the drug administration schedule will follow a fixed cycle and a fixed dose pattern, which is suitably represented by the cycle-part.

Table 1. The multi-point crossover operator for the cycle-wise variable representation

Parents:
$\{94.92, \|_{r_1} \quad (8 \times \overline{10.8}), \quad \|_{r_2} \ (74 \times \overline{10.8})\}$
$\{136.98, \|_{r_1} \ (3 \times \overline{0}), 41.56, (3 \times \overline{0}), 35.85, \|_{r_2} \ (18 \times \overline{(3 \times \overline{0}), 39.58}), 0, 12.34\ \}$
Offspring:
$\{136.98, \|_{r_1} \quad (8 \times \overline{10.8}), \quad \|_{r_2} \ (74 \times \overline{10.8})\}$
$\{94.92, \|_{r_1} \ (3 \times \overline{0}), 41.56, (3 \times \overline{0}), 35.85, \|_{r_2} \ (74 \times \overline{10.8})\}$
$\{136.98, \|_{r_1} \ (3 \times \overline{0}), 41.56, (3 \times \overline{0}), 35.85, \|_{r_2} \ (74 \times \overline{10.8})\}$
$\{94.92, \|_{r_1} \quad (8 \times \overline{10.8}), \quad \|_{r_2} \ (18 \times \overline{(3 \times \overline{0}), 39.58}), 0, 12.34\ \}$
$\{136.98, \|_{r_1} \quad (8 \times \overline{10.8}), \quad \|_{r_2} \ (18 \times \overline{(3 \times \overline{0}), 39.58}), 0, 12.34\ \}$
$\{94.92, \|_{r_1} \ (3 \times \overline{0}), 41.56, (3 \times \overline{0}), 35.85, \|_{r_2} \ (18 \times \overline{(3 \times \overline{0}), 39.58}), 0, 12.34\ \}$

4.2 Elitist Crossover Operator for the Cycle-Wise Variable Representation

Here we combine the standard multi-point crossover operator with the adaptive elitist-population search techniques to construct the adaptive elitist-population based crossover operator for the cycle-wise variable representation. Let r_1 and r_2 be the crossover points in the front and the cyclic parts respectively of the two parents selected randomly from the population. The offspring are produced by taking all the combinations of the 3 segments (separated by r_1, r_2) of the parents' representations. In Table 1, the multi-point crossover operation generally can generate 6 offspring to improve the successful rate in the search process. Then two better solutions, which satisfy all the constraints, are selected from the parents and their offspring for the next generation.

Before we carry out the crossover operation, the adaptive elitist-population search technique incorporated in the crossover operator will delete the worse one from the two selected parents to reduce the population's redundancy, if they are located in the same optimal attraction. Of course, if this is carried out, no crossover will be performed. According to this principle, if the two parents have the same cyclic parts and similar front parts (smaller than the distance threshold σ_s), and the relative optimal directions of their front parts are face to face or one-way, the elitist operation will conserve the better one of these two parents for the next generation and delete the other one. Here we only use the front parts but not the whole cycle-wise representation to determine the relative optimal directions of both the parents to reduce the computation complexity of the algorithm.

4.3 Elitist Mutation Operator for the Cycle-Wise Variable Representation

In an elitist mutation operator, the basic mutation works as follows: for a randomly chosen position in the cycle-wise representation, replace its value with another randomly chosen value (not the same as the one to be replaced) with

Table 2. The one-point mutation operator for the cycle-wise variable representation

Parent:
$\{136.98, \boxed{(3 \times \overline{0})}, 41.56, (3 \times \overline{0}), 35.85, (18 \times \overline{(3 \times \overline{0}), 39.58})\};$
Offspring:
(1): $\{136.98, \boxed{(2 \times \overline{0}), 20.5}, 41.56, (3 \times \overline{0}), 35.85, (18 \times \overline{(3 \times \overline{0}), 39.58})\}$, or
(2): $\{136.98, (3 \times \overline{0}), 41.56, (3 \times \overline{0}), 35.85, (18 \times \overline{(3 \times \overline{0}), \boxed{27.64}})\}$, or
(3): $\{136.98, (3 \times \overline{0}), 41.56, (3 \times \overline{0}), 35.85, \boxed{(14 \times \overline{(4 \times \overline{0}), 39.58}), (2 \times \overline{0})}\}$.

certain mutation probability. For example, in Table 2, the fourth point of the parent is changed from 0 to 20.5 to generate its offspring (1); or the last value of the last cyclic part of the parent is changed form 39.58 to 27.64 to generate its offspring (2); or the number of cycles in the inner cycle of the last cyclic part is changed from 3 to 4 to generate its offspring (3).

The adaptive elitist-population search technique in mutation is that when the parent and its offspring are located in different optimal attractions, they are conserved together for the next generation to increase the population's diversity. For the cycle-wise representation, if the mutation operation is applied to its front part, and the relative optimal directions of the parent's and offspring's front parts is back to back, the elitist mutation operator will conserve the parent and its offspring together for the next generation. If the mutation operation is to apply the cyclic part, the elitist operation will not take place. As a general mutation operator, the elitist mutation operator conserves the best one of the parent and its offspring for the next generation.

4.4 The AEGA for the Drug Scheduling Model

In order to successfully explore multiple optimal solutions of the drug scheduling model, several rules for applying the AEGA are made as follows:

- Use the cycle-wise representation to keep the scheduling freedom and improve the efficiencies of EAs.
- Use the front part of the cycle-wise representation to check the dissimilarity of the individuals to reduce the computational complexity of the algorithm.
- Use the adaptive elitist-population search technique in the crossover operator to reduce the redundancy of the population.
- Use the adaptive elitist-population search technique in the mutation operator to increase the diversity of the population.
- Adaptively adjust the population size to optimally use the elitist individuals to explore multiple optima.

Following these rules, the AEGA for the drug scheduling model is implemented as follows:

1. Set $t = 0$ and initialize a chromosome population $P(t)$ (uniform random initialization within the bounds);
2. Evaluate $P(t)$ by using the fitness measure;
3. **While** (termination condition not satisfied) **Do**
   a) Elitist crossover operation to generate $P(t+1)$;
      i. check the dissimilarity of the randomly selected parents p_i and p_j;
      ii. **if** the parents p_i and p_j are similar, the elitist operation conserves the better one of them for the next generation; **else**, according to multi-point crossover operation, generates 6 offspring, and selects the better two from the parents and their offspring to the next generation;
   b) Elitist mutation operation to generate $P(t+1)$;
      i. according to the one-point mutation operation, generate the offspring c_i from the parent p_i;
      ii. **if** p_i and c_i are dissimilar, the elitist operation conserves p_i and c_i together for the next generation; **else**, selects the better one of them to the next generation;
4. Evaluate $P(t+1)$;
5. Stop if the termination condition is satisfied; otherwise, go to Step 3.

4.5 Experimental Results Under the New Model

The drug scheduling problem were simulated using the AEGA with the following parameters: initial population size=2000; maximal number of generations=20000; crossover rate=1.0; mutation rate=1.0 and the distance threshold σ_s=10. The drug scheduling model was simulated using numerical differentiation method of Runge-Kutta [8], with a small time interval of 0.1 day for good accuracy.

Automating the developed drug scheduling model via our multimodal optimization algorithm AEGA for 50 times can consistently obtain 6 most efficient drug scheduling policies. These results are listed in Table 3. For example, Fig. 3 and 4 show the control variable u, the best performance index x_1 (inversely related to the final mass of the tumor), the change processes of the drug concentration x_2 and the cumulative drug toxicity x_3 of the first and sixth optimal policies. The 6 best results all satisfy the three point constraints, and therefore it is not necessary to find the special solutions for the new model with the three point constraints separately.

The most efficient drug scheduling policies obtained by our new model are at least 8 times better than the best-known solution (corresponded to a final tumor size of $N = 1.53 \times 10^4$) under the previous model without the three point constraints and at least 13 times better than the best-known solution (corresponded to a final tumor size of $N = 2.57 \times 10^4$) under the previous model with the three point constraints. Since our modified dynamic model is more realistic, it has provided better drug administration scheduling solutions together with the AEGA approach used.

On the other hand, the multiple efficient drug scheduling policies under the new model match well with the clinical experience. In the clinical treatment, generally the drug scheduling policies include two kinds: continuous and repeated.

Table 3. The most efficient drug scheduling policies obtained by our new model

The most efficient drug scheduling policies	Tumor cells
(1): $\{94.92,(83 \times \overline{10.8})\}$	20
(2): $\{136.98, 0, 23.35, 0, 10.88, 0, 23.37, (38 \times \overline{0, 20.86}), 8.22\}$	34
(3): $\{136.98, (2 \times \overline{0}), 31.5, (2 \times \overline{0}), 24.5, (25 \times \overline{(2 \times \overline{0}), 30.41}), 0, 22.65\}$	76
(4): $\{136.98, (3 \times \overline{0}), 41.6, (3 \times \overline{0}), 35.9, (18 \times \overline{(3 \times \overline{0}), 39.6}), 0, 0, 25.6\}$	138
(5): $\{136.98, (4 \times \overline{0}), 50.1, (4 \times \overline{0}), 46.9, (14 \times \overline{(4 \times \overline{0}), 48.1}), 0, 21.3\}$	269
(6): $\{136.98, (13 \times \overline{(5 \times \overline{0}), 53.277}), 3 \times \overline{0}, 48.76\}$	1698

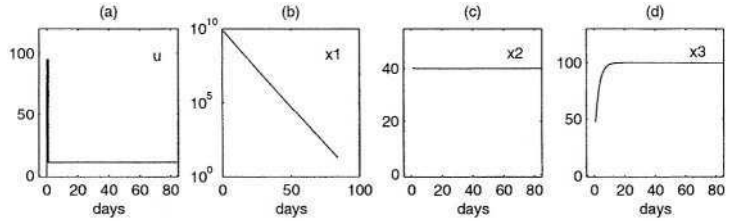


Fig. 3. The first efficient drug scheduling policy under our new model.

The drug scheduling policy (1) and the drug scheduling policies (2)-(6) represent these two kinds respectively. In some patients, the aim of treatment may be to reduce the tumor size with minimum toxicity and the drug scheduling policy (6) is suitable because its cumulative drug toxicity is low and often decreases to 60. For other patients, they may be cure despite higher toxicity, the drug scheduling policy (1) is suitable because this policy is most efficient but with high toxicity. So these multiple efficient drug scheduling policies obtained by the new model are more useful. According to the different conditions of the patients, the doctor can select the suitable drug scheduling policy to treat a cancer and get the best efficiency.

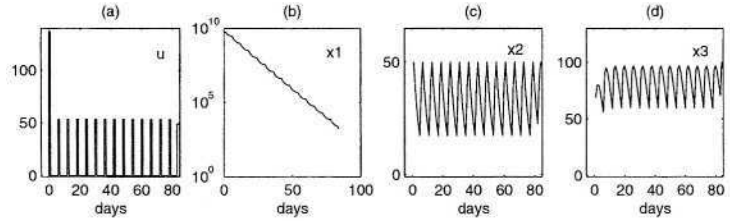


Fig. 4. The sixth efficient drug scheduling policy under our new model.

5 Conclusion

This paper has presented the modified optimal control model of drug scheduling in cancer chemotherapy and how to use our adaptive elitist-population based genetic algorithm (AEGA) to solve it. Working closely with an oncologist,we have firstly modified the existing model, because the existing equation, which control the cumulative drug toxicity x_3, is not consistent with the clinical experience and the medicine knowledge. For exploring multiple efficient drug scheduling policies, we have used our multimodal genetic algorithm (AEGA) to solve this complex multimodal optimization problem. We have proposed the novel variable representation – the cycle-wise representation, for the drug scheduling policy; and have adjusted the elitist genetic search operators in the AEGA to efficiently explore multiple efficient drug scheduling policies. The results obtained by the new model match well with the clinical treatment experience, and can provide much more realistic solutions than that by the previous model. Moreover, it has been shown that the evolutionary drug scheduling approach is simple and capable of solving complex cancer chemotherapy problems by adapting the suitable coding and the multimodal versions of EAs.

Acknowledgment. This research was partially supported by RGC Earmarked Grant 4212/01E of Hong Kong SAR and RGC Research Grant Direct Allocation of the Chinese University of Hong Kong.

References

1. Carrasco, E.F., Banga, J.R.: Dynamic Optimization of Batch Reactors Using Adaptive Stochastic Algorithms. Ind. Eng. Chem. Res. **36** (1997) 2252-2261
2. Coldie, J.H., Coldman, A.J.: A Mathematical Model for Relating the Drug Sensitivity of Tumors of Their Spontaneous Mutation Rate. Can. Treat. Rep. **63** (1978) 1727-1733
3. Crowther, D.: Rational Approach to The Chemotherapy of Human Malignant Disease-II. B. Med. J. **4** (1974) 216-218
4. Leung, K.S., Liang, Y: Genetic Algorithm with Adaptive Elitist-population Strategies for Multimodal Function Optimization. Proc. of Int. Conf. GECCO-2003. (2003) 1160-1171, and nomination for best paper award at GECCO-2003
5. Luus, R., Harting, F., Keil, F.J.: Optimal Drug Scheduling of Cancer Chemotherapy by Direct Search Optimization. Hung. J. Ian. Chen. **23** (1995) 55-58
6. Martin, R.B.: Optimal Control Drug Scheduling of Cancer Chemotherapy. Automatica. **28** (1992) 1113-1123
7. Murray, J.M.: Optimal Control of a Cancer Chemotherapy Problem with General Growth and Loss Functions. Math. Biosci. **38** (1990) 273-287
8. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical Recipes in C: he Art of Scientific Computing. 2nd edn. Cambridge University Press (1992)
9. Skipper, H.E.: Adjuvant Chemotherapy. Cancer. **41** (1978) 936-940
10. Tan, K.C., Khor, E.F., Cai, J., Heng, C.M., Lee, T.H.: Automating the Drug Scheduling of Cancer Chemotherapy via Evolutionary Computation. Artificial Intelligence in Medicine. **25** (2002) 169-185
11. Wheldon, T.E.: Mathematical Models in Cancer Research. Bristol. Adam Hilger. (1998)

An Island-Based GA Implementation for VLSI Standard-Cell Placement

Guangfa Lu and Shawki Areibi

School of Engineering
University of Guelph
Ontario, Canada, N1G 2W1
Tel: (519) 824-4120, X53819
Fax:(519) 836-0227
sareibi@uoguelph.ca

Abstract. Genetic algorithms require relatively large computation time to solve optimization problems, especially in VLSI CAD such as module placement. Therefore, island-based parallel GAs are used to speed up this procedure. The migration schemes that most researchers proposed in the past have migration near or after the demes converged [1,2]. However, for the placement of medium or large standard-cell circuits, the time required for convergence is extremely long, which makes the above migration schemes non practical. In this paper, we propose a novel migration scheme for synchronous island-based GA. Compared to the widely used ring topology that usually produces worse solutions at the beginning of the search but better solutions at later generations, the proposed migration scheme enables a parallel GA to outperform its serial version most of the time. Near linear speedup is obtained. We also implemented an asynchronous model that can achieve super-linear speedup.

1 Introduction

Physical design of VLSI circuits is a process of determining the location and connection of devices inside a chip. The main stages of the physical design phase include partitioning, placement, and routing [3]. Our work focuses on the placement of standard-cell in the physical design phase.

Given a circuit consisting of a set of modules, the standard-cell placement problem attempts to find a layout indicating the positions of the modules in parallel rows such that the total interconnection length, placement area, or some other placement metrics (eg. congestion, circuit delay) are minimized. As shown in Figure 1, a standard cell is a small block of predefined sub-circuit. Cells are restricted to having equal height, but variable width, and are placed in parallel rows that are separated by routing channels, as illustrated in Figure 2.

The VLSI standard cell placement is an NP-hard problem [4]. Various heuristic optimization techniques have been applied to this problem in the past. Genetic algorithms are proved to be able to produce high quality placement solutions for standard-cell circuits as competitive as that of other sophisticated algorithms such as Simulated Annealing and force-directed algorithms [5]. However,

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1138–1150, 2004.

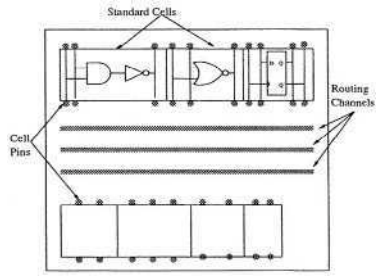


Fig. 1. Standard Cells

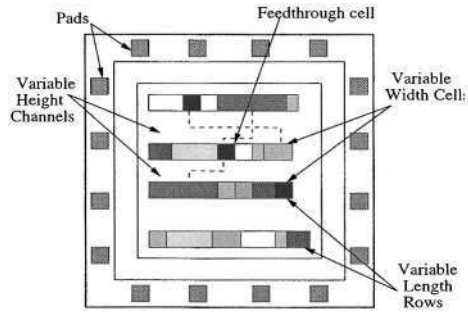


Fig. 2. Standard Cell Placement

the runtime of genetic algorithms axe relatively longer than other algorithms [6, 7] and are becoming less competitive in the real world. Therefore, researchers are seeking parallel GA implementation for better performance.

Island-based GAs are coarse grain parallel models, and can be easily mapped to existing distributed parallel systems like a cluster of networked workstations. Since a fairly good speedup can be easily achieved, the Island-based GA is becoming the most popular parallel method. In this method the whole population is divided into subpopulations (also known as demes) and distributed among multiprocessors. These semi-isolated subpopulations are executed as normal genetic algorithms, except that they would swap a few strings occasionally. By introducing migration, parallel island models have often been reported to display better search quality than that of a serial GA [8,9].

Migration is very important to search quality and parallel efficiency. Usually considered to be a good migration scheme, "Delay" migration schemes are algorithms where migration occurs after the demes are near convergence or completely converged. However, in standard-cell placement, this approach is not suitable since large circuits require extremely long time to converge. We propose a practical migration scheme for placement in this paper, and describe the successful synchronous implementation. Besides the novel migration scheme, our contribution also includes an implementation of an asynchronous model for placement that can achieve super-linear speedup.

The rest of the paper is organized as follows: Section 2 presents a brief overview of previous work on evolutionary based placement algorithms and the challenge of parallel genetic algorithms for standard-cell placement. In section 3, we present a novel migration scheme to meet the challenge and describe the successful synchronous parallel GA implementation for placement. In section 4, we describe an asynchronous parallel implementation that can achieve super-linear speedup. The experimental results are presented in section 5, and the paper concludes in section 6.

2 Background

In this section, we describe a serial genetic algorithm implementation for standard-cell placement and highlight the challenge of parallelizing GAs for such a problem.

There are three primary objectives in the placement problem: minimizing chip area, achieving routable designs, and improving circuit performance. Our work in this paper mainly focuses on the objective of minimizing chip area. Since minimizing the wire-length is approximately equivalent to minimizing the total chip area for the standard-cell layout style [10], the total cost of a placement layout can be estimated by the sum of wire length over all nets:

$$\phi(x,y) = \sum_{1 \le i < j \le N} w_{ij}[(x_i - x_j)^2 + (y_i - y_j)^2] \tag{1}$$

(x_i, y_i) denotes the location of cell i; w_{ij} is a non-negative weight of the edge connecting cell i and cell j. Equation (1) can be rewritten in matrix form as:

$$\phi(x,y) = \frac{1}{2}\mathbf{x}^T\mathbf{C}\mathbf{x} + \mathbf{d}_x^T\mathbf{x} + \frac{1}{2}\mathbf{y}^T\mathbf{C}\mathbf{y} + \mathbf{d}_y^T\mathbf{y} + \mathbf{t} \tag{2}$$

Vectors $\mathbf{x}$ and $\mathbf{y}$ denote the coordinates of the N movable cells; matrix $\mathbf{C}$ is the Hessian matrix; vectors $\mathbf{d}_x^T$ and $\mathbf{d}_y^T$ and the constant term $\mathbf{t}$ result from the contributions of the fixed cells. In the placement stage, this *Semi-perimeter method* is commonly used to approximately estimate the total wire-length.

Previously, [11] has developed a standard-cell placer called *SC3* for academic research, using various optimization techniques. A steady-state genetic algorithm(Figure 3) is one of the selectable algorithms that can be used to optimize initial placement solutions. In this algorithm, a standard-cell placement solution string was represented by a set of alleles (the number of alleles equal to the number of cells). Each allele indicates the cell index, the X- coordinates and row number of the cell. Figure 4a illustrates the string encoding of the standard-cell placement given in Figure 4b. Individuals are evaluated to determine their fitness through a scoring function F, using the *Semi-perimeter method:*

$$F = \frac{1}{\sum_{i=1}^{n} HPWL_i} \tag{3}$$

where $HPWL_i$ is the estimated wire-length of net i, and n is the number of nets. Thus $\sum_{i=1}^{n} HPWL_i$ is the sum of the half perimeter of the smallest bounding rectangle for each net. In the implementation, cell overlaps axe removed and row lengths are adjusted before evaluating the chromosome. Initial solutions can be constructed randomly, or by Cluster-Seed method. The GA starts its evolution by applying stochastic operators such as selection, crossover, and mutation to the individuals in each generations. The main drawback of genetic algorithms is the high computational demand, therefore we apply island-based genetic algorithms to speedup this procedure.

For island-based GAs, an appropriate migration scheme is critical. The methods of migration schemes determined by convergence are very popular. [12] proposed algorithms where migration begins only after the subpopulations are near

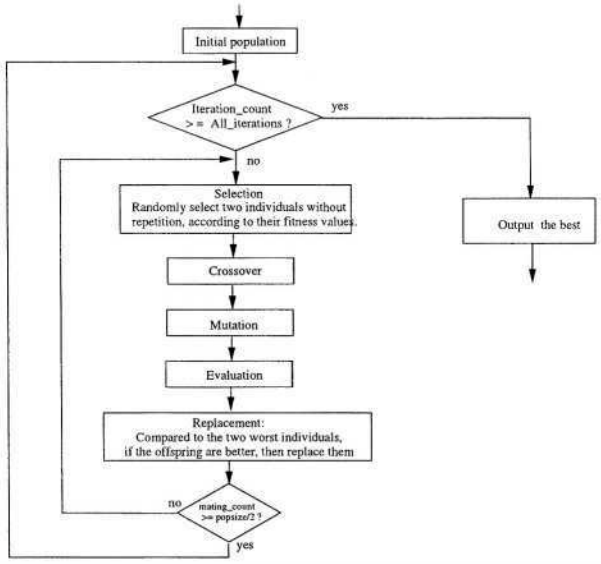


Fig. 3. A steady-state genetic algorithm for placement

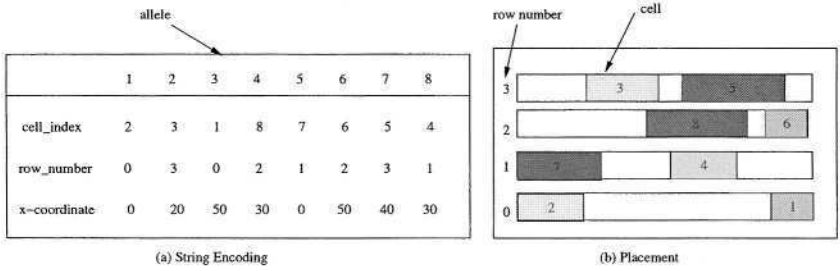


Fig. 4. String Encoding

convergence. In [1], migration occurs after the subpopulations completely converged. The study claimed that if migration is introduced too early before search converged, good original schemata in the demes may be destroyed by incoming migrants and thus the demes may lose their diversity.

However, previous studies show that large circuits require extremely long time to converge. Our experimental work shows that for small circuits convergence takes several hours even if we parallelize the GA using 7 processors. Since using convergence to trigger migration is not practical for placement, we proposed a better migration scheme, and successfully implemented a synchronous island-based GA for standard-cell placement. Besides, we also implemented an

asynchronous model that can achieve super-linear speedup. The two island-based models are described in the next two sections.

3 Synchronous Island-Based GA

In the synchronous island-based GA, the total population is divided equally into several subpopulations, and each processor is assigned a single subpopulation to form an island. As depicted in Figure 5, the sub-algorithm in each island is simply a regular serial steady-state GA plus migration.

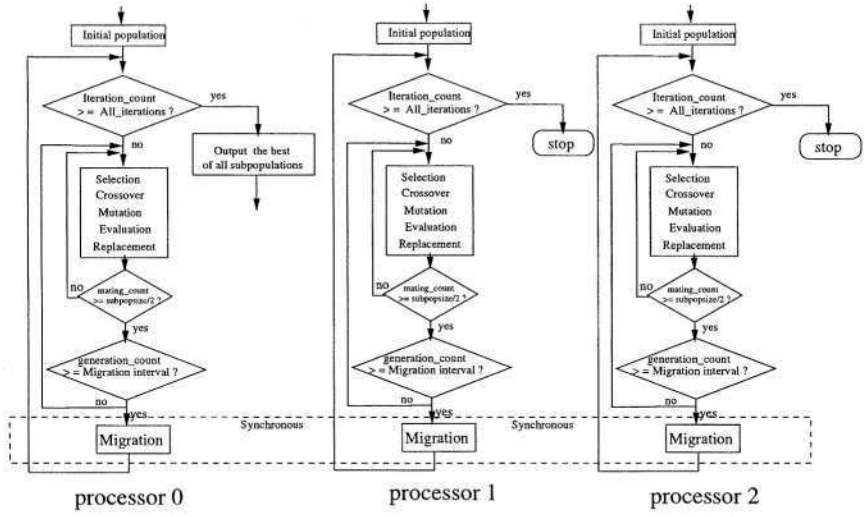


Fig. 5. A synchronous model with 3 processors

Each sub-algorithm includes an extra phase of periodicly exchange of some of their individuals. More specifically, for every certain number of generations (known as migration interval), all islands exchange a candidate of good individuals (migrants) with each other, based on the communication topology. For each island, if an incoming migrant is better than the worst existing individual, this migrant is then inserted into the subpopulation. All islands perform their migrations at the same time.

The mechanism of island models is complex, since many parameters affect the quality of search as well as efficiency of parallelism. Besides the basic GA parameters, other parameters are involved such as deme size, migrants selection method, migration interval, migration rate, communication topology, and replacement rules.

3.1 The Deme Size and the Number of Islands

For a specified population size, the deme (subpopulation) size is determined by the number of processors used. We can imagine that, the more processors we throw into the parallel system, the smaller the size of subpopulations, and the shorter the computation time. However, this is not always the case. The execution time of a parallel GA includes computation time and communication time. As more processors are used, the average available bandwidth for each processor decreases, but more communication is required for each processor. The latter grows especially fast in a low bandwidth parallel system. On the other hand, a very small subpopulation might prematurely converge and leads to poor solutions. Therefore, when we scale up a parallel GA system to pursue better performance, the parallel efficiency usually diminishes.

3.2 Migration

Migration is the process where subpopulations occasionally exchange some individuals. It is a very important parameter that determines the quality of the search as well as the efficiency of the parallelism. If the migration is low, the sub-algorithms on different small subpopulations work independently and the final result may be worse than a serial GA. If the migration is high, the parallel GA behaves very similarily to a serial GA [8,9]. There are many ingredient of migration, and they all affect the behaviors of the parallel genetic search:

- **Migrants Selection Methods:** Among the chromosomes in an island, high quality individuals are chosen to be sent out as migrants. There are several techniques that can be used to select such individuals. One way is just simply picking the best individuals. Other techniques involve selecting the best individuals probabilistically, such as Roulette-wheel selection, Stochastic universal selection, and Binary tournament selection. The higher the pressure, the faster the algorithm reaches equilibrium while sacrificing more genetic diversity of those migrants [13]. In our implementation, the outgoing migrants are from the best individuals, with the restriction that each individual being sent once. In addition, our algorithm restricts an incoming immigrant to be sent out directly but allows such a mechanism for children. In the 'to-all' communication scheme, only the best unsent individual of a deme can be sent out as migrant in each migration.
- **Migration Interval:** Migration intervals specify the migration plan. Most island-based GAs start migrations after subpopulations are converged. But for standard-cell placement, we use a fixed epoch in our synchronous model, and start migrations at very early generations. The reasons are: (i) Without extra computation, it's simple; (ii) A string encodes a placement solution of a VLSI chip, for medium or large circuits, the time of waiting for convergence could be too long in practice. We also believe that if we use such a fixed migration interval and migration scheme we could make distributed demes behave like a single panmictic population (serial GA) that would outperform the serial GA in all generations.

– **Communication Topologies:** The communication topology determines the pattern of communications, or the connections between subpopulations. The ring topology is especially popular, since it has the longest diameter and demes are more isolated than other topologies. With a ring topology, the parallel search may find a better solution than that of a serial GA in late generations. However, because the demes are very much isolated, the small deme size usually produces solutions worse than that of a serial GA with a single population in early generations. To solve this problem, we use a complete graph topology called 'to-all' topology. With such a topology, all demes are fully connected. During migration, each island contributes the best individual to the global migrant pool. Since the connection between demes is very strong in this topology, the frequency of migration must be controlled in some way and the number of migrants must be carefully set, such that the strength of migration will not be excessive. As shown in Figure 6, only a portion of the best individuals in the pool are merged back into the subpopulations. With this migration scheme, a global superfit individual can move across the overall population in every migration interval, while some degree of diversity between the subpopulations are maintained. Therefore, the parallel GA may have the chance to outperform its serial version in a very short time. Also, due to the high connectivity among islands, this synchronous model would search in a trajectory similar to a serial GA.

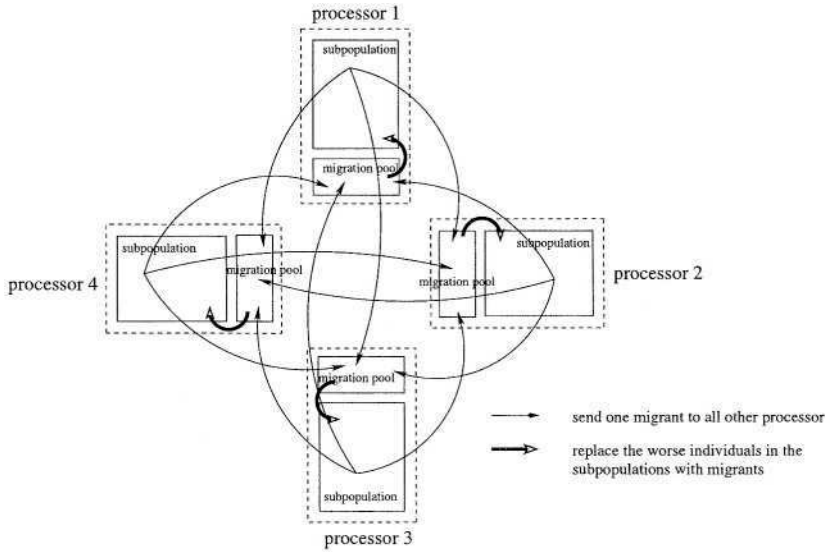


Fig. 6. The migration scheme of the 'to-all' topology

– **Migration Rate:** In our algorithm, if the ring topology is used, the migration rate is defined as the percentage of the individuals in a subpopulation

that need to be sent out as migrants; If the 'to-all' topology is used, the migration rate specifies the percentage of migrants in the migration pool that will spread to all subpopulations. A very high migration rate often leads to two problems: first, high communication costs; secondly, superfit migrants may take over the receiving subpopulation too soon. Search quality usually benefits from an appropriate migration rate.

- **Migrants Replacement Rule:** After an island receives some migrants, it replaces the worst individuals with these incoming migrants. For the 'to-all' migration scheme, although each island sends out only one migrant, it may receive more than one incoming individual to replace the worst members in its subpopulation.

The main disadvantage of synchronous models is that the migration of all islands occurs at the same time and the faster processors have to wait for the slower ones.

4 Asynchronous Island-Based GA

Since our goal aims at performance related issues, an asynchronous island-based GA was also implemented for standard-cell placement. In an asynchronous model, islands are free to evolve and to migrate their individuals without waiting for others. As illustrated in Figure 7, the workstations involved are configured as a master/slaves style, where each slave is a processor holding a subpopulation and the only acting master is a central controller.

4.1 The Master

The master brings two functions to the asynchronous model:

1. **Migration controller:** Controls the overall migration activities and dynamic load balancing including routing migrants, communication topology, migration plan, and dynamic load balancing. Figure 7 illustrates that the slaves report their current states to the master periodically and the master sends control signals to command these slaves.
2. **High speed communication router:** In our current implementation, there is no direct communication between slaves, and migration is performed through the master. Since the master keeps listening to the communication channels, it is able to respond to the requests from slaves almost immediately. After the master receives migrants sent from a slave, it buffers these messages, and controls the migration scheme. Incorporating a master in the system eliminates virtually all delay that might occur between processors. The master can easily choose a communication topology. Since our 'to-all' topology is synchronization in nature, we use an alternative topology in the form of a ring.

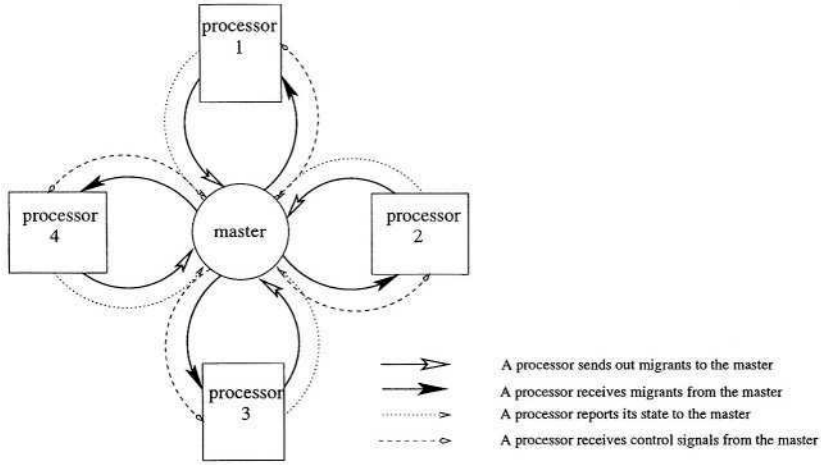


Fig. 7. The migration scheme of the asynchronous model with ring topology

4.2 Dynamic Load Balancing

Since subpopulations are free to evolve, some processors might finish their task much earlier than others. This leads to two problems: (i) First, if a migration occurs between an evolving island and an idle island, the migration becomes meaningless; (ii) Secondly, the parallel system still has to wait for the slowest processor to finish its work. Therefore, a load balancing mechanism is needed to remove some load from slower processors and transfer it to faster processors.

In Matthem's algorithm [14], if a processor completes its work early, it becomes idle and the algorithm reassigns some work to this processor from the other processors. However, we suspect a possible problem in this algorithm: the migration is now between two portions of the same subpopulation instead of two different demes. Therefore, we use different strategies for dynamic load balancing. Instead of taking part of the work from a slower processor and replacing the whole subpopulation of faster processor, we periodically detect the evolution speed on varied demes, and dynamically change the sizes of the subpopulations to match the speed.

4.3 Distributed Random Migration Plans

In the natural world, *islands* are independent and semi-isolated. Since subpopulations are free to evolve in our asynchronous model, we have a chance to give more independence to these subpopulations by (1)applying distributed random migration plan created in each island; and (2) using random number of migrants (with some control) for each migration. With such distributed random plans, the subpopulations are allowed to evolve totally asynchronously. The strategy attempts to simulate the fact that the number of migrants in each migration is not constant in the real world.

5 Experimental Results

All of the test runs were conducted on a cluster of networked Sun workstations, using the MPI (Message Passing Interface) v.1.2.5 based on the 'C' programming language. Many parameters affect an island-based parallel GA, thus we carried out a large number of runs. Table 1 presents the benchmarks that were used in our experiments. Two to seven islands were applied for the parallel algorithms in the tests. Each test was conducted 20 times and the average values were recorded.

Table 1. Benchmarks used as test cases

Circuit	ckt1_52	fract	struct	prim2	avq_large
Nodes	64	149	1952	3121	25178
Nets	55	147	1920	3136	25384
Pins	278	876	10814	22371	165374

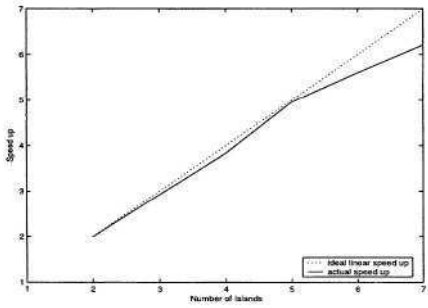


Fig. 8. Synchronous model: 'to-all' topology

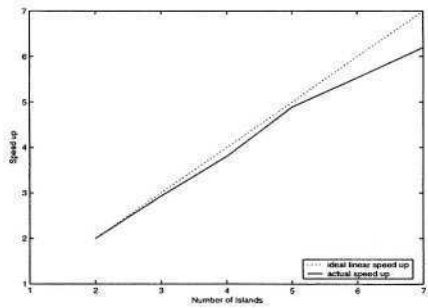


Fig. 9. Synchronous model: ring topology

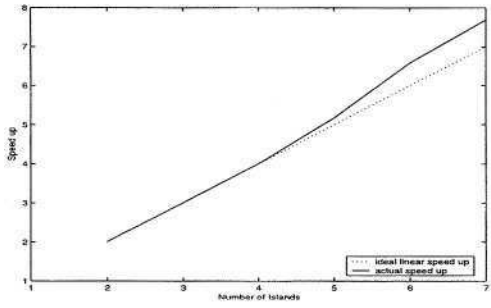


Fig. 10. Asynchronous model: ring topology

5.1 Performance: Speedup Versus Processors

The performance of the two synchronous migration schemes is shown in Figure 8 and Figure 9 respectively. The parallel systems were configured from two to seven processors, and circuit *struct* was tested. The experiment shows both the ring migration scheme and the 'to-all' migration scheme obtained near ideal linear speed up. The latter is even slightly better than the former. As expected, when the number of processors involved increased, the communication costs for synchronization grew, and a decrease of the parallel efficiency was observed.

With asynchronous migration and dynamic load balancing, the asynchronous model obtained super linear speedup (as seen in Figure 10). For example, 7 islands achieved a speedup of 7.6. [15] explains why super linear speedup is possible for asynchronous models, and [16] acquired similar results with their implementation.

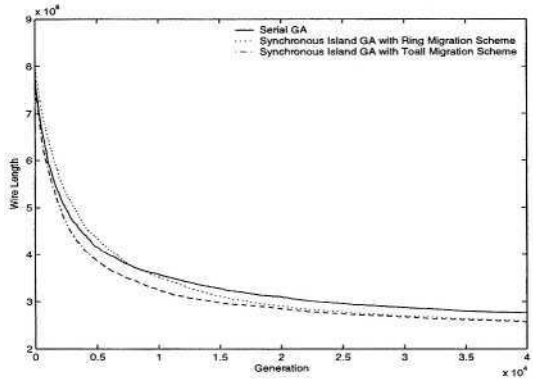


Fig. 11. Placement quality: Serial GA, Ring PGA, and 'to-all' PGA

5.2 Solution Quality

In order to compare the search quality of the proposed 'to-all' migration scheme with that of the ring migration scheme, we plotted their search for 40000 generations as seen in Figure 11 (a serial GA was also plotted as baseline). In this experiment, seven processors were used. To assure the fairness, we maintain the same basic GA parameters for all of them.

Like most implementations in the literature, the parallel search with ring topology outperforms the serial GA in the medium (> 8000 generations, in this case) and later generations. However, no implementation with ring topology (including ours) is able to surpass its serial version in early generations. Usually, to obtain a better solution than a serial GA, the ring topology parallel GA requires a large enough number of generations. The proposed 'to-all' migration scheme, on the other hand, works better than a serial GA almost all the time!

The advantage of the new migration scheme over ring topology is that, depending on how much time available, we can run the parallel GA for 200 generations or 20000 generations, and still get better solution than that of a serial GA. It is important to note that, other experiments show that the execution time for both migration schemes are pretty much the same.

We also compared the search quality of a synchronous island-based GA (with proposed 'to-all' migration scheme) to a serial based implementation in the early generations for different circuits, as shown in Table 2. The algorithms were run for only 100 generations, and 7 processors were used for the PGA (parallel GA). Each test was conducted 20 times, and we recorded the average values. The results show that the parallel implementation achieved speedups from 6.06 to 6.38 with 7 processors, and the proposed migration scheme yielded better solutions than the serial GA from very early generations (the 100th generation).

Table 2. Search results in the 100th generation (7 processors were used for PGA)

Benchmark	Execution time (SGA)	Execution time (PGA)	Speedup
ckt1_52.yal	2.8844803	0.4522641	6.38
struct.yal	475.7	76.139	6.25
prim2.yal	1180.28	188.41	6.264
avq_large.yal	94778.5	15631.4	6.063
Benchmark	**Wire length (SGA)**	**Wire length (PGA)**	**Improvement**
ckt1_52.yal	20177.6	19591.5	2.9%
struct.yal	7.5124×10^6	7.4958×10^6	0.22%
prim2.yal	6.800114×10^7	6.79839×10^7	0.024%
avq_large.yal	1.57018×10^9	1.568035×10^9	0.1366%

6 Conclusion

This paper proposed a practical migration scheme for synchronous island-based GA. Experimental results show that our algorithms can achieve near linear or even super-linear speedup. In addition the proposed migration scheme yields better quality of solutions than a serial GA from an early stage of evolution. Although we carried out some experiments with a large number of generations, we are especially interested in the search quality of initial phase of the search. Running a circuit placement with a GA for 10000 generations is usually not practical. This emphasizes the advantage of the proposed migration scheme in the application of standard-cell placement. The implemented asynchronous model was able to achieve super-linear speedup as expected.

References

1. Braun, H.: On solving travelling salesman problems by genetic algorithm. In Schwefel, H.P., Manner, R., eds.: Parallel Problem Solving from Nature, Springer-Verlag (1990) 129–133
2. Horii, H., Kunifuji, S., Matsuzawa, T.: Asynchronous island parallel GA using multiform subpopulations. Lecture Notes in Computer Science **1585** (1999) 122–129
3. Sherwani, N.A.: Algorithms for VLSI Physical Design Automation. Kluwer Academic Publishers, Intel Corporation, Hillsboro, OR, USA (1998)
4. Chang, H., Cooks, L., Hunt, M.: Surviving the SOC Revolution. Kluwer Academic Publishers, London (1999)
5. Kling, R.M., Banerjee, P.: Optimization by simulated evolution with applications to standard cell placement. In: Conference proceedings on 27th ACM/IEEE design automation conference, ACM Press (1990) 20–25
6. Kilng, R., Banerjee, P.: Esp: Placement by simulated evolution. IEEE Trans. on Computer Aided Design, 8(3) (1989) 245–256
7. Shahookar, K., Mazumder, P.: A genetic approach to standard cell placement using metagenetic parameter optimization. IEEE Trans. on CAD , vol. 9 (1990) 500–511
8. Whitley, D., Rana, S.B., Heckendorn, R.B.: Island model genetic algorithms and linearly separable problems. In: Evolutionary Computing, AISB Workshop. (1997) 109–125
9. Cantu-Paz, E., Goldberg, D.E.: Efficient parallel genetic algorithms: Theory and practice (2000)
10. Shahookar, K., Mazumder, P.: VLSI Cell Placement Techniques. ACM Computing Surveys **23** (1991) 143–220
11. Areibi, S.: Memetic algorithms for vlsi physical design: Implementation issues. In Hart, W., Krasnogor, N., Smith, J., eds.: Second Workshop on Memetic Algorithms (2nd WOMA). (July 2001) 140–145
12. Munetomo, M., Takai, Y., Sato, Y.: An efficient migration scheme for subpopulation-based asynchronously parallel genetic algorithms. In: Proceedings of the Fifth International Conference on Genetic Algorithms. (1993) 649
13. Cantu-Paz, E.: Migration policies, selection pressure, and parallel evolutionary algorithms. In Brave, S., Wu, A.S., eds.: Late Breaking Papers at the 1999 Genetic and Evolutionary Computation Conference, Orlando, Florida, USA (1999) 65–73
14. Mcmahon, M.T., Watson, L.T.: A distributed genetic algorithm with migration for the design of composite laminate structures (1998)
15. Alba, E., Troya, J.M.: An analysis of synchronous and asynchronous parallel distributed genetic algorithms with structured and panmictic islands. In: IPPS/SPDP Workshops. (1999) 248–256
16. Andre, D., Koza, J.R.: A parallel implementation of genetic programming that achieves super-linear performance. In Arabnia, H.R., ed.: Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications. Volume III., Sunnyvale, CSREA (1996) 1163–1174

Exploratory Data Analysis with Interactive Evolution

Sergey Malinchik and Eric Bonabeau

Icosystem Corporation, 10 Fawcett St., Cambridge, MA 01238, USA
{sergey, eric}@icosystem.com
http://www.icosystem.com

Abstract. We illustrate with two simple examples how Interactive Evolutionary Computation (IEC) can be applied to Exploratory Data Analysis (EDA). IEC is particularly valuable in an EDA context because the objective function is by definition either unknown *a priori* or difficult to formalize. The first example involves what is probably the simplest possible transformation of data: linear projections. While the concept of linear projections is simple to grasp, in practice finding the appropriate two-dimensional projection that reveals important features of high-dimensional data is no easy task. We show how IEC can be used to quickly find the most informative linear projection(s). In another, more complex example, IEC is used to evolve the "true" metric of attribute space. Indeed, the assumed distance function in attribute space strongly conditions the information content of a two-dimensional display of the data, regardless of the dimension reduction approach. The goal here is to evolve the attribute space distance function until "interesting" features of the data are revealed when a clustering algorithm is applied.

Keywords: Interactive evolutionary computation, data mining, exploratory data analysis.

1 Introduction

At a time when the amount of data that is potentially available for analysis and exploitation is increasing exponentially in all fields and industries, the ability to quickly explore gigantic databases for valuable patterns is becoming a crucial competitive advantage. The problem is: what should one look for? Exploratory Data Analysis (EDA) [1,2] is the art of exploring data without any clear *a priori* ideas of what to look for. Most techniques used in EDA are *interactive* and *visual.* There are a number of display techniques that take care of the visualization aspect. As the dimensionality of the data (that is, the number of attributes) increases, informative low-dimensional projections of the data are necessary (most often in two dimensions). Families of techniques such as multi-dimensional scaling (MDS) [3] are used to generate such projections. While there exist powerful visualization techniques, the interactivity of EDA is often *ad hoc* and best characterized as tinkering. That is because in EDA the notion of "interestingness" of a particular display or pattern is, by definition, difficult to formalize. Thus designing data mining patterns, filters, projection algorithms,

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1151–1161, 2004.

clustering algorithms and other search models that will produce "interesting" results when applied to a dataset is a difficult problem [2].

If one assumes that interesting patterns will be recognized when they are discovered even though they could not be formulated ahead of time, a technique originally developed to generate "interesting" images and pieces of art [4,5,6,7], Interactive Evolutionary Computation (IEC), can be used. Although IEC has been applied to some data mining problems in the past (see [8] for a review), it has never been described as a canonical tool to perform EDA. This paper is an attempt to show how pervasive the technique might become. IEC is a directed search evolutionary algorithm which requires human input to evaluate the fitness of a pattern (here, the fitness might be how interesting a detected pattern is) and uses common evolutionary operators such as mutation and crossover [9] to breed the individual-level rules that produced the fittest collective-level patterns. IEC combines computational search with human evaluation [8,10].

2 The Problems

We illustrate the use of IEC in EDA with two simple examples using the same five-dimensional real-valued dataset described in section 2.1. In the first example, IEC is used to evolve two-dimensional linear projections of the dataset, a simple but surprisingly powerful aid in exploring data; the linear projections are evaluated according to how much insight they provide the user about the data. In the second example, IEC is used to evolve the distance function in attribute space so as to produce the most compelling clusters using a simple parametric clustering algorithm.

2.1 The Dataset

All experimental results described in this paper were obtained using the same synthetic, five-dimensional real-valued dataset. The five dimensions are represented as $(X_1, X_2, X_3, X_4, X_5)$. The dataset contains 24,000 points in 5 separate clusters. The clusters are generated in the following way: each of the five coordinates $(x_1, x_2, x_3, x_4, x_5)$ of each data point is generated independently from a Gaussian distribution whose mean is equal to the corresponding coordinate of the center of the cluster that point belongs to, and whose standard deviations, listed below, reflect the cluster's elongation or compression along certain dimensions. Some of the planar projections of some of the clusters are then rotated by a certain angle.

The five clusters are defined as follows:

- Cluster #1:
  3000 points, center located at: (650, 255, 540, 500, 300), standard deviations for each coordinate: (50, 40, 20, 200, 60); (X_3, X_4) coordinates rotated by 20 degrees in the (X_3, X_4) plane around point $(x_3=0, x_4=0)$.

- Cluster #2:
  6000 points, center located at: (450, 200, 400, 500, 300), standard deviations for each coordinate: (50, 70, 150, 40, 60); (X_3, X_4) coordinates rotated by 30 degrees in the (X_3, X_4) plane around point $(x_3=0, x_4=0)$.
- Cluster #3:
  7000 points, center located at: (400, 400, 500, 200, 300), standard deviations for each coordinate: (90, 50, 160, 40, 20); (X_3, X_5) coordinates rotated by 90 degrees in the (X_3, X_5) plane around point $(x_3=0, x_5=0)$.
- Cluster #4:
  4000 points, center located at: (600, 550, 350, 300, 540), standard deviations for each coordinate: (80, 70, 20, 40, 80).
- Cluster #5:
  4000 points, center located at: (350, 580, 600, 340, 600), standard deviations for each coordinate: (60, 70, 40, 40, 60)

Although the dataset is simple, discovering its cluster properties (# of clusters, locations, shapes) without any *a priori* knowledge is a non-trivial task.

2.2 Evolving Linear Projections

The general problem here is to find a linear transformation from a N-dimensional space (also called attribute space) onto a two-dimensional representation. For example, with real-valued attributes in N dimensions, the transformation would be from R^N to R^2. Each point x is characterized by N coordinates or attributes: $x = (x_1, x_2 .. x_N)$. x is projected onto a point with coordinates (x', y') in a two-dimensional space, via the following transformation:

$$x' = \sum_{i=1}^{N} \alpha_i x_i , \qquad (1)$$

$$y' = \sum_{i=1}^{N} \beta_i x_i , \qquad (2)$$

where $-1 \le \alpha_i, \beta_i \le 1$ for all i=1,... N and

$$\sum_{i=1}^{N} \alpha_i^2 = 1, \qquad (3)$$

$$\sum_{i=1}^{N} \beta_i^2 = 1. \qquad (4)$$

The transformation's parameters $\alpha = (\alpha_1, \ldots, \alpha_N)$ and $\beta = (\beta_1, \ldots, \beta_N)$ belong to the unit sphere in R^N. The problem here is very simple: what are the most informative linear projections from the user's perspective? The problem in defining a fitness function for "most informative" is that we don't know ahead of time what it means. Although it is probably the simplest possible EDA problem one can think of, it is of

practical importance when one wants to display very high-dimensional data using a small number of two-dimensional projections. Such projections obviously destroy a potentially significant amount of information but two-dimensional displays constitute the most commonly used way of visualizing data. Finding a good set of two-dimensional projections will help the user tremendously.

2.3 Evolving Distance Functions in Attribute Space

Clustering is a useful and commonly used technique in EDA. The goal of clustering is to compress the amount of data by categorizing or grouping similar items together [11]. There are many different clustering algorithms. However, the clusters resulting from the application of one specific clustering algorithm to a data set are heavily dependent on the distance function assumed in attribute space. That distance function is rarely questioned and even more rarely an object of study. For example, when dealing with real-valued data, it is often implicitly assumed without further examination that the relevant distance function is the Euclidian distance or L_p (L_1 = city-block distance; L_2 = Euclidian distance; L_∞ = max norm). But that might not appropriately reflect the potentially complex structure of attribute space. For example, the use of the Euclidian distance in attribute space for the dataset described in Section 2.1 does not lead to the extraction of the data's clusters because of the elongation and compression of the clusters along certain dimensions. The problem here is to discover a distance function in attribute space that contains some of the fundamental properties of that space. To do so we apply a simple parametric clustering algorithm (K-means, well-suited to the globular clusters of our dataset) to the data using a variety of distance functions and we then examine the resulting clusters; the distance function is evolved until the clusters look "right" or provide valuable information.

A commonly used clustering method is K-means clustering [11], which is a least-squares partitioning method allowing users to divide a collection of objects directly into K disjoint clusters. The energy function E_K that is minimized in K-means is the sum of the squared distances between each data item x_m and the nearest cluster centroid:

$$E_K = \sum_m \| x_m - c(x_m) \|^2 \qquad (5)$$

where $c(x_m)$ is the centroid that is closest to x_k. The algorithm starts by initializing a set of K cluster centroids denoted by c_i, $i=1,\ldots K$. Data points are examined sequentially in a random order and each point is assigned to the nearest centroid. The positions of the centroids are then adjusted iteratively by re-computing the centroids to move them closer to the set of points that belong to the corresponding cluster. Several passes through data are performed until every data point is consistently assigned to one cluster, that is, until the assignment of points to clusters is stable.

K-means is characterized by simplicity and computational efficiency but it is sensitive to cluster shapes. In particular, it fails when clusters are too close to one another. When clusters are strongly anisotropic (for example, elongated or compressed along certain dimensions) it is helpful to define the distance function in such a way that it counterbalances cluster asymmetry, thereby revealing the structure of attribute space. The family of weighted Euclidean distance functions in R^N is explored using IEC. A distance function in that family is given by:

$$d_w(x^p, x^q) = \sqrt{\sum_{i=1}^{N} w_i (x_i^p - x_i^q)^2} \tag{6}$$

where x^p and x^q are two points in R^N, and $w=(w_1, w_2, \ldots, w_N)$ is the weight vector that characterizes that distance function, with $0 \le w_i \le 1$, $i=1,\ldots,N$, and

$$\sum_{i=1}^{N} w_i = 1 . \tag{7}$$

The object of the IEC-based search is to evolve w so as to extract valuable information from the clusters.

3 Interactive Evolution

3.1 Overview of the Search Mechanism

The IEC search method works as follows. A small initial population of solutions is generated, where a solution is a linear projection (Example 1) or a distance function (Example 2).

- The resulting two-dimensional representations are computed, generated and displayed to a human observer.
- The observer selects the displays that are the most interesting –the fittest individuals in the population according to whatever set of objective and subjective criteria the observer may be using, or assigns a fitness value to each display.
- A new population (new generation) of solutions is generated by applying mutation and crossover operators to the solutions that correspond to the previous generation's fittest displays.
- In addition to the offspring and mutated versions of those solutions, randomly generated solutions are injected into the population to ensure diversity.
- The new population's two-dimensional representations are then calculated and the results displayed to the observer, and so forth.
- This procedure is iterated until interesting displays emerge from the search, pointing toward corresponding linear projections or distance functions.

The user interface is a critical component of the method. The observer evaluates solutions based on visual inspection of their two-dimensional representations. The vis-

ual interface is shown in Figure 1. Obviously this method can only work if the population size is kept small and if interesting solutions emerge after a reasonably small number of generations.

3.2 Genetic Algorithm

The evolutionary mechanism is the same in Example 1 and Example 2.

- A simple real-valued representation is used to encode the genotype. In Example 1, the genotype is a juxtaposition of $2N$ real-valued genes: $(\alpha_1,\ldots,\alpha_N,\beta_1,\ldots,\beta_N)$, with N=5.

  In Example 2, the genotype is a juxtaposition of N real-valued genes $(w_1,\ldots,w_N)$ with N=5.
- Population size is equal to nine.
- Initial gene values are generated randomly from a uniform distribution over the allowed range.
- The user assigns fitness values to the various displays. The default value for all displays before user intervention is 0. In Example 2, the user can play with different views (two-dimensional projections onto coordinate planes) of the data before assigning a fitness value. After visual inspection of the dataset (using, for example, the linear projections evolved in Example 1), the user sets the number of clusters (K) and the tool randomly generates the starting positions of the cluster centroids in the original 5-dimensional space. Each of the nine solutions displayed represents the results of the application of the K-means clustering algorithm with a different distance function in attribute space, characterized by the weight vector $\boldsymbol{w}$.
- The results are displayed simultaneously to the user. The user can move back and forth and reassign fitness values.
- The user can select one or more displays to be part of an elite pool. Unless otherwise specified by the user, only one elite individual is selected by the algorithm, which is the one with the highest fitness.
- Clicking the mouse on the corresponding image user indicates his choice and the tool automatically sets the fitness function to the value of 5. Double clicking means the selection of the elite individual, which fitness function will be equal to 10. The user's choice is indicated on the screen by a green or yellow frame around ordinary selected individuals or elite ones, respectively.
- The mating pool is comprised of solutions with fitness larger than zero.
- The next generation is created in the following way: one elite solution remains unchanged. Four solutions are created from the ranked mating pool. Two offspring are created from two randomly selected parents in the mating pool by applying a random single point crossover. The last four solutions are created from single parents from the mating pool after application of a single point mutation. The new value of the mutated point is drawn from a normal distribution function whose mean is equal to the current gene value and whose standard deviation equal to one third of the gene value range.

4 Results

4.1 Example 1

The search is performed until the user finds a rich enough representation of the data in the form of a linear two-dimensional projection. It typically takes from 5 to 15 generations to reveal the internal structure of the dataset described in Section 2.1.

Figure 1 demonstrates the evolution of the projections of the five-dimensional dataset. Initial projections are very poor, providing little insight into the data.

A typical selection strategy by the user consists of selecting projections based on the visual improvement of some data substructure.

4.2 Example 2

The user performs the interactive search for distance functions in the same way as for Example 1, looking for valuable and consistent information.

Although the user may not know ahead of time what "valuable" information is contained in the data, there are obvious shapes he will be looking for in the search, for example ensembles of points that seem to belong to the same cluster when viewed

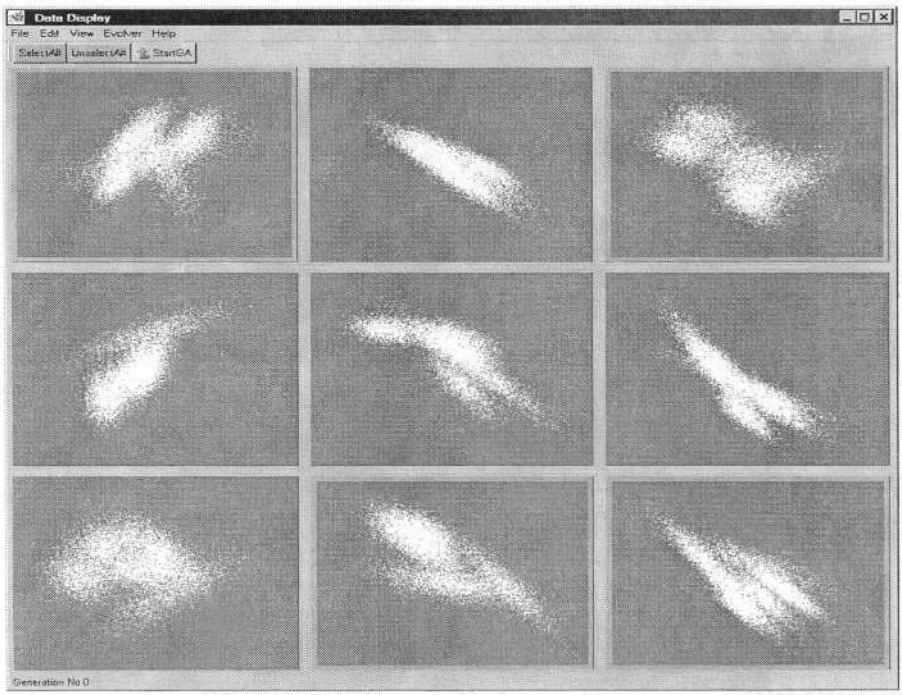


Fig. 1 (a)

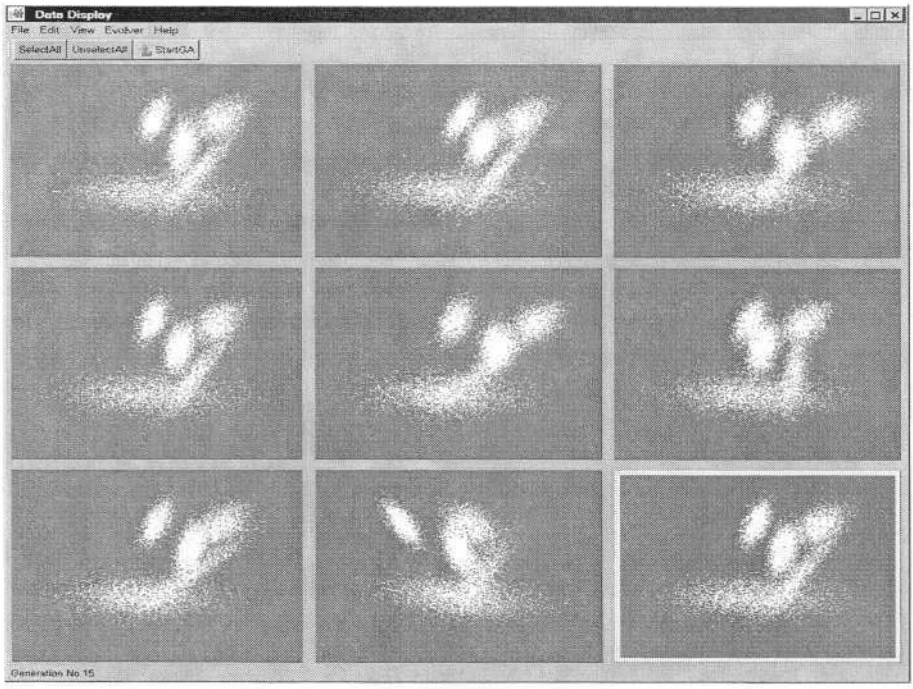


Fig. 1 (b)

Fig. 1. Example 1: User interface with nine solutions displayed. (a) Initial population. (b) Generation 15. Projection parameters: $\alpha_1 = 0.02$, $\alpha_2 = -0.04$, $\alpha_3 = 0.6$, $\alpha_4 = 0.08$, $\alpha_5 = -0.05$, $\beta_1 = -0.11$, $\beta_2 = -0.15$, $\beta_3 = -0.35$, $\beta_4 = 0.74$, $\beta_5 = 0.26$.

from different projections but are not assigned to the same cluster by the clustering algorithm, thereby suggesting a flaw in the distance function. Difficult clusters for a clustering algorithm such as K-means to map are for example elongated clusters. The user will therefore tend to favor distance functions that can map those clusters.

In Figure 2 we present nine displays resulting from the application of K-means clustering for randomly generated values of the weight vector $\boldsymbol{w}$ (Figure 2a), and nine displays obtained after 15 generations with interactively evolved $\boldsymbol{w}$ (Figure 2b). In both cases, the display uses the same specific projection of the dataset, obtained from the IEC run described in Example 1. A criterion that emerged as the main basis for the user's selection of the best distance functions is the presence of homogeneously colored clusters. If a cluster is not homogeneously colored or if it appears to be cut in the middle, the user easily detects this spatial inconsistency and assigns a low fitness value to such a solution. Although there is a range of values of w that yield a good clustering, all of the best solutions typically share the same properties: w_3, w_4 and w_5 have significantly lower values than w_1 and w_2, thereby indicating that the attribute space has anisotropic properties. A good solution arrived at with IEC is: $w_1 = 0.33$, $w_2 = 0.35$, $w_3 = 0.12$, $w_4 = 0.06$ and $w_5 = 0.12$.

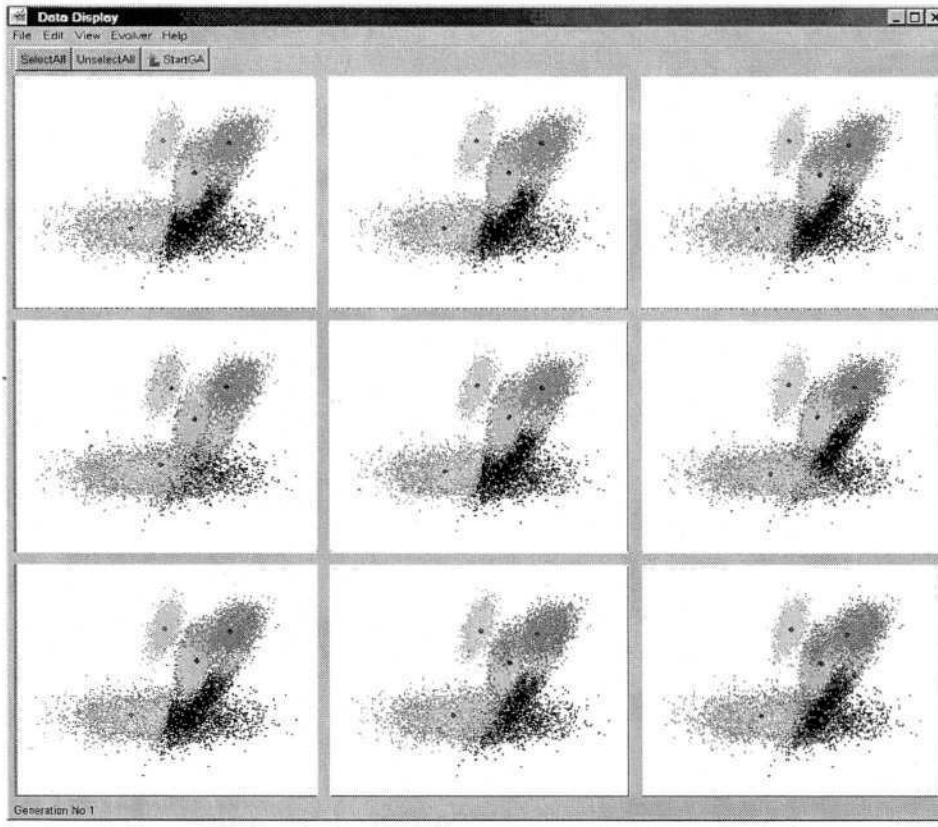


Fig. 2. (a)

5 Discussion

We have illustrated with two simple toy examples how IEC can assist in EDA and could in fact become a canonical EDA tool, helping to structure the user's intuition and insights in situations where the value of the results from a data mining algorithm is not known ahead of time and/or is difficult to formalize. There are obviously limitations to what can be achieved with IEC since it relies on small populations and a small number of generations [8]. One can however increase the potential of an IEC search by combining it with an automated pre-processing or filtering of the solutions when some objective, formalized criteria and constraints are known, so that only pre-processed solutions are presented to the user. Another useful extension would be to give the user the ability to explicitly select sub-structures or sub-modules of the display that seem promising. The specific genetic algorithm used in this paper can also be improved, for example by introducing a self-adaptable mutation step size.

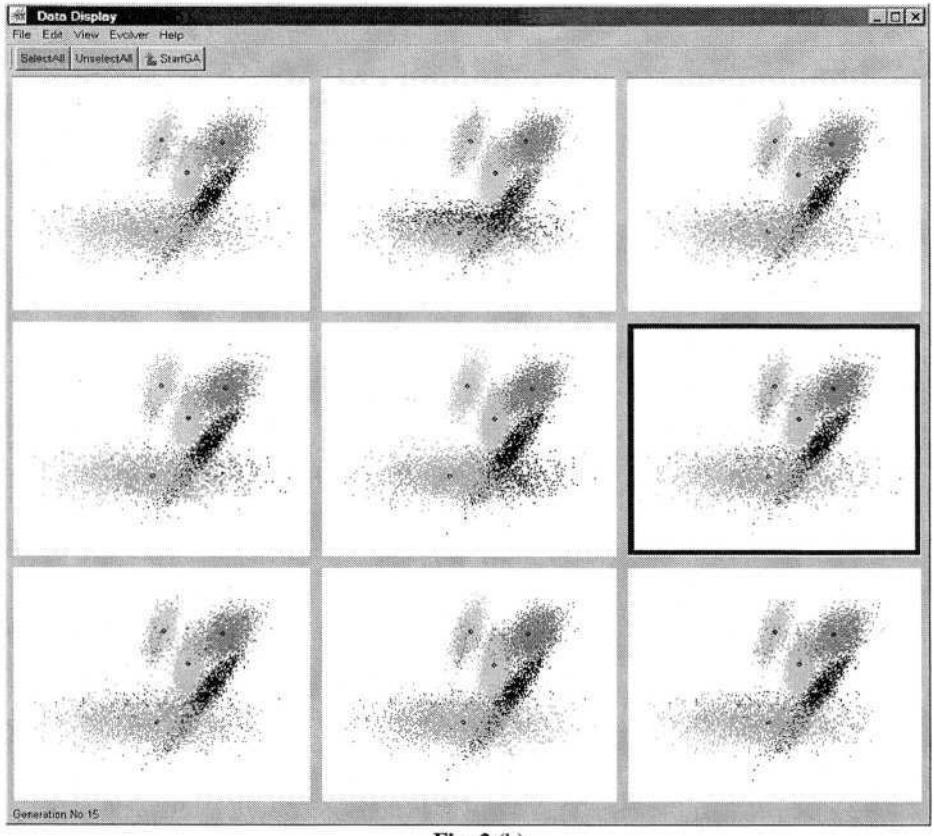


Fig. 2. Example 2: User interface with nine solutions displayed. Black circles represent cluster centers. Different colors represent different clusters. (a) Initial population. (b) Generation 15.

References

1. Tukey, J. W. 1977. *Exploratory Data Analysis.* Addison Wesley, Reading, MA.
2. Hand, D., Mannila, H. & Smyth, P. 2001. *Principles of Data Mining.* MIT Press, Cambridge, MA.
3. Cox, T. F. & Cox, M. A. A. 1994. *Multidimensional Scaling.* Chapman and Hall, London.
4. Dawkins. R. 1987. *The Blind Watchmaker.* W. W. Norton, New York.
5. Sims, K. 1991. Artificial evolution for computer graphics. *Computer Graphics* **25**: 319-328.
6. Sims, K. 1992. Interactive evolution of dynamical systems. Pages 171-178 in: *Towards a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life* (F. J. Varela & P. Bourgine, eds.), MIT Press, Cambridge, MA.
7. Sims, K. 1993. Interactive evolution of equations for procedural models. *Vis. Comput.* **9**: 446-476.

8. Takagi, H. 2001. Interactive evolutionary computation: fusion of the capabilities of EC optimization and human evaluation. *Proc. IEEE* **89**: 1275-1296.
9. Michalewicz, Z. 1999. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer
10. Bonabeau, E., Guérin, S., Snyers, D., Kuntz, P., Theraulaz, G. & Cogne, F. 2000. Complex three-dimensional architectures grown by simple agents: an exploration with a genetic algorithm. *BioSystems* **56**: 13-32.
11. Jain, A. K. & Dubes, R. C. 1988. *Algorithms for Clustering Data.* Prentice Hall, Englewood Cliffs, NJ.

Designing Multiplicative General Parameter Filters Using Adaptive Genetic Algorithms

Jarno Martikainen[1] and Seppo J. Ovaska[2]

Institute of Intelligent Power Electronics, Helsinki University of Technology,
P.O. Box 3000, FIN-02015 HUT, Finland
[1]jkmartik@cc.hut.fi, [2]ovaska@ieee.org

Abstract. Multiplicative general parameter (MGP) approach to finite impulse response (FIR) filtering introduces a novel way to realize cost effective adaptive filters in compact very large scale integrated circuit (VLSI) implementations used for example in mobile devices. MGP-filter structure comprises of additions and only a small number of multiplications, thus making the structure very simple. Only a couple of papers have been published on this recent innovation and, moreover, MGP-filters have never been designed using adaptive genetic algorithms (GA). The notion suggesting the use of adaptive parameters is that optimal parameters of an algorithm may change during the optimization process, and thus it is difficult to define parameters beforehand that would produce competitive solutions. In this paper, we present results of designing MGP-FIR basis filters using different types of adaptive genetic algorithms, and compare the results to the ones obtained using a simple GA.

1 Introduction

Predictive lowpass and bandpass filters play an important role in numerous delay-constrained signal processing applications, especially in the area of 50/60 Hz power systems instrumentation. In these applications distorted line voltages or currents should be filtered without delaying the fundamental frequency component. Vainio and Ovaska introduced the multiplicative general parameter (MGP) finite impulse response (FIR) filtering scheme in [1] and [2]. Since the line frequency tends to vary within a constrained interval, typically ±2%, adaptive filters should be used. In MGP-FIR the adaptation is achieved through adjusting the two MGPs. The coefficient values of the basis filter do not change during the adaptation process. An MGP-FIR is designed in the following way. First we optimize the basis filter either by applying traditional hard computing methods, or, as this paper presents adaptive genetic algorithms [5]. Next, the MGP-FIR is used in the actual application, and fine-tuning is left to the multiplicative general parameters. The better the basis filter attenuates unwanted components and the better the prediction capabilities of the basis filter are, the easier it is for the MGP filter to adapt to the changing characteristics of the input signal. Digital filters have been designed before using genetic algorithms, for example in [8] [9], but MGP-FIRs have never been designed using adaptive genetic algorithms. Thus, the speedup in convergence of the GA in this application using adaptive parameters was yet to be studied.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1162–1176, 2004.

The paper is organized as follows. Section 2 describes the multiplicative general parameter approach to filtering. Section 3 explains the basics of genetic algorithms and defines their use in context with MGP filtering. Section 4 introduces the obtained results. Section 5 includes concluding remarks and Section 6 presents the paths of our future research work.

2 Multiplicative General Parameters

The MGP filtering method sets low computational requirements to the implementation platform, while it simultaneously provides effective and robust filtering of disturbances. All this is achieved without delaying the primary signal component, and maintaining adaptation capabilities around the nominal frequency.

In a typical MGP-FIR, the filter output is computed as

$$y(n) = g_1(n)\sum_{k=0}^{N-1} h_1(k)x(n-k) + g_2(n)\sum_{k=0}^{N-1} h_2(k)x(n-k). \tag{1}$$

Where $g_1(n)$ and $g_2(n)$ present the MGP parameters, and $h_1(k)$ and $h_2(k)$ are the fixed coefficients of an FIR basis filter. Taking this into account, the coefficients of the composite filter are $\theta_1(k) = g_1(n)h_1(k),\ k \in \lfloor 0,1,...,N-1 \rfloor$, for the first MGP, and $\theta_2(k) = g_2(n)h_2(k),\ k \in \lfloor 0,1...,N-1 \rfloor$, for the second MGP. An example of MGP-FIR with N=4 is shown in Fig. 1. Here N denotes the filter length.

The filter coefficients in the adaptation process are updated as follows.

$$g_1(n+1) = g_1(n) + \mu e(n)\sum_{k=0}^{N-1} h_1(k)x(n-k) \tag{2}$$

$$g_2(n+1) = g_2(n) + \mu e(n)\sum_{k=0}^{N-1} h_2(k)x(n-k). \tag{3}$$

Where μ is the adaptation gain factor and $e(n)$ is the prediction error between the filter output and the training signal. The MGP-FIR has two adaptive parameters to adapt only to the phase and amplitude of the nominal frequency. More degrees of freedom would allow the filter to adapt to undesired properties, such as the harmonic frequencies. The training signal $s(t)$ is defined as

$$\begin{aligned}
s(n) &= \sin(2\cdot\pi\cdot 49\cdot n) + \sum_{m=3,5,7,...}^{15} 0.1\cdot\sin(2\cdot\pi\cdot m\cdot 49\cdot n) + 0.004\cdot r(n),\ 0 < n \le 300 \text{ samples} \\
s(n) &= \sin(2\cdot\pi\cdot 50\cdot n) + \sum_{m=3,5,7,...}^{15} 0.1\cdot\sin(2\cdot\pi\cdot m\cdot 50\cdot n) + 0.004\cdot r(n),\ 300 < n \le 600 \text{ samples} \\
s(n) &= \sin(2\cdot\pi\cdot 51\cdot n) + \sum_{m=3,5,7,...}^{15} 0.1\cdot\sin(2\cdot\pi\cdot m\cdot 51\cdot n) + 0.004\cdot r(n),\ 600 < n \le 900 \text{ samples.}
\end{aligned} \tag{4}$$

$r(n)$ denotes a uniformly distributed random value between -1 and 1. The duration of the training signal is 900 samples. This signal is divided into three parts, each of which contains 300 samples. These training signal blocks correspond to frequencies of 49 Hz, 50 Hz, and 51 Hz. The training signal constitutes thus of the nominal fre-

quency sinusoid, odd harmonics up to the 15^{th} with amplitudes 0.1 each, and white noise. The training signal is similar to that used in [1] and [2], and it mimics the line signal (voltage/current) with varying fundamental frequency, harmonics, and noise.

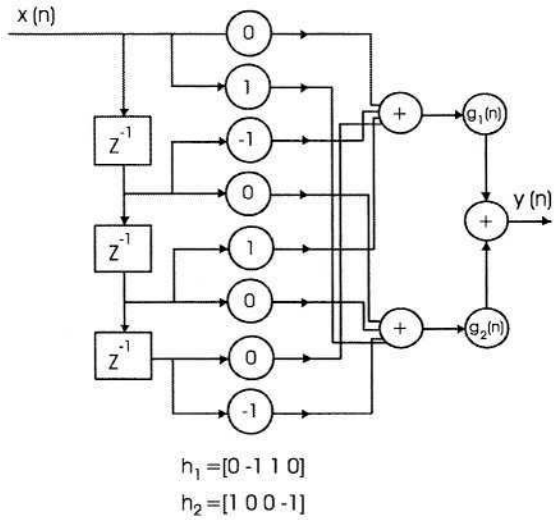


Fig. 1. An example of MGP implementation, where N=4. Signal values $x(n-1)$ and $x(n-2)$ are connected to the first MGP and values $x(n)$ and $x(n-3)$ are connected to the second MGP with filter coefficients -1,1,1, and -1 respectively

The basic idea of MGP-FIR filters is that all the samples of input delay line should be connected either to the first or to the second MGP, and no value should be left unused. Computational efficiency of these particular MGP filters arises from the fact that the filter coefficients are either -1, 0, or 1. Thus the number of multiplications in filtering operations is radically reduced compared to a normal filtering operation using more general coefficient values. Especially in VLSI or other hardware implementations, the number of multiplications, as well as other operations should be kept at minimum to guarantee fast operation and low cost of implementation.

3 Genetic Algorithms

Genetic algorithms model the nature's way of optimizing the characteristics of a system to comply with the demands of the surrounding environment. Genetic algorithms manipulate the possible problem solutions using selection methods, crossover and mutation operators, and survival strategies [3] [4] [7].

In this application, the cross connections and coefficients of the basis filter are to be optimized using a GA. Using non-evolutionary methods would limit the set of tools to using exhaustive search, since no gradient information is available in this problem domain. These filter tap cross connections and coefficients are modeled as

chromosomes in the following way. There are two separate vectors in one chromosome: the first vector, $h_1(n)$, of the chromosome represents the weights of the input samples connected to the first MGP, and the second vector, $h_2(n)$, corresponds to the weights of the input samples connected to the second MGP. The weights can be -1, 0, or 1. -1 and 1 in a vector indicate that this particular input value is associated to the corresponding MGP. 0, on the other hand, determines that a particular input value is not associated with the MGP in concern, rather it is connected to the other MGP. An example of a chromosome is presented in Table 1.

Table 1. Example of the the chromosomes, N=40. h_1 and h_2 associate the input signal values to the first and second MGPs, respectively

h_1	1 1 -1 0 0 1 -1 0 1 1 0 -1 -1 0 1 0 -1 -1 0 1 0 0 0 1 1 0 -1 0 -1 0 1 -1 0 0 0 -1 0 1 1 0
h_2	0 0 0 1 -1 0 0 1 0 0 -1 0 0 1 0 -1 0 0 -1 0 1 -1 1 1 0 0 1 0 1 0 -1 0 0 1 1 1 0 -1 0 0 -1

In [2] Ovaska and Vainio stated that satisfactory results can be achieved using an initial population, the size of which is twice the number of the filter length. Since the filter length N in this case is 40, the initial population size was set to be 80.

In this paper, we compare five different genetic algorithms. The first one is a simple genetic algorithm with no adaptive parameters. The second genetic algorithm applies adaptive mutation and crossover probabilities as presented in [5]. The third genetic algorithm features also the adaptive mutation and crossover scheme, but with different parameters than in [5]. In the fourth genetic algorithm, we used adaptive μ (see Eqs. (2) and (3)). The fifth algorithm used modified probabilities of [5], adaptive μ, and seeding.

The fitness of the chromosomes is evaluated using the fitness function:

$$fitness = \frac{K}{\max(NG49, NG50, NG51) \cdot (ITAE49 + ITAE50 + ITAE51)}. \tag{5}$$

K is assigned a value of 10^7 to scale the output of the fitness function to be expressed in thousands. Terms $NG49$, $NG50$, and $NG51$ represent the noise gain at a specific stage of the test input signal. These terms were added to the fitness function to attenuate noise. The noise gain is calculated as

$$NG(n) = \sum_{k=0}^{N-1} [g_1(n) \cdot h_1(k)]^2 + \sum_{k=0}^{N-1} [g_2(n) \cdot h_2(k)]^2. \tag{6}$$

$g_1(n)$ and $g_2(n)$ represent the first and second MGP at the end of a certain frequency period, respectively, whereas $h_1(n)$ and $h_2(n)$ denote the filter coefficients associated to the corresponding MGPs. In other words, $NG49$ is calculated using the MGP values after 300 samples, $NG50$ and $NG51$ are calculated after 600 and 900 samples, respectively, the frequency of the training signal changing every 300 samples.

$ITAE49$, $ITAE50$, and $ITAE51$ stand for the Integral of Time-weighted Absolute Error (ITAE) for each of the three signal parts, respectively. These terms were added to the fitness function to smoothen the adaptation of the filter to the varying input signal characteristics. The ITAE is calculated as follows.

$$ITAE = \sum_{n=1}^{M} n \cdot e(n). \tag{7}$$

n is the sample index, and M stands for the sample number at the end of a specific frequency period, in this case 300, 600 and 900. $e(n)$ is the error between the output of the system and the pure input signal without any harmonics or noise. The GA may in some cases create filters with unreasonably low noise gains, leading thus to very high fitness value of chromosomes. Theoretical minimum limits can be calculated that have to be passed in order the filter to be practical [10]. If these limits were not met, a penalty term was added to corresponding chromosomes' fitness value so that it was very likely eliminated from the next generation.

The genetic algorithms were given 1000 generations to converge to a solution, and each genetic algorithm was run 20 times to get some statistical reliability. Elitist mutation scheme was applied in each case, so that the best chromosome was never mutated. A single simulation run of 1000 generations using initial population of 80 chromosomes and filter length of 40, took 70-80 minutes to run on a Dell computer equipped with 2.66 GHz Intel Pentium 4 processor, 512 MB of memory, and MATLAB 6.5.0 software.

3.1 Reference Genetic Algorithm

To create a reference point, we implemented a simple genetic algorithm without any fine tuning to solve the problem. The algorithm starts by creating the initial population. The probabilities of assigning -1, 0, or 1 to a particular gene in the chromosome related to the first MGP are 1/3 all. In the case of assigning -1 or 1 to a gene in $h_1(n)$, 0 is assigned to this same gene in $h_2(n)$. This is because one input sample can be connected to only one MGP, not both. If 0 is assigned to a gene in the vector related to the first MGP, -1 or 1 is assigned to a gene related to the second MGP with the probability of 1/2 for both -1 and 1.

After the generation of the initial population, the chromosomes are arranged in descending order according to their fitness values. Mating is based purely on the rank of the chromosomes: the best and the second best chromosomes act as parents for two offspring. The offspring are created using single-point crossover, the crossover point of which is selected randomly. Also, the third and fourth best chromosomes constitute parents for two offspring, and so on. Altogether, 40 best chromosomes produce 40 offspring. Thus the 40 chromosomes with the lowest fitness scores are replaced by the offspring of the 40 best chromosomes. Also the parents remain within the population thus keeping the population size constant at 80 chromosomes.

The convergence behavior of this reference GA can be seen in Fig. 2. This figure shows clearly how the simple genetic algorithm converges in a typical optimization run. Maximum and average fitness values for each generation are well separated, and the minimum fitness lies very close to zero. This simple GA does not fully converge during the 1000 generations it was allowed to operate. Mutation probability of 5% and μ value 0.002 were used in this GA.

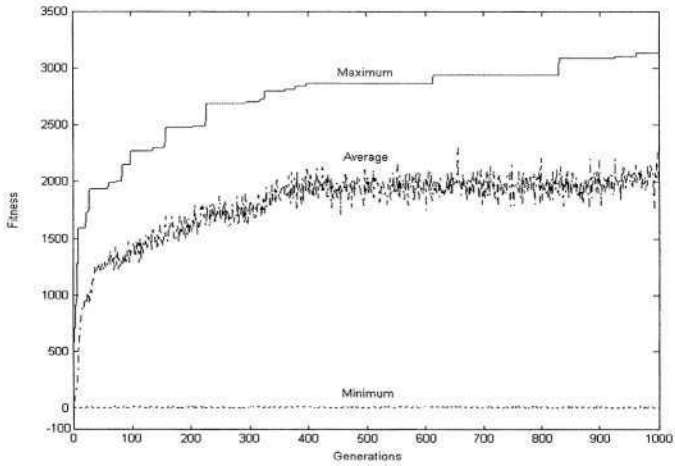


Fig. 2. Typical convergence behavior of the reference GA

3.2 Adaptive Genetic Algorithm

Optimal parameters for a GA may change during the iteration process and thus it is usually difficult to choose initial parameters that produce satisfactory results. Adaptive GA (AGA) parameters have been studied by Srinivas and Patnaik in [5]. In their approach, the mutation probability, p_m, and the crossover probability, p_c, are adjusted as follows.

$$p_c = \frac{k_1(f_{\max} - f')}{f_{\max} - f_{ave}}, \quad f' \geq f_{ave} \tag{8}$$

$$p_c = k_3, \quad f' < f_{ave} \tag{9}$$

$$p_m = \frac{k_2(f_{\max} - f)}{f_{\max} - f_{ave}}, \quad f \geq f_{ave} \tag{10}$$

$$p_m = k_4, \quad f < f_{ave}. \tag{11}$$

k_1, k_2, k_3, and k_4 represent coefficients with values equal or less than one. f_{max} denotes the maximum fitness value of a generation, whereas f_{ave} denotes the average fitness value of the same generation. f' denotes the higher fitness value of a parent pair. The authors of [5] propose values 1.0, 0.5, 1.0, and 0.5 for k_1, k_2, k_3, and k_4 respectively.

Using this approach, the crossover probability decreases as the fitness value of the chromosome approaches f_{max} and is 0.0 for a chromosome with fitness value equal to f_{max}. Also, mutation probability approaches zero as the fitness of a chromosome approaches f_{max}. μ value 0.002 was also used in this GA.

The convergence behavior of this AGA can be seen in Fig. 3. This algorithm converges rapidly and the maximum value stays the same for the rest of the time. The maximum is however local, since the fitness value is smaller than with the reference GA. Maximum, average, and minimum fitnesses are the same after about 70 generations.

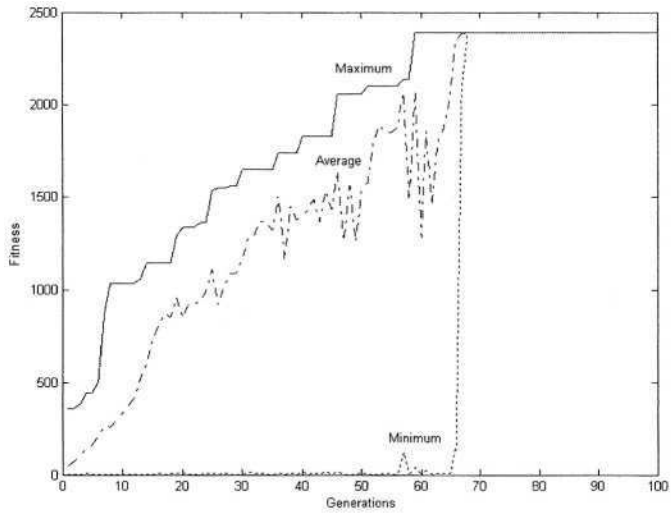


Fig. 3. Typical convergence behavior of the AGA

3.3 Adaptive Genetic Algorithm with Modified Probabilities

The values for k_1, k_2, k_3, and k_4 presented in [5] are not obviously appropriate for all applications. Referring to the "no free lunch" theorem discussed by Wolpert and Macready in [6] and Fogel in [7], we decided to give value equal to one for each of the k_1, k_2, k_3, and k_4, because the values presented in [5] do not perform so well in this application. This way we may prevent premature convergence, which seemed to be a problem in our application, when 1.0, 0.5, 1.0, and 0.5 were assigned to k_i, i = 1,2,3,4, respectively. Since elitist mutation saves the best solution from corruption, we apply the mutation probability of 1.0 to all but the best chromosome. This way, although possibly near the optimal solution, we still search efficiently the surroundings for a better one. Typical convergence behavior of this AGA can be seen in Fig. 4. This GA converges to the maximum rather slowly, while the minimum stays close to 0 all the time.

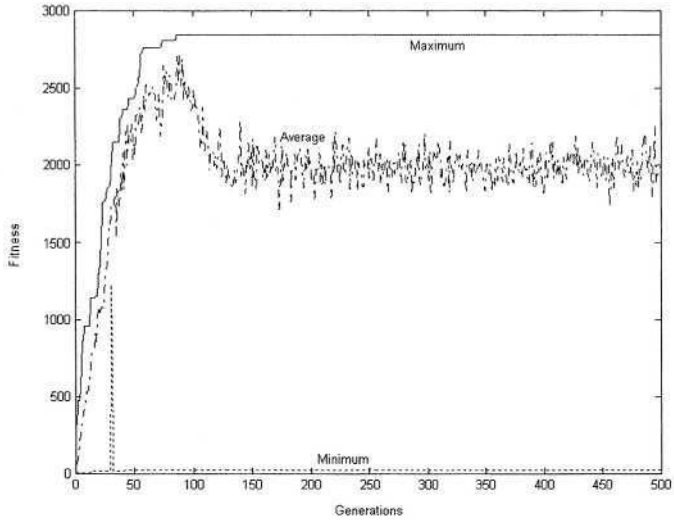


Fig. 4. Typical convergence behavior of the AGA using modified probabilities

3.4 Adaptive Genetic Algorithm with Modified Probabilities and the μ-Term

In the previously presented GAs we used a constant adaptation gain factor μ (see Eqs. (2) and (3)). Since the characteristics of the input signal vary with time, it is difficult to determine the optimal parameters of the system before the optimization process. Adaptive μ was applied in order to speed up the convergence of genetic algorithm.

The chromosome was modified so that one gene was added to present the μ value. The μ values in the initial population were chosen randomly so that $0 < \mu < 0.009$. Values larger than this can easily make the filter unstable. In crossover the offspring μ value, μ_o, was calculated as follows.

$$\mu_o = r \cdot \mu_{p1} + (1-r) \cdot \mu_{p2} . \tag{12}$$

r stands for a random number between 0 and 1. μ_{p1} and μ_{p2} are the μ values of the first and second parents, respectively. Mutation was applied as follows.

$$\begin{aligned} \mu_m &= \mu + 0.1\mu, r < 0.5 \\ \mu_m &= \mu - 0.1\mu, r \geq 0.5 . \end{aligned} \tag{13}$$

μ_m presents the value of μ after mutation, and r is a uniformly distributed random number between 0 and 1. An example of the converged μ values and corresponding fitnesses are shown in Table 2. The average converged μ value of the 20 runs was 0.0034. This value can be considered as the practical maximum μ value. Values larger

than 0.0034 could easily cause the filter to be unstable, whereas values below 0.0034 merely slow the convergence of the GA down.

Table 2. Example of converged μ values an correponding fitnesses. Each run was done using 1000 generations

Run	Converged μ	Best fitness
1	0.0034	3420
2	0.0025	3069
3	0.0032	3735
4	0.0030	3225
5	0.0042	3387
6	0.0044	2962
7	0.0034	2877
8	0.0042	3249
9	0.0029	3504
10	0.0036	3052
11	0.0029	3541
12	0.0033	3447
13	0.0037	3761
14	0.0026	3606
15	0.0039	3481
16	0.0040	3215
17	0.0037	3304
18	0.0026	3776
19	0.0028	3772
20	0.0026	3722
Average	0.0034	3405

The convergence of this AGA with modified probabilities of [5] and adaptive μ is shown in Fig. 5. This GA converges quite rapidly, and the average fitness follows quite close the maximum fitness. The minimum fitness values vary a lot, because μ is a sensitive parameter and even small changes can easily make the filter unstable resulting in a low fitness value.

3.5 Adaptive Genetic Algorithm Using Seeding

When analyzing the chromosomes produced by the GA mentioned above, one could see a systematical structure in the converged solutions. All the GAs had a tendency of creating blocks of consecutive equal gene values within a chromosome. An example of a converged chromosome can be seen in Table 3.

This kind of repeating structure was present in nearly all the converged chromosomes. Bearing this in mind, seeding was applied to the GA while generating the initial population. Seeding means applying the information of the structure of the solution to the generation process of the initial set of chromosomes. Seeding was imple

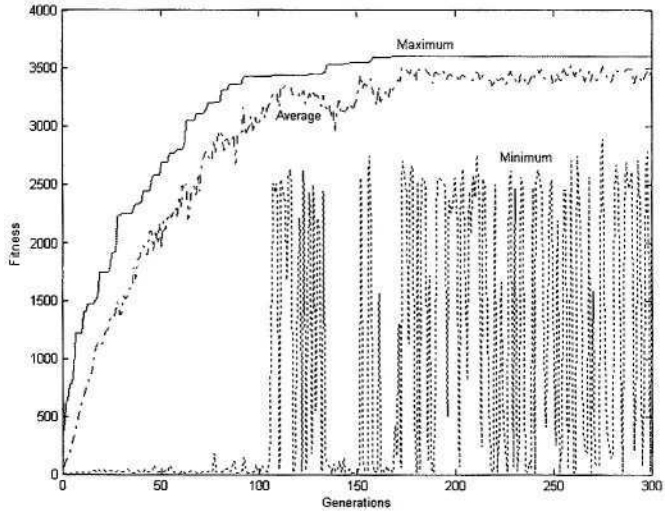


Fig. 5. Typical convergence behavior of the AGA using adaptive μ

Table 3. Example of converger chromosomes. Note the block like structure of the genes

h_1	0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 -1 -1 -1 -1 -1 -1 -1 -1 -1 0 0 0 0
h_2	-1 1 1 1 -1 1 1 1 -1 0 0 0 0 0 0 0 0 0 0 -1 -1 -1 -1 -1 -1 -1 -1 1 0 0 0 0 0 0 0 0 0 -1 1 1 1

mented so that the chromosomes in the initial population we formed of blocks of random number of consecutive equal gene values. The block length was set to be 1 at the minimum and the other extreme was a chromosome constituting only of a single gene value.

The typical convergence behavior of this GA is presented in Fig. 6. The convergence rate is rather fast. Variations in minimum fitness are explained by the adaptive μ, since the filter can easily become unstable if the μ value changes too drastically.

4 Results

The performance of the different GA approaches in MGP filter design are reported in Table 4. Each GA took about 70-80 minutes to go through 1000 generations. However, there were significant differences in the convergence properties of the algorithms. The adaptive GA proposed in [5] was clearly the fastest to converge, but the algorithm was very likely trapped in a local minimum, and it thus behaved worse in statistical sense than the simple GA used for reference.

Modifying the probabilities of the AGA produced fitness average closer to the reference GA than with original parameters, while simultaneously reducing the number of required generations to one tenth. Adding the adaptive μ term succeeded in

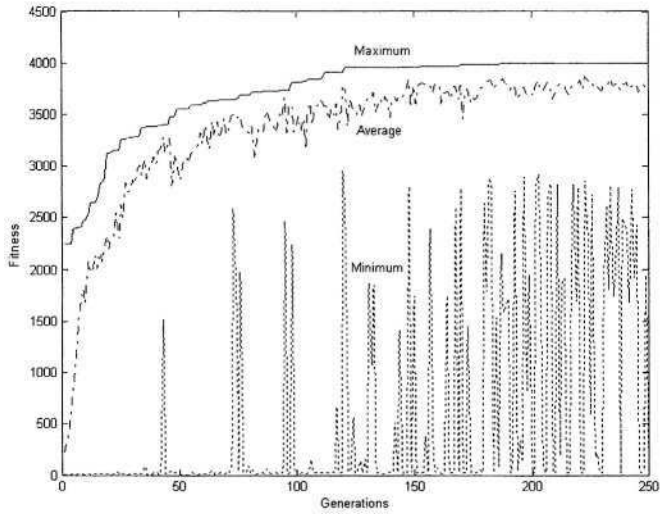


Fig. 6. Typical convergence behavior of the AGA using modified probabilities, adaptive μ, and seeding

achieving higher average fitness, while simultaneously increasing the number of required generations. Applying seeding helped the adaptive μ enhanced AGA to produce best results, indicating highest average fitness with even better convergence characteristics as the AGA with adaptive μ but without seeding. Standard deviations presented in Table 4 indicate that the AGA with modified probabilities, adaptive μ, and seeding converges to high fitness solutions rather reliably.

The time to achieve the best results was 11.5 minutes, which is about 18% of the time required by the reference GA to produce results. The gained speedup makes the introduced method a tempting tool for practicing engineers. The average time to go through 1000 generations is almost the same for all the GAs. Then again, the differences between the average convergence times are notable.

Table 4. Performance comparison of the featured Gas. The results are averages of 20 runs

Genetic Algorithm	Average Best Fitness / Standard Deviation	Convergence (generations) / Standard Deviation	Time (mins)	Convergence Time (mins)
Simple GA	3249 / 324	837 / 130	76.4	63.9
AGA	1472 / 505	21 / 11	77.9	1.6
AGA, Modified Probabilities	2824 / 557	89 / 34	76.7	6.8
AGA, Modified Probabilities and μ	3638 / 287	329 / 218	73.9	24.3
AGA, Modified Probabilities, μ, seeding	3617 / 329	156 / 65	73.6	11.5

The best converged chromosome had fitness value of 3985 and was achieved using AGA with adaptive μ and seeding. This chromosome is presented in Table 5.

Table 5. The best converged chromosome

h_1	1 1 1 0 0 0 0 0 0 0 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
h_2	0 0 0 -1 -1 -1 -1 -1 -1 -1 -1 0 0 0 0 0 0 0 0 0 0 0 1 -1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 -1 -1 -1 -1

Fig. 9 describes the signals before and after filtering. The evaluation signal *eval(n)* used is defined as follows.

$$eval(n) = \sin(2 \cdot \pi \cdot 49 \cdot n) + \sum_{m=3,5,7,\ldots}^{15} 0.15 \cdot \sin(2 \cdot \pi \cdot m \cdot 49 \cdot n),\ 0 < n \le 300 \text{ samples} \tag{14}$$

$$eval(n) = \sin(2 \cdot \pi \cdot 50 \cdot n) + \sum_{m=3,5,7,\ldots}^{15} 0.15 \cdot \sin(2 \cdot \pi \cdot m \cdot 50 \cdot n),\ 300 < n \le 600 \text{ samples}$$

$$eval(n) = \sin(2 \cdot \pi \cdot 51 \cdot n) + \sum_{m=3,5,7,\ldots}^{15} 0.15 \cdot \sin(2 \cdot \pi \cdot m \cdot 51 \cdot n),\ 600 < n \le 900 \text{ samples.}$$

The amplitude response of the filter presented in Table 5. after 600 samples of the evaluation signal can be seen in Figs. 7. This response shows how the GA tries to dampen the harmonic frequencies of the nominal components (i.e., 150 Hz, 250 Hz, ..., 750 Hz). The phase delay in samples is shown in Fig. 8. The phase delay is negative in the 50 Hz region, and thus the filter has the desired predictive capabilities. Table 6 shows the calculated total harmonic distortion (THD) values for each of the training signal nominal frequency components before and after filtering. After filtering the THD values as well as the amplitudes of the harmonics are about a tenth of the values before filtering.

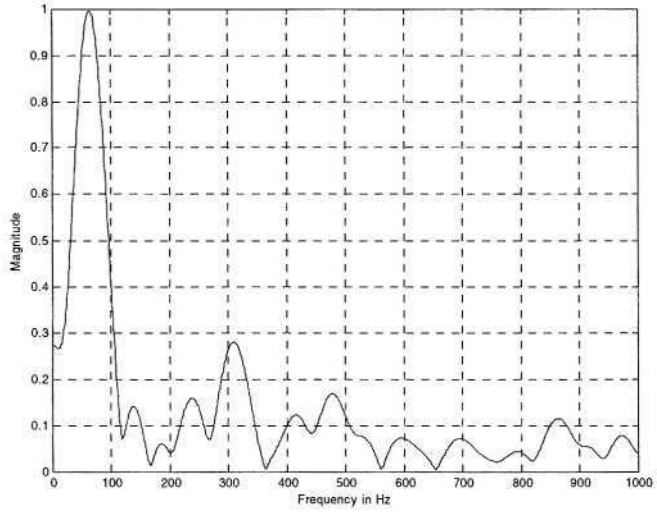


Fig. 7. Instantaneous magnitude responses of the MGP-FIR

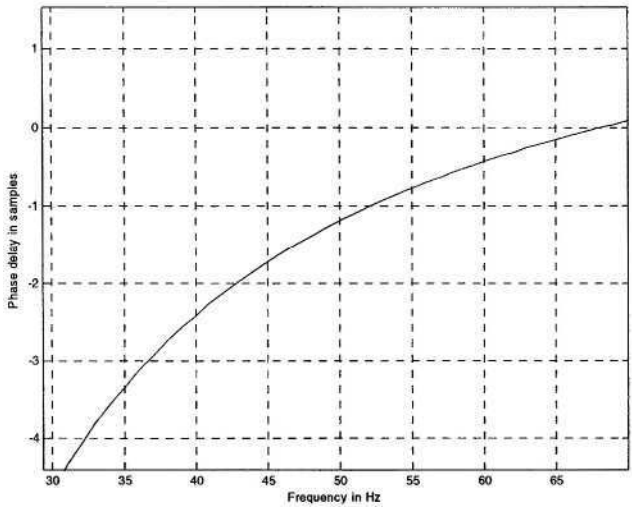


Fig. 8. Phase delay of the MGP-FIR. The phase delay in samples is negative in the 50 Hz region, as it should be in predictive filters

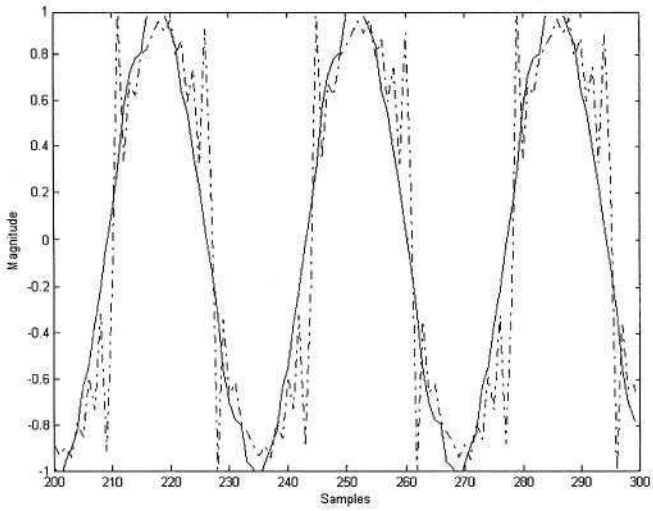


Fig. 9. Test signal defined in Eq. (13) (dash dotted) is filtered using the best MGP-FIR given in Table 5. Output (solid line) is almost sinusoidal

Table 6. Total harmonic distortions (THD) of the best MGP-FIR basis filter

	Input amplitude	Output at 49 Hz	Output at 50 Hz	Output at 51 Hz
Nominal frequency	1	0.87	0.93	0.88
3rd harmonic	0.15	0.018	0.016	0.011
5th harmonic	0.15	0.023	0.025	0.023
7th harmonic	0.15	0.018	0.009	0.008
9th harmonic	0.15	0.012	0.012	0.013
11th harmonic	0.15	0.010	0.008	0.006
13th harmonic	0.15	0.005	0.002	0.007
15th harmonic	0.15	0.005	0.004	0.007
THD	39.7%	4.4%	3.7%	3.6%

5 Conclusions

In this paper, we have presented a new way to design MGP-FIR basis filters. Adaptive genetic algorithms produced better results than genetic algorithms without any adaptive parameters. Knowing what to look for, or understanding the structure of the possible solution helped to apply seeding in the GA in order to further speed up the optimization process. Also, it is likely that using adaptive μ and seeding as parts of the design process had a greater impact on the final results than the use of adaptive mutation and crossover probabilities. MGP-FIRs were designed using evolutionary programming in [2], but the approaches used in the present paper make the GA converge more rapidly. Having said that, we conclude that the best proposed method excels among the others in MGP-FIR design methods, and should therefore be regarded as a competitive tool for experts working in the field of power systems instrumentation.

6 Future Work

Our future work will concentrate on experimenting how multiple population GAs would affect the performance of the optimization process. Also, since GAs are by nature well parallelizable, a parallel computing approach to GA aided design of MGP-FIRs will be explored.

References

1. Vainio, O., Ovaska, S.J., Pöllä, M.: Adaptive filtering using multiplicative general parameters for zero-crossing detection. IEEE Transactions on Industrial Electronics, Vol. 50, No. 6, Dec 2003, 1340-1342
2. Ovaska, S.J, Vainio, O.: Evolutionary programming in the design of adaptive filters for power systems harmonies reduction. IEEE International Conference on Systems, Man and Cybernetics, Vol. 5, Oct 2003, 4760-4766
3. Bäck, T.: Evolutionary Algorithms in Theory and Practice. Oxford University Press, New York, NY (1996)

4. Haupt, R. L., Haupt, S. E.: Practical Genetic Algorithms. John Wiley & Sons, New York, NY (1998)
5. Srinivas, M., Patnaik, L. M.: Adaptive probabilities of crossover and mutation in genetic algorithms. IEEE Transactions on Systems, Man and Cybernetics, Vol. 24, No. 4, Apr 1994, 656-667
6. Wolpert, D. H., Macready, W. G.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation, Vol. 1, No. 1, Apr 1997
7. Fogel, D.B.: Evolutionary Computation, Toward a New Philosophy of Machine Learning. IEEE Press, Piscataway, NJ (2000)
8. Wade, G., Roberts, A., Williams, G.: Multiplier-less FIR filter design using a genetic algorithm. IEE Proceedings of Vision, Image and Signal Processing, Vol. 143, No. 3, Jun 1996
9. Lee, A.; Ahmadi, M., Jullien, G.A., Miller, W.C., Lashkari, R.S.: Digital filter design using genetic algorithm. IEEE Symposium on Advances in Digital Filtering and Signal Processing, 1998, Jun 1998, 34-38
10. Vainio, O., Ovaska, S.J.: Noise reduction in zero crossing detection by predictive digital filtering. IEEE Transactions on Industrial Electronics, Vol. 42, No. 1, Feb 1995, 58-62

Reducing the Cost of the Hybrid Evolutionary Algorithm with Image Local Response in Electronic Imaging

Igor V. Maslov

Department of Computer Science, City University of New York / Graduate Center,
365 Fifth Avenue, New York, NY 10016, USA
ivm3@columbia.edu

Abstract. The paper focuses on the efficiency of the hybrid evolutionary algorithm (HEA) for solving the global optimization problem arising in electronic imaging. The particular version of the algorithm utilizes image local response (ILR), in order to reduce the computational cost. ILR is defined as the variation of fitness function due to a small variation of the parameters, and is computed over a small area. ILR is utilized at different stages of the algorithm. At the preprocessing stage, it reduces the area of the image participating in fitness evaluation. The correlation in the response space identifies the set of subregions that can contain the correct match between the images. Response values are used to adaptively control local search with the Downhill simplex method by applying a contraction transformation to the vector of the standard simplex coefficients. The computational experiments with 2D- grayscale images provide the experimental support of the ILR model.

1 Introduction

There are several problems in electronic imaging that can be approached in a unified manner, and formulated as a global optimization problem. Image registration, object recognition, and content-based image retrieval are commonly used in remote sensing, robotic vision, industrial control, as well as in many medical, security and defense applications. In image registration, two images are given, a reference image Img_0 and an image Img_1 subject to registration. A transformation has to be found that correctly maps points of Img_1 into the corresponding points of Img_0. In object recognition, a particular object with a signature (i.e. image) Img_1 has to be found in a scene Img_0 that contains many different objects. In content-based image search and retrieval, an original scene Img_0 has to be found in a large image database, such that Img_0 contains a query image Img_1.

All three aforementioned problems essentially attempt to find some transformation A providing the correct match between the images Img_1 and Img_0. If Img_1 and Img_0 are 2-dimensional grayscale images, then it is often convenient to evaluate the quality of the match by measuring the difference between the pixel values of the two images, after the transformation A has been applied to one of them (e.g. Img_1). Then the minimum value of the difference indicates a likely match between the images, and the corresponding parameters define the correct transformation. The problem of finding the correct transformation A can be formulated now as the problem of finding a feasible

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1177–1188, 2004.

vector of parameters V^* minimizing the difference F between the images Img_1 and Img_0, so that

$$F(V) > F(V^*), \text{ for all } V \quad V^* . \tag{1}$$

Without loss of generality the paper assumes that A is an affine transformation including image translation, rotation, and non-isotropic scaling, such that the transformed vector $p' = \{x',y'\}^T$ of the original coordinates $p = \{x,y\}^T$ of a pixel $P \in Img_1$ can be found as

$$p' = A(p) = SRp + T , \tag{2}$$

where the matrices S, R, and T denote scaling, rotation, and translation, respectively [1]. Vector $V = \{DX, DY, \theta, SX, SY\}$ defines the transformation A, and has five components: translations DX and DY along the x- and y-axis, rotation angle θ, and scaling factors SX and SY along the x- and y-axis. Matrices R and T correspond to the rigid body motion, while matrix S describes the local deformation of the image when the scaling factors are non-isotropic, i.e. $SX \neq SY$.

The difference F between the images is defined as the squared difference of the gray levels of the two images divided over the squared area of their intersection, i.e.

$$F = \frac{\sum_{\Omega} (g_1(x',y') - g_0(x,y))^2}{\Omega^2} , \tag{3}$$

where $g_1(x',y')$ and $g_0(x, y)$ are the gray levels of the images Img_1' and Img_0, respectively, and Ω is the area of their intersection [1].

The difference F has to be computed numerically for every trial parameter vector V. If the reference image Img_0 is a multi-object, cluttered and noisy scene, and the object Img_1 is significantly distorted during the transformation A, the problem (1)-(3) becomes a nonlinear, multimodal global optimization problem. Moreover, in many imaging applications the difference F is a non-convex function, which causes gradient-based optimization methods to fail or perform poorly [2]. Stochastic approach to global search and optimization has proved to be a successful alternative to classical optimization methods in solving real world problems [3], [4]. In particular, a hybrid version of the evolutionary algorithm is used in the paper to solve the optimization problem (1)-(3) for 2-dimensional grayscale images under the affine transformation [5], [6]. According to the evolutionary terminology, the vector $V = \{DX, DY, \theta, SX, SY\}$ is called a chromosome, and the function F corresponding to V is called the chromosome's fitness [7], [8], [9]. The algorithm concurrently works with a population of chromosomes $\{V_i, i = 1,\ldots, P_v\}$, and attempts to find a chromosome V^* that has the minimum value of its fitness $F(V^*)$. During the search, the algorithm uses genetic operators of selection, crossover, and mutation. Local neighborhood search is

utilized to improve and refine a set of the best (i.e. fittest) chromosomes found during the global evolutionary search.

One of the important problems arising in practical application of the hybrid evolutionary algorithm is associated with its relatively high computational cost. Different approaches have been suggested in the literature including the reduction of the total population size and the fraction of the population subjected to local search, the use of the gradient-based local optimization techniques, the reduction of the search space, various self-adaptation techniques, etc. [5], [8], [9]. The purpose of this paper is to introduce a consistent approach that can be used throughout the entire algorithm, either in addition to other conventional methods, or as a particular form of their expansion. The approach is based on the concept of image local response, and focuses on the problem of reducing the computational cost associated with solving the global optimization problem (1)-(3) arising in electronic imaging applications.

The paper is organized as follows. The concept of image local response (ILR) is introduced in section 2. Sections 3 through 5 discuss the usage of ILR at different stages of the hybrid evolutionary algorithm, namely in fitness evaluation (section 3), selection and crossover (section 4), and in local search (section 5). Computational experiments are presented in section 6. Section 7 concludes the paper.

2 Definition of Image Local Response

Image local response is defined here as the variation of the objective (fitness) function caused by a small variation of the parameter vector [10]. Response is computed over a small pixel area called the response area ω. For convenience and without loss of generality, the response area is chosen as a square with a side r called the radius of ω. The following procedure computes ILR at a base point P:

- Partial affine transformations are applied to the response area ω near P, such that each transformation corresponds to a unit variation of one of N components of the vector V (here *DX, DY,* θ, *SX,* and *SY).*
- For each affine transformation, the partial difference F_i between the initial ω and the transformed ω' areas is computed.
- The local response R_P at point P is computed as the averaged sum of all N partial differences.

Response R_P has the following important properties:

- R_P is nearly inversely proportional to the distance d from the base point of the response area ω over which it is computed, i.e. $R_P \approx f(1/d)$. As the distance d increases, the value of the response very rapidly decreases.
- If two points P and Q have similar gray level distributions, the difference between their respective responses R_P and R_Q is small.
- If the gray level near the base point P has a significant change (e.g. near the object edge), the response R_P has the corresponding increase in its value.

Image local response can serve as an indicator of the smoothness of the fitness function F (i.e. fitness landscape) on a micro level, in a small locality near its respec-

tive base point. If the landscape is smooth, the change of ILR in the locality is small, while any significant change of F causes the corresponding significant change of the ILR value. If R_P is known, one can define a "deformation" of the local area caused by R_P. If the response is nearly flat, the deformation of the area is small; when the response grows, the area correspondingly shrinks. An approximate one-dimensional model suggests the estimate of the deformation ε of the linear segment enclosed between the image points P and Q caused by their responses R_P and R_Q in the following form [10]:

$$\varepsilon = \ln(R_P / R_Q) . \tag{4}$$

Since the line segment experiences the deformation of shrinking, the following condition holds:

$$0 < (1 - \ln(R_P / R_Q)) \quad 1 . \tag{5}$$

The approximate model of the linear deformation is used in section 5 to derive the adaptive control mechanism for local search with the Downhill simplex method.

3 Cost Reduction in Fitness Evaluation

The total computational cost TC of the hybrid evolutionary algorithm (HEA) includes the cost of fitness evaluations, and the overhead related to various evolutionary and bookkeeping operations, i.e.

$$TC = \sum_{j=1}^{J} \sum_{i=1}^{G_j} e_{ij} t_i + O(J, G) , \tag{6}$$

where J is the total number of generations, G_j is the total number of chromosomes in the j–th generation, e_{ij} is the number of fitness evaluations for the chromosome $\boldsymbol{V}_i$ in the j–th generation, t_i is the time required for one fitness evaluation, and the term $O(J,G)$ is the overhead associated with the total number of iterations J and the total number of chromosomes G.

In a typical imaging application, the most of the computational cost is attributed to fitness evaluations, i.e. to the first term in formula (6). A single fitness evaluation for a chromosome $\boldsymbol{V}_i$ includes the following three operations:

- transformation of the image $\boldsymbol{Img}_1$,
- pixel-wise comparison of the transformed image $\boldsymbol{Img}_1'$ with the original image $\boldsymbol{Img}_0$ of the scene,
- evaluation of the difference F between both images.

For the $M \times N$ - pixel image, each of the above operations has to be performed $M \times N$ times, so the reduction of the number of pixels participating in the evaluation would result in the significant reduction of the total cost of the algorithm. The effect of the reduction can be estimated using the area reduction factor Φ_Ω defined as

$$\Phi_\Omega = \frac{\Omega'}{\Omega}, \tag{7}$$

where $\Omega = M \times N$, and Ω' is the reduced image area due to the reduced number of pixels for fitness evaluation.

The equivalent number of evaluations C_e corresponding to the full image is defined as

$$C_e = \Phi_\Omega C_r, \tag{8}$$

where C_r is the number of fitness evaluations with the reduced area Ω'. Image local response can be used for the reduction of the number of pixels participating in fitness evaluation.

Image local response identifies those segments of the image that change the most during its transformation, i.e. ILR extracts the important feature of the image, its dynamic contents. After computing ILR, the total area Ω of the image can be represented as the sum of the dynamic contents Ω_D and the static contents Ω_S.

The dynamic area Ω_D will contribute the most to the fitness evaluation for the transformed image Img_1. The effect of the static area is not significant, and can be neglected during the beginning of the search. The following procedure for pixel reduction in fitness evaluation can be formulated now:

- The $M \times N$ matrix M_R of image local response for Img_1' is computed during the preprocessing stage.
- The response threshold T_R identifying the dynamic contents Ω_D is chosen, such that image segments (pixels) with the response values below T_R are temporarily excluded from the fitness evaluation.
- The $M \times N$ bit mask of the image is formed based on M_R. Image segments that have response values below T_R correspond to 0s in the mask; the segments that have response values above T_R correspond to 1s in the mask.
- The bit mask is used at the beginning of the search, and fitness F is computed only over those segments of the image that correspond to 1s in the bit mask.
- In the process of the evolutionary search, the procedure computing F switches to the full image evaluation when fitness of the best chromosome falls below some pre-set fitness threshold T_F.

4 Cost Reduction in Selection and Crossover

The selection mechanism plays the key role in evolutionary search guiding it toward the fitter fraction of the chromosome population. It is aimed at creating more copies of higher-fit candidate solutions, i.e. chromosomes with lower values of fitness function (in minimization problem).

When the best match between images Img_1' and Img_0 has to be found, the efficiency of the selection mechanism can be increased with the help of the likelihood matrix M_p defined as follows. The size of the matrix M_p corresponds to the image size of the scene (i.e. image Img_0). The value of the element (i,j) corresponds to the likelihood of the vector V being at the point (i,j) of the image Img_0. If the image Img_0 was partitioned (segmented) into sub-regions corresponding to the objects in the scene, then the background would have nearly zero likelihood of the location of the solution. At the same time, the sub-regions corresponding to the objects would have fairly high probability values in M_p. Matrix M_p can be considered as a mask applied to the image, so that the background areas are eliminated, while the areas corresponding to the objects have the high probability of the location of the optimal solution.

Image local response can be used to compute the likelihood matrix M_p, according to the following procedure:

- The response matrices M_{REF} and M_{OBJ} are computed for the scene Img_0 and the object Img_1'.
- Correlation is applied to the matrices M_{REF} and (possibly scaled down) M_{OBJ}, so that the latter serves as the correlation kernel. In order to increase the signal-to-noise ratio, the correlation can be repeated with M_{OBJ} rotated e.g. by 90°.
- The result of the correlation (i.e. the modified matrix M_{REF}) is scaled to fit in the range (0,1).
- After the scaling, the resulting matrix M_p serves as the likelihood matrix identifying the potential sub-regions for selection and crossover. During the selection process, the modified quality F' of the parental chromosome is evaluated according to the following formula:

$$F' = F \cdot \exp[(1-p)^2], \tag{9}$$

where F is the actual fitness value, and p is the probability corresponding to the chromosome's entry in the likelihood matrix M_p. The exponential term in (9) plays a role of the penalty if the chromosome is in the sub-region of the low likelihood of the optimal solution: the penalty and the corresponding modified fitness F' exponentially grow as the probability p decreases.

5 Cost Reduction in Local Search

The particular version of the hybrid evolutionary algorithm described in this paper combines random search and the Downhill simplex method (DSM) to refine the solu-

tion [2]. Local search significantly improves the performance of evolutionary algorithm, but requires additional evaluations of the fitness function F, with most of them attributed to the DSM search. The number of additional evaluations can be reduced using ILR, as described below.

The Downhill simplex method is an iterative procedure maintaining a non-degenerate $(N+1)$-dimensional simplex in the N-dimensional search space [11], [12]. The vertices $(\boldsymbol{V}_1, \boldsymbol{V}_2, \ldots, \boldsymbol{V}_{N+1})$ of the simplex, together with the corresponding function values $(F_1, F_2, \ldots, F_{N+1})$, form a set of approximations to the N-dimensional parameter vector $\boldsymbol{V}$ and the objective function F, respectively. The algorithm attempts to change the worst vertex of the simplex to a better position, so the function value at the vertex would decrease. As the procedure moves from iteration to iteration, the simplex is continuously changing its shape and shrinking. The process terminates when either the size of the simplex or the difference between the function values becomes very small. The movement of the simplex is based on the ranking order of its vertices, and is controlled by the vector of four coefficients $\boldsymbol{\alpha} = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$. The coefficients define expansion or contraction of the simplex. The commonly used values of the coefficients are usually fixed at {-1, 2, 0.5, 0.5}. The number of additional fitness evaluations can be reduced by making the vector $\boldsymbol{\alpha}$ variable and adaptive to the local properties of the fitness function F in the vicinity of the simplex [10].

A contraction transformation $T(\boldsymbol{\alpha})$ is applied to the vector $\boldsymbol{\alpha}$ of the DSM coefficients between two points P and Q of an image, with the contraction coefficient C_{PQ} estimated using the values of the local responses R_P and R_Q at points P and Q, as follows:

$$C_{PQ} = (1 - \ln(R_P / R_Q)) . \tag{10}$$

Coefficient C_{PQ} satisfies the following properties:

- $C_{PQ} \approx 1$ for a smooth surface with the small variation of the local response, i.e. when $R_P \approx R_Q$,
- $C_{PQ} < 1$ for a rough surface with the large variation of the local response, i.e. R_P R_Q,
- C_{PQ} decreases as the roughness of the surface increases.

6 Computational Experiments

In order to validate the use of image local response at different stages of the hybrid evolutionary algorithm, computational experiments were conducted on a series of 2-dimensional grayscale images [13], [14]. Some results of the experiments are presented in this section.

The first experimental set estimated the computational cost reduction in fitness evaluation. Figure 1 shows a sample test image of a ship. It includes the 256×256-pixel reference image Img_0 of the scene (Figure 1a), and the 256×128-pixel image Img_1' of the object (Figure 1b) obtained by cropping a section from the scene. The im-

age Img_1 was transformed using the 5-dimensional vector $V = \{DX, DY, \theta, SX, SY\}$, where translations DX and DY, and rotation θ defined the location of the object in the scene, and non-isotropic scaling factors SX and SY defined the local deformation of the object. The object was stretched along the x-axis with the ratio $SX / SY = 2$. The problem of object recognition was formulated as the optimization problem of finding the optimal vector V^* minimizing the difference F between the images.

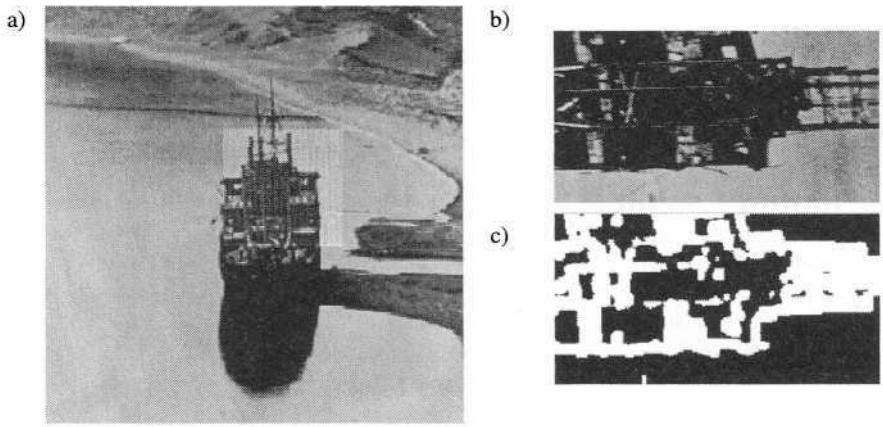


Fig. 1. Sample test image of a ship: a) scene Img_0 containing the object (indicated with the light box), b) transformed image Img_1' of the object, c) bit mask of the object after applying the response threshold $T_R = 4.0$

The response matrix M_R was computed for the object as described in sections 2 and 3 of the paper, and the response threshold $T_R = 4.0$ was applied to obtain the corresponding bit mask shown in Figure 1c. At the beginning of the evolutionary search, only pixels corresponding to 1s (colored black in Figure 1c) in the bit mask participated in the fitness evaluation. The fitness threshold T_F was set to $T_F = 0.19$. When the fitness value of the best chromosome fell below T_F, the evaluation procedure switched to the full 256×128 image Img_1'.

The area reduction factor Φ_Ω for the sample ship image was $\Phi_\Omega = 0.408$; the actual number C_r of fitness evaluations with the bit mask was $C_r = 1836$; the equivalent number of evaluations was $C_e = \Phi_\Omega C_r = 749$. The number of fitness evaluations C_f with the full image was $C_f = 1836$; the total number of evaluations C was $C = (C_f + C_e) = 6088$. For comparison, the number of fitness evaluations using the algorithm without image reduction was $C = 9119$. The use of ILR for the reduction of the area participating in the fitness evaluation resulted in the significant reduction of the computational cost associated with the evaluations. The total number of evaluations dropped from 9119 to 6088, constituting nearly 33% savings. The optimal value of the parameter vector was virtually the same, and very close to the exact value. Table 1 summarizes the findings of the experiment.

Table 1. Summary of experimental results for the sample test image of a ship

Parameters	***DX***	***DY***	***θ***	***SX***	***SY***
Exact values	108.0	144.0	1.57	3.56	1.78
Full image	108.7	146.2	1.57	3.52	1.74
Reduced image	109.1	146.9	1.57	3.51	1.79

Attributes	**Total number of generations**	**Total number of evaluations**	**Min. fitness**
Full image	20	9119	0.00876
Reduced image	17	6088	0.01154

The second experimental set evaluated the cost reduction in selection and crossover, as described in section 4 of the paper. Figure 2 shows a sample recognition problem for the distorted image of a boat (Figure 2b) in a scene (Figure 2a). The small boat object is located in the upper half of the scene, which is cluttered with bushes and trees.

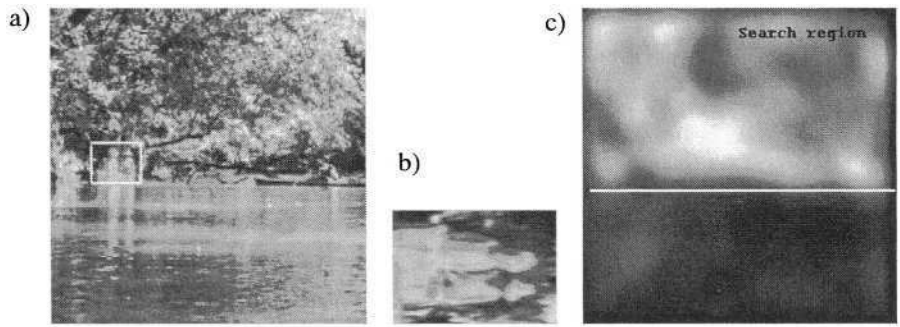


Fig. 2. A sample test image of a boat: a) image Img_0 of the scene with the indicated location of the object; b) image Img_1' of the object; c) correlation result in the response space with the indicated sub-region of the search

The presence of the vast water area with high-intensity reflections in the lower half of the image makes the recognition task particularly complex for the regular evolutionary algorithm. Following the selection pressure, the evolutionary search focuses on the sub-regions of the image that have lower values of the difference F between the scene and the object. Analysis shows that the chromosomes located in the water area have lower average values of F than the chromosomes located in the upper half of the image. This makes the regular EA focus the search in the water area. In the particular case of the boat image, the regular algorithm was terminated after 90 generations, with the search being trapped at one of the local minimum points in the lower half of the image. This type of problem is known as deceptive function in the EA theory [15]. For deceptive function, the intermediate partial solutions have low fitness values misleading the algorithm, and directing the search toward the areas that are away from the optimal solution.

The average local responses of the scene and the object were computed and correlated, with the response matrix M_{OBJ} of the object served as the correlation kernel. Correlation was applied twice, with the correlation kernel M_{OBJ} scaled down and rotated 90°. The result of the correlation of the response matrices M_{REF} and M_{OBJ} shown in Figure 2c was used as the likelihood matrix M_p defining the probability of finding the solution in different sub-regions of the image Img_0. The sub-regions in the correlation image that had higher intensity levels corresponded to the sub-regions with the higher probability of the optimal solution.

The correlation of the response matrices effectively lowered the probability of the search in the deceptive water area, so selection and recombination were limited to the upper half of the scene, where the boat object is located – see Figure 2a. The algorithm was able to find the object after 24 generations, with the optimal parameter vector $V^* = \{33, 144, 1.61, 3.71, 2.1\}$, while the exact solution was $V^* = \{26, 141, 1.57, 4.01, 2.0\}$. The analysis of the performance of the regular algorithm and its response-based modification shows that the regular version leads at the beginning of the search while exploring the deceptive water area; it stalls in one of the many minima located in that area. The modified version utilizes the probability matrix (shown in Figure 2c) to avoid the deceptive area, and successfully finishes the search with the nearly optimal configuration of the parameter vector.

The third set of computational experiments was designed to show that the transformation $T(\alpha)$ of the DSM coefficients described in section 5 of the paper efficiently controls the movement of the DSM simplex, and significantly reduces the number of additional fitness evaluations during the DSM search. A set of 100 runs for a number of test images was performed. For every test run, the initial simplex was placed in the proximity of the optimal solution using the following technique. The value of each component of the vector V for each of the six vertices was independently drawn at random with uniform probability from the (+10%) range of the corresponding domain centered at the component's optimal value. For example, the translation DX for a 256×256-pixel image has the domain range 0 – 255. Correspondingly, the value of DX for the image of a boat (Figure 2a) was drawn at random with uniform probability from the interval (26.0 ± 25.5), i.e. from the interval (0.5, 51.5).

The set of 100 runs was performed for the standard DSM coefficients, and using the proposed ILR-based modification. Table 2 shows the sample comparative results for the number of fitness evaluations for the regular and the modified versions of DSM, for the image of a boat. The standard values of the DSM coefficients required the most fitness evaluations 5204 over 100 runs. The use of the response coefficients in the DSM search significantly reduced the number of evaluations across all its measures: the mean, the standard deviation, the maximum, the minimum, and the sum over 100 runs. The overall reduction rate was 43.4%, which constituted significant savings in the computational cost of the local search. The interesting effect of the ILR-enhanced DSM search was the decrease of the variance of the number of fitness evaluations. It is reasonable to assume that smoothing occurred due to the averaging operation in the response value. Moreover, the reduction occurs virtually across all sample runs, and not just on the average.

Table 2. Number of fitness evaluations and minimum fitness value for the regular version of DSM (Reg), and its ILR-enhanced modification (Rsps), for the boat image

Value	Number of evaluations		Reduction rate, %	Minimum fitness value	
	Reg	Rsps		Reg	Rsps
Mean	52	29	44.2	0.025	0.031
St.dev	9.6	6.7	30.2	0.009	0.005
Max	90	47	47.8	0.035	0.042
Min	30	14	53.3	0.007	0.011
Sum	5204	2945	43.4	--	--

7 Conclusions

Image registration, object recognition, and content-based image retrieval are common problems in electronic imaging applications. The concept of global optimization provides a general and versatile framework for solving these problems. The hybrid version of evolutionary algorithm utilizing image response analysis is discussed in the paper. The algorithm solves the optimization problem in electronic imaging applications, i.e. the search for a proper transformation that provides the best match between two images.

In order to compute the unique characteristics of the object and the scene that are invariant to image transformation and distortion, the image transform is utilized in the form of image local response (ILR). The response values are computed for all image points participating in the evolutionary search, and used at different stages of the algorithm. The image transform is first applied at the pre-processing stage, to extract the dynamic contents of the images. The response adequately captures the dynamics of the image transformation, which makes it particularly well suited for the evolutionary search. The response matrix of the object is evaluated and used to reduce the area of the image participating in the fitness evaluation.

The correlation in the response space is applied then to both images, in order to reduce the search space, and to identify the set of sub-regions that can contain the potentially correct match between the sought object and the object in the scene. Correlated response matrices of the object and the scene form the likelihood matrix that limits the creation of new chromosomes to the sub-regions of the search space that most likely contain the optimal match. During the selection of parental chromosomes for crossover, their quality is estimated via the modified fitness. The latter contains the penalty term based on the probability that the chromosome is located in the area of the optimal match.

The particular model of HEA is used that alternates random search and the Downhill simplex method (DSM) for local search and refinement. Image response values are used to adaptively control the DSM simplex by applying a contraction transformation to the vector of the standard DSM coefficients. The technique correlates the movement of the DSM simplex with the local properties of the fitness function in the vicinity of the simplex.

Computational experiments with 2D grayscale images provide the experimental support and justification of the analytical model of image local response, and its utilization for the reduction of the computational cost of the hybrid evolutionary algorithm in electronic imaging. Moreover, the quality of the final solution does not degrade, in comparison with the regular version of HEA.

References

1. Brooks, R.R., Iyengar, S.S.: Multi-sensor Fusion: Fundamentals and Applications with Software. Prentice Hall, New York (1998)
2. Gertner, I., Maslov, I.V.: Using Local Correction and Mutation with Memory to Improve Convergence of Evolutionary Algorithm in Image Registration. In: Sadjadi, Firooz A. (ed.): Automatic Target Recognition XII. Proceedings of SPIE, Vol. 4726, SPIE (2002) 241-252
3. Guus, C, Boender, E., Romeijn, H.E.: Stochastic Methods. In: Horst, R., Pardalos, P.M. (eds.): Handbook of Global Optimization. Kluwer Academic Publishers, Dordrecht, The Netherlands (1995) 829-869
4. Ali, M.M., Storey, C., Törn, A.: Application of Stochastic Global Optimization Algorithms to Practical Problems. J. Optim. Theory Appl. **95** (1997) 545-563
5. Hart, W.E., Belew, R.K.: Optimization with Genetic Algorithm Hybrids That Use Local Search. In: Belew, R.K., Mitchel, M. (eds.): Adaptive Individuals in Evolving Populations: Models and Algorithms. Proceedings of Santa Fe Institute Studies in the Sciences of Complexity, Vol. 26. Addison-Wesley, Reading, MA (1996) 483-496
6. Joines, J.A., Kay, M.G.: Utilizing Hybrid Genetic Algorithms. In: Sarker, R., Mohammadian, M., Yan, X. (eds.): Evolutionary Optimization. Kluwer Academic Publishers, Boston, MA (2002) 199-228
7. Holland, J.H.: Adaptation in Natural and Artificial Systems. 2nd edn. MIT Press (1992)
8. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley (1989)
9. Michalewicz, Z., Fogel, D.B.: How to Solve It: Modern Heuristics. Springer-Verlag, Berlin New York (2000)
10. Maslov, I.V.: Improving Local Search with Neural Network in Image Registration with the Hybrid Evolutionary Algorithm. In: Priddy, Kevin L., Angeline, Peter J. (eds.): Intelligent Computing: Theory and Applications. Proceedings of SPIE, Vol. 5103. SPIE (2003) 166-177
11. Nelder, J.A., Mead, R.: A Simplex Method for Function Minimization. Comput. J., **7** (1965) 308-313
12. Wright, M.H.: Direct Search Methods: Once Scorned, Now Respectable. In: Griffiths, D.F., Watson, G.A. (eds.): Numerical Analysis: Proceedings of the 1995 Dundee Biennial Conference in Numerical Analysis. Addison Wesley Longman, Harlow, UK (1996) 191-208
13. http://lisar.larc.nasa.gov (public domain)
14. http://www.photolib.noaa.gov (public domain)
15. Goldberg, D.E.: Simple Genetic Algorithms and the Minimal Deceptive Problem. In: Davis, L. (ed.): Genetic Algorithms and Simulated Annealing. Hyperion Books (1987) 74-88

The Lens Design Using the CMA-ES Algorithm

Yuichi Nagata

Graduate School of Information Sciences,
Japan Advanced Institute of Science and Technology
nagatay@jaist.ac.jp

Abstract. This paper presents a lens system design algorithm using the covariance matrix adaptation evolution strategy (CMA-ES), which is one of the most powerful self-adaptation mechanisms. The lens design problem is a very difficult optimization problem because the typical search space is a complicated multidimensional space including many local optima, non-linearities, and strongly correlated parameters. There have been several applications of Evolution Algorithms (EA) to lens system designs, and Genetic Algorithms (GAs) are generally expected to be more suitable for these kind of difficult optimization problems than Evolution Strategy(ES) because GAs can provide a global optimization. We demonstrate, however, that a CMA-ES can work better than the GA methods previously applied to lens design problems. Experimental results show that the proposed method can find human-competitive lens systems efficiently.

1 Introduction

The lens system design involves very difficult optimization problems because the search space is typically a complicated multidimensional space including many local optima, non-linearities, and strongly correlated parameters. Modern lens system design is thus generally done with specialized GAD softwares that help designers visualize the lens, evaluate its quality, and locally optimize its variables. Several non-linear optimization methods are available for the local optimization of the lens system (the Newton method, the conjugate gradient method, and the steepest descent method, etc.), and the Damped Least Squares method [3] has been used frequently. But the design processes are highly dependent on the designer's experience in finding the proper starting points because the search space has many local optima. Several global optimization methods, such as the conventional bit-string genetic algorithms, have therefore recently been used in lens system design [3].

The lens system design problems can be formalized as real-valued function optimization problems by fixing the discrete variables such as refractive indexes. Ono et al. applied the real-coded genetic algorithms to lens system design problems [1]. The crossover operator they used was the unimodal normal distribution crossover (UNDX) [5] and their generation-alternation model was the minimal generation gap (MGG) [1] respectively, both of which had shown good performance in benchmark problems for real-valued function optimization.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1189–1200, 2004.

There are two kinds of evolutionary algorithms for the optimization of real-valued functions. One is the genetic algorithm (GA), where the solver keeps the multiple search points (a population) and evolves the population so as to cover the promised search area in the search space. The other is the evolutionary strategy (ES), where a single search point is considered and the promised mutation shape is evolved so that it can efficiently generate improved candidates. The GA is generally thought to work better than the ES in the problems that have many local optima because GA can provide global optimization. The ES with covariance matrix adaptation (CMA-ES) [4], which adapts the covariance of mutation shape, has recently been shown to perform well on some benchmark optimization problems which have single peak landscapes.

In this paper we will first apply the CMA-ES to the lens system design problem formalized as real-valued function optimization problems [1] and show that it works better than the real coded-GA. And then we will apply it to the benchmark problems for lens design problem.

Section 2 describes the lens system design problems, explaining both the basis of lens system design and the formalization of the design problems. Section 3 outlines the CMA-ES algorithm. Section 4 gives experimental results and discusses them. We apply the CMA-ES to the benchmark in section 5. Section 6 concludes this paper.

2 The Lens Design Problem

2.1 The Basis of Lens System Design

Given an object of a certain size in the object space, the main task of a lens system is to produce an accurate images of the object. An ideal lens system is defined as follows:

(i) The divergent ray bundles emerging from any single object point on the object plane should be concentrated, after passing through the lens system, to a corresponding image point on the image plane.
(ii) The image of an object on the image plane should be similar to the object on the object plane by a given lateral magnification.

A lens system satisfying these requirements is called an ideal lens system, but an ideal lens system cannot actually be constructed because there are inevitably the differences between a real image and a corresponding ideal image. These differences are aberrations, and the purpose of lens design is to determine the lens arrangement – its surface curvatures, its thickness and its materials – so as to minimize the aberrations.

To calculate the aberrations we need to know real image and corresponding ideal image. The real image is calculated by the procedure called ray trace. Starting at a given point in the object space and proceeding in a given initial direction, a ray trace is the computation of the trajectory of a light ray through the lens system until it reaches the image point on the image plane. The exact

ray trace is obtained from the first law of refraction, which governs the behavior of light rays passing through the interface between two media having different refractive indexes. The path of a ray passing from medium 1 to medium 2 obey the following equation:

$$n_1 \sin\theta_1 = n_2 \sin\theta_2 \tag{1}$$

where n_1 and n_2 are the refractive indexes of media 1 and 2, and θ_1 and θ_2 are incident and refracted angles. By using the first-order approximation of the expansion of a sine function, we can write Eq. (1) as

$$n_1\theta_1 \approx n_2\theta_2 \tag{2}$$

This is the case when rays are close to the optical axis, and this approximation is called the paraxial approximation. When Eq. (2) holds, a lens system having spherical surfaces becomes an ideal lens system. Therefore an ideal image point of an object point is defined as the point at which rays diverging from the object point converge according the paraxial approximation. The calculation of the basic specifications of the lens system – such as a focal length, f-number, and back focal length – can be based on the paraxial approximation.

The cross-section of a lens system having three lens elements is shown in Figure 1. An N-surfaces lens system (N = 6 in Figure 1) can be characterized by the parameters, $c_1, \ldots, c_N, d_1, \ldots, d_N, h_1, \ldots h_N, n_1, \ldots, n_{N+1}$ and s, where c_i is the curvature of the i-th surface and d_i is the distance from the i-th surface to the succeeding one (the N+1-th surface is defined as the image surface). h_i is the aperture of the i-th surface. n_i is the refractive index of the medium occupied between i-th surface and preceding one, and s is the position of the stop. The c_i, s and h_i can take any real value as long as the lens system is physically feasible. Because the refractive indexes are characteristic of materials, however, we can select for the refractive indexes only the values of available materials. Refractive indexes are also wavelength dependent, so three or four refractive indexes corresponding to representative wavelengths in the visible range need to be known.

The f-number, the focal length, and the field size are fundamental specifications that can be explained with reference to Figure 1, which illustrates parallel ray bundles that enter the lens system at field angles of 0 and θ. The line through the centers of curvature of the surfaces is called the optical axis, and the rays passing through the center of the stop are defined as principal rays. Considering the ray bundle parallel to the optical axis, we can see that the focal length is the distance between the focal point C and the vertical position H at which the ray refract. The focal length f is also related to the field angle θ and the image height y. When an object is located infinitely far from the lens system, this relation becomes $y = f\tan\theta$. The f-number, which is related to the brightness of the image, is defined by $F = f/D_{pupil}$, where D_{pupil} is the diameter of the effective ray bundle parallel to the optical axis (i.e. the parallel ray bundle that can reach the image surface). The field size $2w$ is the maximum size of the field that the lens systems should cover. Therefore, we must consider any ray bundle having a field angle θ less than w.

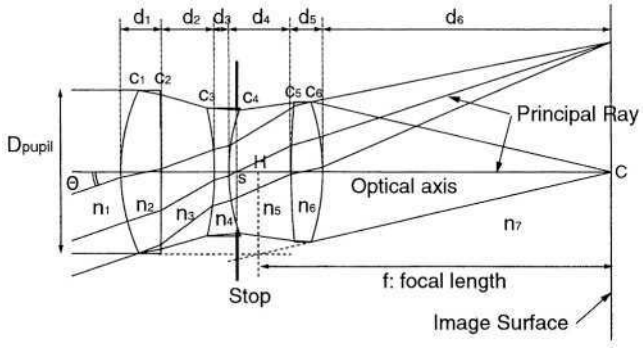


Fig. 1. An example of the lens system.

2.2 Formalization of Lens System Problems

Ono et al.[1] selected the $c_1, \ldots, c_{N-1}, d_1, \ldots, d_{N-1}$ as the control parameters, and characterized the lens system by $2N-2$ parameters having real values. These parameters were scaled so that the search ranges of curvatures and the distance given in advance were linearly transformed into [0,1]. It should be noted that c_N and d_N were not control variables because they are determined for a lens system to meet a specified focal length. The position of the stop was determined heuristically. Beaulieu et al. [3] further selected the stop position as a control variable. Because their experiments treated monochromatic problems, the refractive indexes were fixed at the helium d wavelength of frequently used glass.

They evaluated the lens systems by using spot diagrams, which are the sets of the points at which the ray bundle diverging from an object point intersect on the image plane. Although as many as possible rays included in the ray bundle should be traced when evaluating the lens system accurately, computational cost are usually reduced by tracing only representative rays including the principal ray and several other several types. Moreover, only three or four representative object points on the object plane are usually selected. When the object is located at infinity, three field angles are selected: 0, $0.65w$, w. Figure 2 illustrates an example of the spot diagram and the intersection points of the traced rays having the field angle of 0 at the first surface.

To evaluate lens systems in view of the requirements (i) and (ii) for the ideal lens system, Ono et al. defined the resolution R and the distortion D as follows:

$$R = \sum_{\theta \in \{0, 0.65w, w\}} \sqrt{\sum_{k=1}^{M} \{(x_{\theta k} - x_{\theta 0})^2 + (y_{\theta k} - y_{\theta 0})^2\}/M} \tag{3}$$

$$D = \sum_{\theta \in \{0, 0.65w, w\}} \sqrt{(x_{\theta 0} - x_{\theta I})^2 + (y_{\theta 0} - y_{\theta I})^2} \tag{4}$$

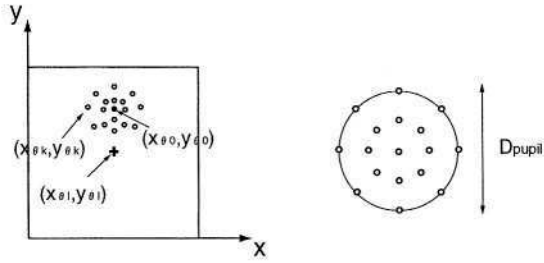


Fig. 2. Spot diagrams and the intersection points of the traced rays at the first surface (the field angle is 0)

where $(x_{\theta I}, y_{\theta I})$ is the ideal image points, $(x_{\theta 0}, y_{\theta 0})$ is the image point of principal ray and $(x_{\theta k}, y_{\theta k})$ are the image points of the other surrounding rays on the image plane, each inclined to the optical axis by an angle θ. In this case $(x_{\theta I}, y_{\theta I})$is defined as $(0, f\tan\theta)$. Given the number of surfaces *N,* a focal length f, a f-number *F,* and a field size $2w$, as a specifications, Ono et al. used real-coded genetic algorithms to optimize the lens system under the evaluation function $F = D + R$. They also applied the multi-objective GA using the Pareto optimal selection strategy. Beaulieu et al. [3] applied the bit-string GA and the genetic programming under the similar conditions, where the distortion D was considered as the constraint. In their experiments the distortion had to be less than 1% and the definition of D was different from that specified by Eq. (4). It was

$$D = \max_{\theta \in \{0, 0.65w, w\}} \{ \frac{\sqrt{(x_{\theta 0} - x_{\theta I})^2 + (y_{\theta 0} - y_{\theta I})^2}}{\sqrt{(x_{\theta I})^2 + (y_{\theta I})^2}} \} \tag{5}$$

3 The CMA-ES Algorithm

The algorithm for the covariance matrix adaptation evolution strategy (CMA-ES) [4], one of the most powerful self adaptation mechanisms available, is the following.

(1) Initialize $g \in R$, $C \in R^{n \times n}$, $p_c \in R^n$, $p_\sigma \in R^n$, $\sigma \in R^+$ and $< x >_\mu \in R^n$ as follows:
$g = 0$, $p_c^{(0)} = p_\sigma^{(0)} = 0$, $C^{(0)} = I$ (unity matrix). $\sigma^{(0)}$ and $< x >_\mu^{(0)}$ are given as inputs.

(2) At each generation $g + 1$, compute the λ offspring by using Eq. (6), where z_k are $\mathcal{N}(0, I)$ distributed (elements of z_k are independently $\mathcal{N}(0, 1)$ distributed).

$$x_k^{(g+1)} = < x >_\mu^{(g)} + \sigma^{(g)} B^{(g)} D^{(g)} z_k^{(g+1)}, \qquad k = 1, \ldots, \lambda \tag{6}$$

(3) Select the μ best offspring among them (the set of indices of the selected offspring is denoted as $I_{sel}^{(g)}$ and compute $< x >_{\mu}^{(g+1)}$ and $< z >_{\mu}^{(g+1)}$.

$$< x >_{\mu}^{(g+1)} = 1/\mu \sum_{I_{sel}^{(g+1)}} x_i^{(g+1)} \tag{7}$$

$$< z >_{\mu}^{(g+1)} = 1/\mu \sum_{I_{sel}^{(g+1)}} z_i^{(g+1)} \tag{8}$$

(4) Update the global step size σ and matrices B and D as follows:

$$p_c^{(g+1)} = (1 - c_c)p_c^{(g)} + \sqrt{c_c(2 - c_c)}\sqrt{\mu}/\sigma^{(g)}(< x >_{\mu}^{(g+1)} - < x >_{\mu}^{(g)}) \tag{9}$$

$$C^{(g+1)} = (1 - c_{cov})C^{(g)} + c_{cov}p_c^{(g+1)}(p_c^{(g+1)})^T \tag{10}$$

$$p_{\sigma}^{(g+1)} = (1 - c_{\sigma})p_{\sigma}^{(g)} + \sqrt{c_{\sigma}(2 - c_{\sigma})}\sqrt{\mu}B^{(g)} < z >_{\mu}^{(g+1)} \tag{11}$$

$$\sigma^{(g+1)} = \sigma^{(g)} \exp(\frac{1}{d_{\sigma}} \frac{\|P_{\sigma}^{(g+1)}\| - \hat{\chi_n}}{\hat{\chi_n}}) \tag{12}$$

The matrices B and D are determined so as to satisfy

$$C^{(g)} = B^{(g)}D^{(g)}(B^{(g)}D^{(g)})^T \text{ and } C^{(g)}b_i^{(g)} = (d_{ii}^{(g)})^2 b_i^{(g)} \tag{13}$$

(5) Increment g and go to step (1)

The parameter setting is discussed in [4] and the default setting is as follows:

$$c_c = \frac{4}{n+4}, \quad c_{cov} = \frac{2}{(n+\sqrt{2})^2}, \quad c_{\sigma} = \frac{4}{n+4}, \quad d_{\sigma} = c_{\sigma}^{-1} + 1 \tag{14}$$

The term $< x >_{\mu}^{(g)}$ in Eq. (7) represents the mean vector of the selected individuals of generation g that becomes the center of search point of next generations. The offspring is created by adding a distributed vector $\mathcal{N}(0, \sigma C)$ to $< x >_{\mu}^{(g)}$, where σ is the global step size and C is the covariance matrix. The term σ and C respectively represent the overall variance and distribution shape of the mutation. These two factors are adapted separately because they should change on different time scales. The covariance matrix C is adapted by Eq. (9) and Eq. (10), and the global step size σ is adapted by Eq. (11) and Eq. (12) respectively. The covariance matrix C is updated with $p_c^{(g+1)}(p_c^{(g+1)})^T$, which is a symmetric matrix of rank one. p_c is the evolution path that is the weighted accumulation of the previously successful mutation. The evolution path often speeded-up the adaptation of the covariance matrix C. The global step size σ is updated with p_r in Eq. (12) where $\hat{\chi_n}$ is expectation of the length of a (0, I)-normally distributed random vector which is approximated by $\hat{\chi_n} \approx \sqrt{n}(1 - \frac{1}{4n} + \frac{1}{21n^2})$. The p_{σ} is

also the evolution path calculated by Eq. (11) that is similar to Eq. (9) except that the scaling matrix D is omitted.

The distributed vector $\mathcal{N}(0, C)$ can be created by BDz, where z is the $\mathcal{N}(0, I)$ distributed vector and BD satisfies Eq. (13). The matrix B is the orthogonal $n \times n$ matrix whose i-th columns are normalized eigenvectors of the covariance matrix C, and the matrix D is the $n \times n$-diagonal matrix whose elements d_{ii} are square roots of eigenvalues of C. The main drawback of the CMA algorithm is that the computation time for a single cycle of the algorithm is $O(N^3)$, where N is the dimension of the search space. Several modified algorithm CMA-ES algorithms have therefore been proposed with the aim of reducing the computational time [7,8].

4 Experiments and Results

To evaluate the capability of the CMA-ES for the lens design problems, we apply it to the lens design problems and compare the results with those of representative real-coded GAs.

4.1 Problems

In the experiments we choose two specifications of lens systems used in Ono's experiments [1]. For each specifications we design the lens systems having 4 or 6 lens elements. The specifications are given as follows.

- Specification 1: The focal length, $f = 100mm$, the F number, $F/2.0$, and the field angle, $2w = 45.0\,\mathrm{deg}$. The refractive indexes of all lens are fixed at 1.613.
- Specification 2: The focal length, $f = 100mm$, the F number, $F/3.0$, and the field angle, $2w = 38.0\,\mathrm{deg}$. The refractive indexes of all lens are fixed at 1.613.

The evaluation function is the resolution defined by Eq. (3) and the distortion defined by Eq. (5) is used as a constraint that it must be less than 1%.

4.2 Parameter Setting

N-surfaces lens system are represented by vector $x = (c_1, \ldots, c_{N-1}, d_1, \ldots, d_{N-1}) \in [0, 1]^{2N-2}$. The search range of each parameter is given in advance and linearly transformed into [1,0]. The search ranges of the curvatures c_i and the distances d_i are set to [–0.66, 0.66] (the absolute values of radius are larger than 15.0) and [0.0, 20.0] respectively. The position of stop is determined heuristically in the same way as the Ono's experiments [1].

The CMA-ES uses the parameter setting $\lambda = 20$, $\mu = 4$, and $\sigma^{(0)} = 10.0$. 100 trials are performed for each problem, where each run is stopped if converged.

In Ono's experiments, GA consisted of UNDX and MGG. UNDX-m crossover has been proposed as an generalization of the UNDX and shown a better performance on some benchmark [6]. So we apply UNDX-m, where m is set to 1,2, and 4 (UNDX(m=1) is equal to UNDX). Moreover we try two kind of alternation-generation model, one is the original MGG where the best and the roulette-selection individuals survive to the next generation. and the other is modified MGG where the best two individuals survive. The later has strong selection pressure. Though we applied the 6 type of GAs, only the results of two GAs are shown. One is the GA with UNDX and the original MGG model, that is used in Ono's experiments. The other is the GAs with UNDX-m(m = 4) and the modified MGG, that showed the best performance among them. 30 trials are performed for each problem, where each run is stopped if converged.

4.3 Results

Figure 3 shows the convergence curve on the 6-elements lens design of the specification 1. The horizontal axis and vertical axis respectively represent the number of evaluations and the resolution of the best individual obtained so far. The CMA-ES can converge about 40 times faster than the GAs. Moreover, the qualities of the lens systems obtained by CMA-ES are not inferior to those of the GAs as shown Figure 4.

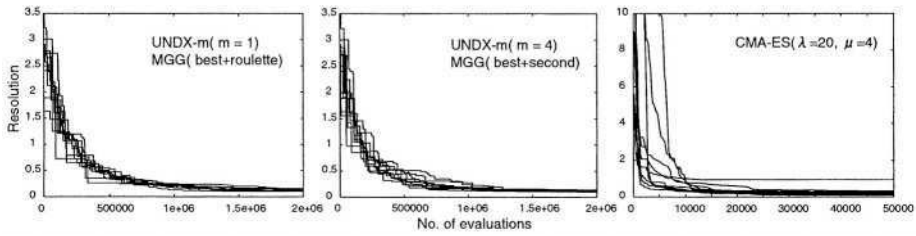


Fig. 3. The converge curve of each optimization method. The 6-elements lens design problems on the specification, $f = 100mm, F/2.0, 2w = 45.0\,\mathrm{deg}$

Fig. 4 shows the qualities of the lens systems obtained by each method. The horizontal axis and vertical axis respectively represent the resolution and the frequency, where the number of trials of the CMA-ES and the GAs are 100 and 30 respectively. Fig. 5 shows the best lens systems obtained by CMA-ES.

The computation time for a single run of the CMA-ES is about 300 sec when applied to the 6-elements lens design of the specification 1. The GAs converge about 40 times slower than the CMS-ES. The algorithms are implemented by C++ language on Xeon processors at 1.7 GHz.

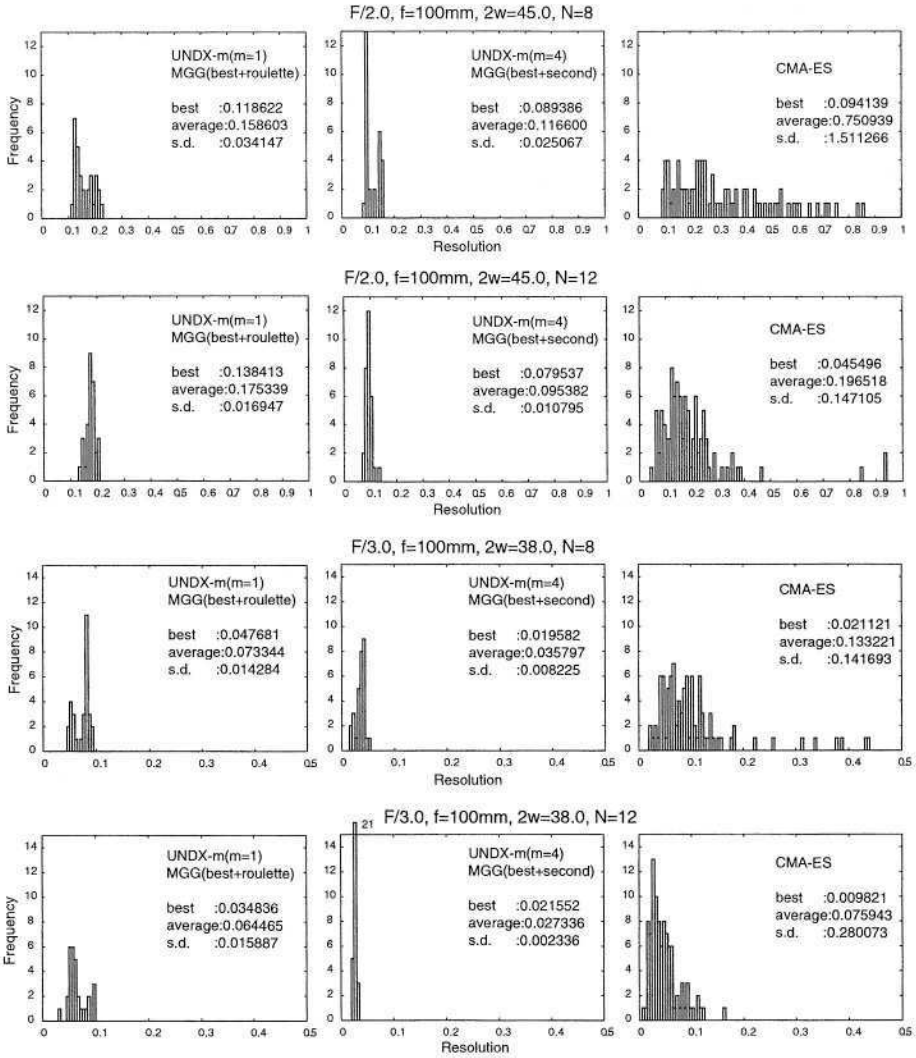


Fig. 4. The frequency of the best solutions. GAs perform 30 trials and CMA-ES performs 100 trials.

4.4 Discussion

The figure 4 shows that the GAs always produce the high quality solutions. Though the averages of resolution obtained by CMA-ES are significantly worse than those of the GAs, the CMA-ES can produce high quality lens systems more efficiently than the GAs because the number of evaluations of CMA-ES requited for the convergence is much smaller than those of the GAs (See Figure 3). The GA with UNDX-m(m = 4) and MGG(best two) work better than GA

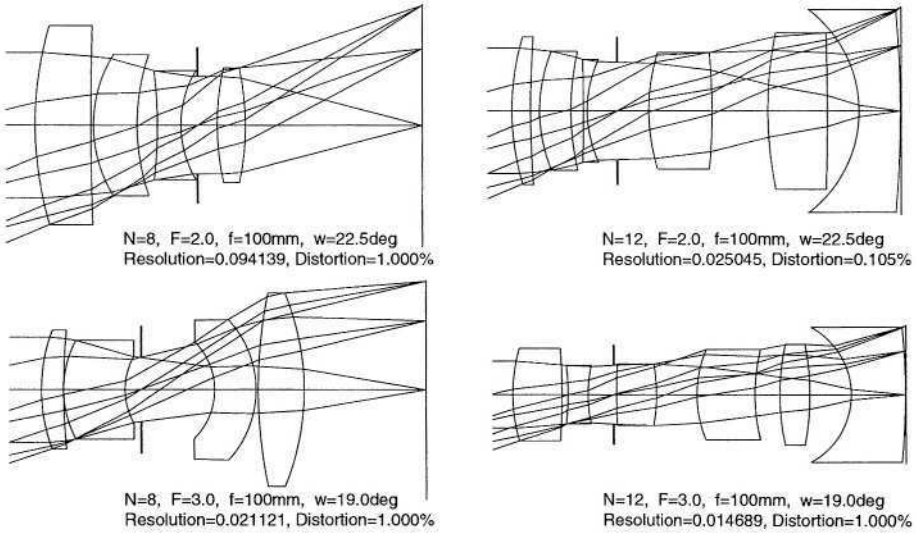


Fig. 5. The best lens design obtained by GAs and CMA-ES

with UNDX and MGG(best+roulette). The results indicate that the GA that converges rapidly tend to work well on the lens system design problems. The landscapes of the lens system are very complicated, therefore crossovers have a few chance to generate improved offspring from parents located remotely in the search space because there is no correlation between such parents. Therefore the population need to converge to a certain degree. On the other hand CMS-ES can adjust the global step size rapidly and locally estimates the landscape. Moreover the results shows that the CMA-ES doesn't easily fall into local optimum

5 Application to the Benchmark

Beaulieu et al.[3] applied the bit-string GA to the benchmark for the lens design problems. The problem is stated in the 1990 International Lens Design Conference (ILDC) [9] and human experts made several solutions, one of these seem to be global optimum. The specification of the lens system is as follows:

- Specification 3: The focal length, $f = 100mm$, the F number, $F/3.0$, and the field angle, $2w = 30.0$ deg. The refractive indexes of all lens are fixed at 1.51680. The number of lens elements is 4.

We apply the CMA-ES with the same parameter setting and the evaluation function described in the previous section. The search ranges of the curvatures c_i and the distances d_i are set to [–0.66,0.66] and [0.0, 75.0] respectively. We

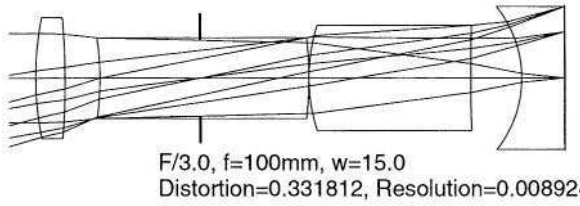


Fig. 6. The best lens design obtained by the CMA-ES under the specification $f = 100mm, F/3.0, 2w = 30.0\,\mathrm{deg}, N = 8$

Table 1. Parameters of the best lens design obtained by the CMA-ES. The stop position is given in the column of thickness.

surface	radius	thickness	aperture	fractive index
1	112.865308	10.887726	22.383670	1.000000
2	-276.900906	12.930491	22.383670	1.516800
3	-103.878518	77.560903	15.018847	1.000000
4	-127.478825	0.005718	15.018847	1.516800
5	63.343159	60.486656	19.410025	1.000000
6	671.192334	19.317317	19.410025	1.516800
7	-41.532186	15.531193	26.488189	1.000000
8	3805.818724	0.198941	26.488189	1.516800
stop		61.012946	14.064979	

performed 300 trials for this problem. The averaged computation time for a single run is 113 seconds on the same processors.

Fig. 6 and Table 1 show the best lens system obtained by 300 trials. The resolution of the best solution is 0.008942. The resolution of the best 1990 ILDC lens system is 0.010434 in our used evaluation function. The best lens system obtained in this experiment is better than the best presented at the 1990 ILDC. Though the evaluation function may be differ a bit from the one used in 1990 ILDC because of the flexibility for selecting tracing rays in the process of the ray trace, we can conclude that the CMA-ES can find the human-competitive solutions efficiently.

6 Conclusion

In this paper we applied the CMA-ES to the lens system design problems and demonstrated that this method can provide high quality solution efficiently better than the representative real-coded GAs. Moreover the human-competitive solution was found on the benchmark of the 4-element lens design problem. GAs are generally expected to be suitable for the problems whose landscape has a lot of local optima And the ES is expected to converge rapidly to the local optimum. Many papers have reported these features on the benchmarks. In the

lens system design problem, the CMA-ES and the GAs have almost same capabilities of reaching at good solution even if the landscapes are very complex. More over, the CMA-ES converges much faster than the GAs and doesn't easily fall into local optima. We conclude that the landscape of the lens system design problems are very complicated and then crossovers cannot generate improved offspring from parents located remotely in the search space because there is no correlation between such parents. In this case the ES can perform better than GAs.

References

1. I. Ono, S. Kobayashi, and K. Yoshida: Optimal lens design by real-coded genetic algorithms using UNDX , Computer methods in applied mechanics and engineering, 2000, pp. 438-497.
2. D. Vasiljevic: Classical and Evolutionary Algorithms in the Optimization of Optical Systems, Kluwer Academic Publishers, 2002.
3. J. Beaulieu, C Gagne, and M. Parizeau: Lens System Design and Re-Engineering with Evolutionary Algorithms, Proc. of the Eleventh International Conference on Genetic Algorithms(ICGA-2002), 2002, pp.155-162.
4. N. Hansen, and A. Ostermeier: Completely Derandomized Self-Adaptation in Evolution Strategies, Evolutionary Computation 9(2), 2002, pp. 159-195.
5. I. Ono and S. Kobayashi; A real-coded genetic algorithm for function optimization using unimodal distribution crossover, Proc. of the Seventh International Conference on Genetic Algorithms(ICGA-97), 1997, pp.246-253.
6. H. Kita: A comparison study of self-adaptation in evolution strategies and real-coded genetic algorithms, Evolutionary Computation 9(2), 2002, pp. 223-241.
7. J. Poland and A. Zell: Main Vector Adaptation: A CMV Variant with Linear Time and Space Complexity, Proc. of the Tenth International Conference on Genetic Algorithms(ICGA-2001), 2002, pp.1050-1055.
8. N. Hansen, S. Muller, and P. Koumoutsakos: Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation(CMA-ES), Evolutionary Computation 11(1), 2003, pp. 1-18.
9. O'Shea, D.C.:The monochromatic quartet: A search for the global optimum, Proc. of International Lens Design Conference 1990, Vol.1345 of SPIE.

Automatic Synthesis of an 802.11a Wireless LAN Antenna Using Genetic Programming

A Real World Application

Rian Sanderson

SiRiFIC Wireless
Santa Rosa, CA 95409
rsanderson@sirific.com

Abstract. This paper describes the application of genetic programming to synthesize a small form factor, 2 dimensional wire antenna for a 5.2 GHz 802.11a wireless LAN application. Utilizing basic genetic programming techniques and using a novel approach to discretizing the search space leads to a more simple problem representation and implementation. The paper presents simulation results exceeding expectations and that are competitive to commercial products.

1 Introduction

The enigmatic nature of radio frequency (RF) engineering, the problem domain's large search spaces, and the availability of simulators are all factors in the frequent choice of evolutionary algorithms [1,2,3] for synthesizing and optimizing antennas. Specifically, genetic programming (GP) is well suited to this domain when antenna geometries are represented with an implementation of the LOGO drawing language [2, 3].

This paper uses GP to create a 5.22 GHz antenna for an 802.11a wireless LAN application. An efficient antenna, one with good gain and low voltage standing wave ratio (VSWR), is an important part of an 802.11 system; it can extend the reach of signals, or prolong battery life by requiring less transmission power.

1.1 What Is Genetic Programming?

To paraphrase, [2] genetic programming is a technique for automatically creating computer programs which solve, or approximately solve, problems by utilizing a high level statement of the structure's desired characteristics. Genetic programming has been successfully used for sequencing proteins, synthesizing electronic circuits, creating robot control algorithms, and other difficult tasks. Genetic programming is further explained by Koza [5].

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1201–1213, 2004.

1.2 Why an 802.11a Antenna?

The IEEE 802.11 standards are the most popular wireless networking standards for home and enterprise. IEEE 802.11b/g [6], in the 2.4 GHz band, has a low cost and is prevalent in homes, school campuses, airports and coffee shops. IEEE 802.11a [7] with its higher data rates, reduced range, and higher cost, is geared more for the enterprise and is not yet as prevalent.

Antennas for 802.11a are difficult to find, there are far more antennas for 802.11b/g. Standard, low gain, 5/8 wavelength, omni directional whip antennas are a feature on most access points. Yaego, a manufacturer of RF components, has a discrete ceramic 802.11a antenna [11] available from major parts suppliers. Murata also has a discrete ceramic antenna, the LDA131 [12], but only releases it and its data under non disclosure agreement. Due to their size and fixed position application, Centurion Technologies Whisper 5 GHz [13], at 6.4 x 6.4 x 1.7 cm, and the Hyperlink Technologies HG5308P and HG5311P [14], at 11.4 x 11.4 x 2.5 cm, are the most comparable antennas to this research. All these antennas work in the range from 5.18 GHz–5.35 GHz.

1.3 Other Research

As its name implies, Electromagnetic Optimization by Genetic Algorithms, is a whole text describing genetic algorithm (GA) optimization of various RF problems, but the chapter by Altshuler and Linden, [1] describes automatic synthesis of "crooked wire antennas" using a GA. This successful antenna work focuses in the 1600 MHz range and uses the NEC2 simulator.

Lohn et al [3] use both a conventional GA encoded in real numbers, and a branching GA with elements of the LOGO drawing language. This branching GA is close to a GP representation with two functions, forward and rotate, each taking a real valued argument. Their successful 8470 MHz antenna, with a footprint about the size of a quarter, is scheduled for launch with NASA's Space Technology 5 space craft.

Comisky et al [2] describe the rediscovery, by GP, of a 424 MHz Yagi Uda type antenna which fits in a bounding rectangle .4 meters by 2.65 meters. They utilize the LOGO drawing language to represent antenna geometries and directly compile in the source for a later version of the NEC2 simulator, NEC4. They use real valued quantities for wire lengths and angles, and have a more broad function set utilizing function argument constraints. To deal with simulator geometry restrictions they employ a complex wrapper which applies a lethal fitness to individuals violating geometry constraints, and which replaces wire intersections with 4 new wires.

To varying degrees these works preconceive what the final antenna will look like. While Altshuler and Linden [1] do not specify the configuration of wires, their GA implementation fixes the number of them at 7. Lohn et al [3] use the GA to design one arm, which will is symmetrically repeated, in a wire antenna with four arms separated at 90 degrees. Comisky et al [2] had a goal to not preconceive the solution as a Yagi Uda antenna, but the extra weight on elements to create separate wires, the choice of landmark functions and "no wire" terminals, and the symmetrical reflection of the generated geometry both lead toward a Yagi Uda type solution.

Based on this research we were confident we could make an antenna, but questioned: the quality of VSWR and gain we could find, if the targeted physical size was too small, and if our lack of predefined antenna structure or symmetry would hamper results.

2 The Problem

The problem is to synthesize a high gain, low VSWR, two dimensional, continuous wire antenna for the lower sub band (frequency range 5.18–5.24 GHz) of the 5 GHz FCC Unlicensed National Information Infrastructure (U-NII) band which is no larger than 6 cm square, and made of American wire gauge (AWG) #19 wire. The antenna is simulated using the freely available NEC2 simulator at a single representative frequency of 5.22 GHz, in free space, assuming a lossless conductor.

Rather than using real valued quantities for wire sections and angles, the 6 cm square problem space is discretized into a matrix of 15 by 15 points. As Figure 1 shows, a continuous wire antenna can be viewed as a collection of small wires each joined at a point in the matrix.

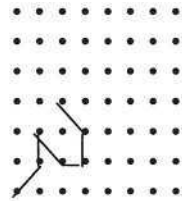


Fig. 1. An antenna viewed as the connection of six individual segments

NEC2 has many conditions regarding allowable geometries [8] with respect to wavelength, wire radii, and segment length. By spacing the matrix points appropriately the number of unsimulatable individuals can be minimized. For this problem, with its 5.7 cm wavelength signal on AWG #19 wire, 4 mm is the minimum spacing.

We discretized the search space in an intent to obviate the need for complicated geometry verification, as in [2]. However we ended up having to employ a simple geometry check to look for improper segment crossing, duplicate segments, and segments of zero length.

Segment crossing is the major simulator condition that the quantization does not protect against. Figure 2 demonstrates valid and invalid junctions.

NEC2 is based on the method of moments [8], and the simulator cannot compute the current on overlapping segments due to an indeterminate matrix calculation.

Discretizing the problem space puts few limits on what the end antennas will look like. While in this implementation all antennas are continuous and non branching, it is the choice of terminals which does not allow discontinuity, and the handling of duplicate segments which prevents branching. Furthermore, the problem space could

readily be extended to 3 dimensions, where discretizing would help prevent even more geometry conflicts.

Fig. 2. Wires may only cross at end points, not in the middle of segments

Though bounded, the total size of the search space is difficult to calculate. Calculating the permutations of all possible wire segments is not correct; each segment must share at least one end point with another segment, leaving no stray unattached segments. Also, crossed segments, as discussed previously, must not be counted, nor can antennas with branches.

3 Methods

The concept and fitness measure are closely modeled after Comisky et al [2], though our simplifications render the architecture similar to Koza's artificial ant problem [5]. The methods are summarized in Table 1 and discussed in detail in the following sections.

3.1 Architecture

The simple architecture of one result producing branch, the connective functions progn2 and progn3, and terminals which actually do the work, bears similarity to Koza's implementation [5] of the Artificial Ant problem. The connective functions, progn2() and progn3(), evaluate each of their arguments in order, and return no value. The real work, output of LOGO instructions, happens as a side effect of evaluating the terminals at the leaves of progn trees.

The rotate terminals, rotxy 45, rotxy 90, rotxy 315, rotxy 270 change the direction of the turtle in the X-Y plane, and were chosen with the thought of expanding the experiment to 3 dimensions. While not strictly necessary, all four rotate terminals seem appropriate when looking at the problem from a human's point of view. In the actual results, rotate operators are often stacked five or more deep, causing the turtle to spin several revolutions to change perhaps 45 degrees.

The move terminal is the only operation that creates wires. It advances the turtle to the next matrix point in the current direction, creating a wire segment from the point at the start of the operation to the point at the end of the operation. Moves that will take the turtle outside of the matrix are ignored and become no-ops.

Table 1. Tableau sumarizing methods

Objective:	synthesize a two dimensional continuous wire antenna for the Lower UNII band (frequency range 5.18– 5.24 GHz) which is no larger than 6 cm square, and made of American wire gauge (AWG) #19 wire.
Architecture:	One result producing branch
Functions:	progn2(x,y), progn3(x,y,z)
Terminals:	move, rotxy 45, rotxy 90, rotxy 315, rotxy 270
Fitness Case:	simulation with NEC2 at a single frequency of 5.22 GHz, generating radiation data for a full 3 dimensional plot with 5 degree increments
Raw Fitness:	-maxgain + V(vswr) ; smaller fitness is better
Standardized Fitness:	-success predicate + Raw Fitness
Hits:	not used
Wrapper:	ignores out of bounds move instructions. Geometry checker: crossed segments, zero length segments, duplicate segments
Selection:	tournament with size of 7, .9 probability of cross over, .1 probability of mutation
Parameters:	M=1000,G=100; M=5000,G=40; M=6500,G=35; M=1000, G=30
Success Predicate	exit at fitness of -14

3.2 Fitness and Selection

Simulating at only one frequency cannot be called rigorous, but it speeds computation. A more robust fitness case, such as [2], would run the simulation over a frequency range and use the worst case.

The raw fitness measure (1) combines the two most important antenna criteria, the maximum gain and the voltage standing wave ratio (VSWR), to try and get the most negative number possible

$$\text{fitness} = -\text{maxgain} + V(\text{vswr}) \tag{1}$$

$$V(\text{vswr}) = \text{vswr} * C, \tag{2}$$

where

$C = .1$ for $\text{vswr} <= 2$,
$C = 1$ for $\text{vswr} <= 3$,
$C = 10$ for $\text{vswr} > 3$.

This fitness measure is based on work by Altshuler and [1] as well as Comisky et al [2], though an even higher penalty for poor VSWR was assessed because VSWR is critical in 802.11 systems. Early on in the process the VSWR term dominates, but to get a really negative (better) fitness the antenna must have a high gain.

The voltage standing wave ratio (VSWR) [9] is a measure of reflective wave interference, and can be thought of as how well matched an antenna input is to the transmission line feeding it. As its name suggests, VSWR is a ratio; a value of 1.0 means all energy is radiated from the antenna and it reflects nothing back to the feeding source. A VSWR of infinity radiates all energy back to the line feeding the signal and radiates no energy in its intended direction. The VSWR for a particular antenna is not a single quantity; it is a function which varies across the excitation frequency. When VSWR is mentioned as a single quantity in this paper it is at the simulated frequency of 5.22 GHz with a 50 ohm input impedance. A VSWR greater than 3 is unacceptable in a WLAN system.

Gain describes an antenna's ability to apparently amplify a signal [9]. The higher the gain the more distant a signal of a given power may be transmitted or received. Gain is measured in decibels relative to isotropic (dBi); where an isotropic antenna is an ideal antenna which radiates equally in all directions. Standard whip antennas have gains around 2 dBi, directional antennas such as the biquad have gains near 10 dBi.

Though not explicitly included in the measure of fitness, beam width is also an important characteristic. Beam width, measured in degrees, describes how wide the gain pattern is dispersed. Outside the arc of beam width, signals are transmitted at less than half the power as inside the beam width. Antennas with higher gain tend to have a more narrow beam width. The success predicate indirectly affects beam width, because it terminates the run before the antenna overspecializes with a super high gain and narrow beam width.

We employ a geometry checker as a wrapper to screen individuals to ensure they do not violate simulator constraints. The geometry checker looks for crossed segments, segments of zero length, and duplicate segments. Rather than try and fix each of these problems, as in the simple case of ignoring move instructions when they happen, the geometry checker flags the individual as a geometry error, giving it an enormously large, and lethal, fitness value.

Standardized fitness, a transformation of raw fitness so that 0 is the most fit, is necessary for the breeding process, and is achieved using the success predicate term.

3.3 Choice of Parameters

Based on results from Koza [10] tournament selection is used with .9 probability of crossover and . 1 probability of mutation.

Initially we chose values for M and G, population and number of generations, with a 12 hour run as the target. After hypothesizing that genetic diversity was being lost due to excessive culling by the geometry checker we increased the population size and found faster servers to run on. Initial results ran in about .4 seconds per individual (M1000, G100 in approximately 11 hours), faster hardware yields evaluation times of .16 seconds per individual (M6500, G35 in approximately 10 hours). Improvement in large M runs started to level off around generation 30.

4 Structures Undergoing Adaptation

The trees of functions and terminals that actually undergo adaptation bear no resemblance to the antennas they generate. Figure 3 shows a portion of the program tree for a random individual of generation 0.

```
(progn2(progn2 (progn3 (progn2 (progn2 rotxy 90 rotxy 90)
(progn3 rotxy 90 rotxy 45 move))
(progn3 (progn2 (progn3 rotxy 315 rotxy 270 move)
(progn2 rotxy 90 move))
(progn3 rotxy 90 move rotxy 270 )
(progn3 rotxy 90 rotxy 270 rotxy 45))
```

Fig. 3. A portion of a program tree from an individual in generation 0

The individual in Figure 3, like most others, wastes effort spinning around in circles.

The antennas are much easier to view in their evaluated structure, as Figure 4 demonstrates. This average individual has a reasonable gain of 4.79 dBi, but its dismal VSWR of 240 contributes most to its raw fitness of 2395.21.

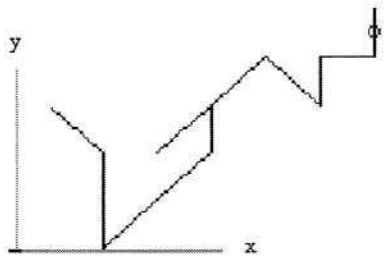


Fig. 4. Average individual from generation 0 with raw fitness of 2395.21. The branched portion is a sign of a duplicate segment, and the latest geometry checker would tag it with lethal fitness

The starting wire where the excitation signal is fed to the antenna is represented with a small circle. Note that in this run the individual started from the middle of the grid, later runs put the starting wire at the lower left. We changed the starting position for figure clarity rather than any performance difference.

As stated previously, a wire is created between two points upon the execution of a move instruction and rotate instructions don't actually create wires, they only select which point the next move instruction will connect to.

Figure 4 is generated from a tree that condenses into: rotxy 90, move, rotxy 270, move, rotxy 90, rotxy 45, move, rotxy 270, move, move, rotxy 90, rotxy 90, move, rotxy 90, rotxy 45, move, rotxy 45, move, move, rotxy 90, rotxy 45, move, move, rotxy 315, move.

The branched portion in the middle of figure 4 demonstrates an illegal duplicate segment. The turtle creates a segment by moving forward, rotating 180 degrees, then moving again. This is from an early run without the geometry check, which would tag it with lethal fitness.

5 Implementation

A foundation of this project is the quantizing of the problem space. In conjunction with the scaling feature of NEC, the quantization allows direct writing of matrix points to the NEC input file. This creating a one to one mapping from the theoretical points of the matrix and their actual mapping in the input file. In addition to bounding the search space, the discretization also fixes the two variables of wire size and segment length, further narrowing the dimensionality of the problem.

We use the lil-gp genetic programming toolkit for taking care of the grunt work involved in breeding, program tree representation, and program tree execution. We had little difficulty getting this C++ toolkit to run under Linux with gcc 2.9, WIN32 Cygwin with gcc 3.3.1, and native WIN32 with MSVC 6.0.

We ran NEC2 v2.3 for the simulator, compiling it for both Linux and WIN32 as a separate application. Under WIN32 the supporting 4nec2 program plotted antennas, drew gain fields, and gave instructional geometry warnings on models.

Our data flow is disk intensive, but writing to a file at each stage made modular development and discrete unit testing possible.

The evaluation of an individual is staged over several steps. The data flow starts with evaluating the GP tree to generate a LOGO instruction file as show in Figure 5.

```
rotxy 315
rotxy 90
rotxy 45
rotxy 45
move
rotxy 45
move
rotxy 270
```

Fig. 5. LOGO instructions generated from a GP tree

The LOGO instructions are easier for a person to understand than the original GP tree, but still difficult to visualize.

The parser converts from LOGO to NEC input, shown in Figure 6. The first wire is a fixed stub for the excitation connection and its second endpoint is the starting point for the first GP generated instruction. This fixed wire also allows for an individual which is made up of only rotate instructions.

```
CM Copyright (c) 2003
CM GP generated 802.11a antenna, 4 mm segs, #19 wire
CM TAG, #Segs, x1, y1, z1, x2, y2, z2, radius
CE
GW 1 1    0  0  0    1  1  0    .120
GW 2 1    1  1  0    2  1  0    .120
GW 3 1    2  1  0    3  2  0    .120
GW 4 1    3  2  0    2  3  0    .120
GW 5 1    2  3  0    1  4  0    .120
GS 0 0 .004
GE
EX0 1 1 0 1 0
FR0 1 0 0 5220
RP 0 73 73 1000 -180.0 0.0 5.0 5.0
EN
```

Fig. 6. GP generated NEC input file

In the lines starting with GW (generate wire card), notice the direct matrix coordinates in the NEC input file, as mentioned previously. The starting segment goes from (0, 0, 0) to (1, 1, 0). The direct coordinates are possible due to the problem space discretization and the 4 mm scaling operation of the GS card. Looking at this file a person can get a better sense of where the antenna is going: the first wire moves up and to the right, the second goes right, the third goes up and to the right again.

After translation, the geometry checker is run on the .nec input file. On a successful geometry check the input is sent to the NEC2 simulator via a system call to generate a .out file. Finally the fitness evaluator parses this .out file to compute a raw fitness.

The .out file can then be reviewed interactively outside of the GP run using 4nec2. The 4nec2 software will plot the antenna geometry for easy viewing, give a three dimensional radiation plot, and plot VSWR over a range of frequencies.

5.1 Problems Encountered

Error checking on file opening and closing, programming niceties usually shirked by most programmers, turned out to be important implementation details. Early runs failed due to a filled user disk quota and segmentation faults after opening too many file handles.

We added a duplicate segments check to the geometry screening after finding wildly successful data then discovering that duplicates were a serious error. 4nec2 does not warn about duplicate segments, and literature does not explicitly forbid them, however there is no physical way to represent them. Doubling the wire diameter for that segment violates the minimum spacing between wire segments rule, and also increases the VSWR. Removing the duplicate segments from a final model increases the VSWR above 10, invalidating results.

6 Results

All best of run individuals show convergence and improvement, with several runs producing excellent high gain, low VSWR antennas. One run exceeded our expectations with high gain and low VSWR in a frequency range doubling the problem specifications.

The best antenna synthesized, run 404803468, is pictured in figure 7 with its full radiation pattern and VSWR across the band. This antenna's performance far exceeds the Yaego chip antenna, it is within .6 dBi of the Hyperlink HG5308P, and 1.6 dBi of the Centurion Whisper 5GHz. Though a distance from the 11 dBi gain of the HG5311P, that antenna's narrow beam width of 60 degrees horizontal and 30 degrees vertical makes it more difficult to use. VSWR for antenna 404803468 creeps slightly above 2 at the bottom of the band but its average VSWR of 1.62, with a minimum of 1.27, is as good or better than the others. Its approximate size of 6 cm x 2.3 cm, one tenth the area of the HG5308P/HG5311P and one third the area of the Whisper 5GHz arguably makes it a portable or embeddable antenna.

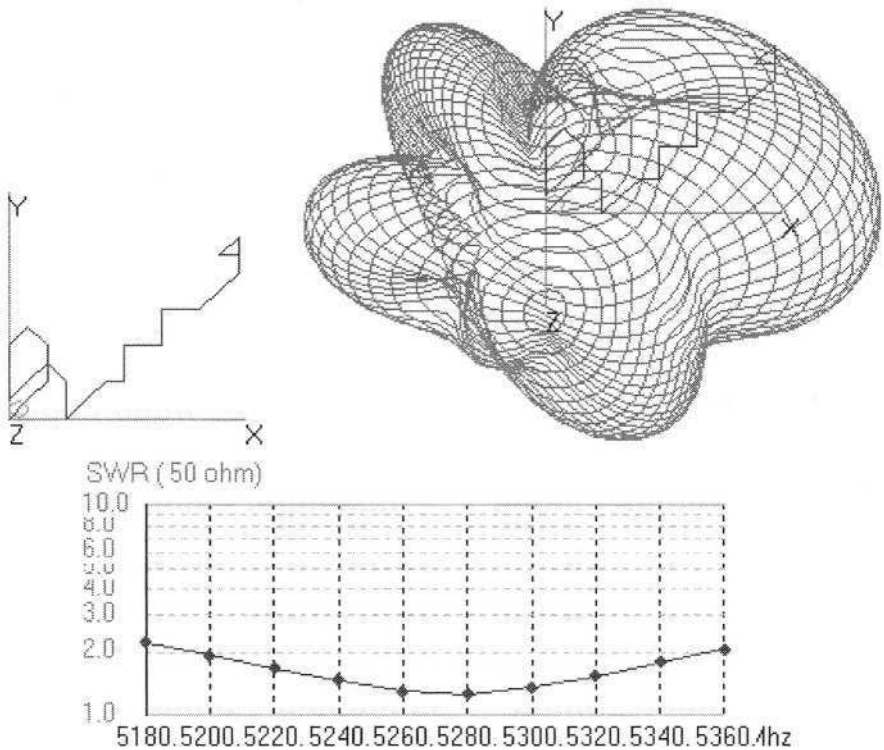


Fig. 7. The best antenna synthesized, from random seed 404803468. It has a max gain of 7.42 dBi, a VSWR of 1.68 at the fitness frequency, and a minimum VSWR of 1.27

Beyond 404803468's raw fitness, it has several desirable traits. The gain pattern is well focused, yet not overly narrow, with a beam width of 90 degrees vertical and 45

degrees horizontal. The simple geometry lends itself to more accurate construction. Most importantly, its VSWR is under 2.0 for twice the frequency range of the other GP generated antennas.

Along with antenna 404803468, Table 2 shows several other competitive and interesting antennas. Antenna 624971 (population 5000) has the second best fitness, but its complicated shape and VSWR past 5.24 GHz limit its use. Antenna 392561, though the worst in raw fitness, is actually a decent omni directional antenna with simple construction and a small 12mm by 12mm right isosceles triangle footprint. Antenna 274933 has the best minimum VSWR of 1.05 at 5.18 GHz, but this quickly rises above 3.0 after 5.24 GHz. Antenna 624971 (population 10000) looks good in the table, but its many lobed gain pattern makes it less desirable.

Table 2. Summary of results, with whip, Yaego, Centurian, and HyperLink antennas for reference

Random Seed	Population Size	Fitness	VSWR	Max Gain (dBi)	Comments
whip	for reference, not from GP		1.3	2	omni directional
Yaego	for reference, not from GP		2.5*	4	max VSWR
Centurion	for reference, not from GP		2.0*	9	max VSWR
HG5308P	for reference, not from GP		1.5**	8	avg. VSWR
HG5311P	for reference, not from GP		1.5**	11	60/30 beam width
5281999	1000	-13.86	1.46	14	invalid model, duplicate segments
8281999	1000	-3.23	2.52	5.75	
5647348	1000	-4.71	2.98	7.69	
392561	1000	-1.63	1.57	1.79	good omni pattern, trivial construction
624971	5000	-7.1	1.73	7.27	
624971	10000	-6.23	1.36	6.37	odd gain pattern
274933	6500	-6.57	1.362	6.71	narrow VSWR band, min VSWR 1.05
212519772	6500	-4.4	1.74	4.61	
404803468	10000	-7.25	1.68	7.42	best antenna

Though not an explicit goal, the fitness measure over optimizes high gain. Runs without the geometry checker, such as 5281999, produced extraordinarily high (and invalid) gain numbers, and when these runs were allowed to go for many generations the gain numbers kept getting higher while VSWR stayed just under 2.0. Increasing the weight of the VSWR term, or framing the fitness measure as a single objective problem with constraints could remedy this.

Culling out all antennas with geometry errors may be too drastic. While easy to implement, it seems to lead to a loss of too much genetic material. This hypothesis is

validated by the success of runs with a greater initial population, and suggests that a less drastic strategy for handling geometry errors should be employed.

7 Conclusion

We achieved better than expected results which validate the methodology and create the opportunity for more investigation. The genetic programming architecture is proven and simple. Discretizing the problem space reduces complexity without sacrificing the quality of results, though in the end does not obviate all geometry errors. The fitness measure is successful in selecting low VSWR, high gain individuals, but can be improved.

Though antenna 404803468 is a competitive solution and could be used, its size is still prohibitive for the average 802.11a user. This exercise really serves as an important proof of concept. We chose the problem dimensions with manufacturability in mind; AGW#19 bent at 45 degree angles with sections no smaller than 4mm can be built with a careful hand, and then tested to verify model accuracy. If after further work, simulations and empirical data correlate well we will try to shrink the wire size down so the entire grid would be 3 mm square, or apply this technique to printed circuit board antennas.

The market for external antennas is small, but a printed circuit board antenna or a wire antenna less than 3mm square (and encased in resin) could then be used in an embedded wireless device, or placed into the top of the housing of a notebook computer.

Acknowledgements. Thanks to Dr. John Koza for his interest and assistance with this endeavor, Brad Marsh from Texas Instruments for making time in a demanding schedule for the initial project, and Bryan Bell at SiRiFIC wireless for its continued support.

References

1. Altshuler; Edward E. and Linden; Derek S. 1999. Design of wire antennas using genetic algorithms. In Rahmat-Samii, Yahya and Michielssen, Eric (editors). Electromagnetic Optimization by Genetic Algorithms. New York, NY: John Wiley & Sons. Pages 211 - 248.
2. Comisky, William, Yu, Jessen, and Koza, John. 2000. Automatic synthesis of a wire antenna using genetic programming. Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference, Las Vegas, Nevada. Pages 179 - 186.
3. Lohn, Jason D; Linden, Derek S; Hornby S, Gregory; Kraus, William.F; Rodriguez-Arroyo, Adaan. Evolutionary Design of an X-Band Antenna for NASA's Space Technology 5 Mission. 2003 NASA/DoD Conference on Evolvable Hardware (EH'03) July 09-11, 2003 Chicago, Illinois. Page 155
4. Abelson,Harold and diSessa,Andrea. 1980. Turtle Geometry. Cambridge,MA: The MIT Press.

5. Koza, John R. 1992. Genetic Programming: On the Programming of Computers by Means of Natural Selection.Cambridge, MA: The MIT Press. Chapter 1, Chapter 12 Pages 327-345
6. Std. IEEE 802.11b-1999, Wireless LAN Medium Access Control and Physical Layer Specifications:Higher Speed Physical Layer Ext. in the 2.4 GHz band.September 1999.
7. Std 802.11a-1999. Wireless LAN Medium Access Control and Physical Layer specifications High-speed Physical Layer in the 5 GHz Band. September 1999.
8. Burke, G. J.; Poggio, A. J. Numerical Electromagnetics Code (NEC) Method Of Moments Part II Program Description Code. UCRL- Livermore, CA: Lawrence Livermore National Laboratory. 1981
9. Sinnema, William; Electronic Transmission Technology, Lines, waves, and antennas. Pearson Education POD, 17 February, 1998. Pages 76-78, 275
10. Koza, John R. 1994a. Genetic Programming II: Automatic Discovery of Reusable programs. Cambridge, MA: MIT Press.
11. Multilayer Ceramic Antenna for Bluetooth/Wlan IEEE 802.11b & WLAN IEEE 802.11a (2.45/5.2GHz) (Surface Mounted Ceramic Dual Band Antenna) Product Specification. Taiwan: Phycomp Taiwan LTD. September, 2001
12. http://www.murata.com/nproduct/6104.html
13. http://www.centurion.com/antennaProd/whisper5ghz.asp
14. http://www.hyperlinktech.com/web/hg5308_11p.php

A Generic Network Design for a Closed-Loop Supply Chain Using Genetic Algorithm

Eoksu Sim, Sungwon Jung, Haejoong Kim, and Jinwoo Park

Department of Industrial Engineering, Seoul National University,
San 56-1, Shillim-Dong, Kwanak-Gu, Seoul, 151-742, Korea
{ses, jsw25, ieguru}@ultra.snu.ac.kr, autofact@snu.ac.kr

Abstract. Recently much research has focused on both the supply chain and reverse logistics network design problem. The rapid progress in computer and network technology and the increasingly fierce competition in recent times have compelled global company to consider these two networks in integrated view for efficient decision-making throughout the supply chain. The integrated problem, however, resembles a combinatorial problem, whose computation time to obtain an optimal solution increases exponentially in proportion to the size of the problem. Therefore, an algorithm able to generate a relatively good solution within a reasonable time is needed. In this study, we propose an LP-based genetic algorithm. The experimental results show that the proposed algorithm is superior to MIP solver in time and to traditional genetic algorithm in quality.

1 Introduction

In a world of finite resources and disposal capacities, recovery of used products and materials is key to supporting a growing population at an increasing level of consumption [1]. Reverse logistics is the process of planning, implementing, and controlling the efficient, cost effective flow of raw materials, in-process inventory, finished goods and related information from the point of consumption to the point of origin for the purpose of recapturing value or proper disposal [2]. This process is sometimes treated as an extension of the traditional concept of supply chain management (SCM). In this case, reverse logistics does not treat only the return process from the customer to the manufacturer, but also includes the SCM processes, resulting in a new concept, which is referred to as the 'closed-loop supply chain (CLSC)'.

There are many issues involved in CLSC. Within the framework of the CLSC, there are two major flows to be considered. These are the traditional forward flow from the manufacturer to the customer and the reverse flow from the customer to the manufacturer. The former is referred to as traditional SCM and the latter as reverse logistics. It is not necessary, however, to treat these flows separately. By considering these two flows as constituting a CLSC, we can consider them in an integrated way. Several of the facilities in the CLSC should be considered as the most important sites, because they constitute a virtual enterprise that gives more efficient results to the whole supply chain.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1214–1225, 2004.

Recently, some research has been done on supply chain network design problem using GA. Chu [3] proposed a new approach to the digital data service network design problem using GA. As in the logistics problem, communication networks have some similar characteristics, such as location and flow decision. The authors compared the results of Tabu search methods. Gen [4] summarized research works on network design problem using GA, including transportation problem, centralized network design issues etc. The authors showed the effectiveness and efficiency of the GA-based approach by performing a large number of numerical experiments. Jaramillo [5] evaluated the performance of GA as an alternative procedure for generating optimal or near-optimal solutions for location problem. Four kinds of problems were considered. The performance of GA-based heuristics was compared with well-known heuristics from the literature. Zhou [6] proposed a new model for designing supply chain networks that optimized the best balance of transportation costs and customer service. The authors made a tree and constructed the solution using Prüfer numbers. Using experiments, the efficiency of the proposed algorithm was shown. These studies, however, usually considered the forward supply chain, and so did not take into consideration the return process.

In this paper, we consider an extended model for closed-loop supply chains taking into consideration potential facilities, multi-commodity aspects, the planning period, and the reverse flow of the return products, in such a way to minimize the overall costs, which consist of transportation costs, operating costs and production/storing costs. We consider the facilities as important decision factors, generate models consisting of all components integrally, propose an LP-based Genetic Algorithm (GA) solution heuristic to solve this problem and perform computational experiments to show the efficiency of this heuristic.

The rest of the paper is organized as follows: Section 2 describes problem definition and mathematical model. The proposed solution procedures are described in Section 3. Section 4 reports on the computational experiments. Finally, Section 5 provides concluding remarks and suggests future research direction.

2 Problem Definition

2.1 Closed-Loop Supply Chain

Fleischmann [1] proposed a generic recovery network model (RNM). The author concentrated on the number of facilities, their locations and the allocation of the corresponding goods flow and formulated an MILP optimization problem. However, the model was kept as simple as possible, by adapting an uncapacitated, single-period, single-commodity formulation. That is, RNM allowed the modeling of reverse logistics situations. Nowadays, other more realistic conditions involved in modeling the whole supply chain need to be considered, for example the transportation mode or planning period. Time-phased planning is important, since the structural decisions made in designing supply chains are irreversible. Also, the possible transportation mode (car, rail, ship or airplane, etc.) is an important factor, because not all of the possible transportation modes can be used or the transportation costs may vary in any planning period. Note that the transportation mode in each planning period is

significant, such as in the case of global manufacturing companies. So, there is a need to include the various transportation modes when modeling the network design problem.

In this paper, we consider a generic closed-loop supply chain network model, which is basically an extended version of RNM. As shown in Fig. 1, we consider three intermediate facilities. More precisely, we include the 'reverse center', where the inspection, disassembly and remanufacturing of products are carried out, 'plants' where final products are manufactured and the 'distribution center (D/C)' used for delivering the products to various customers.

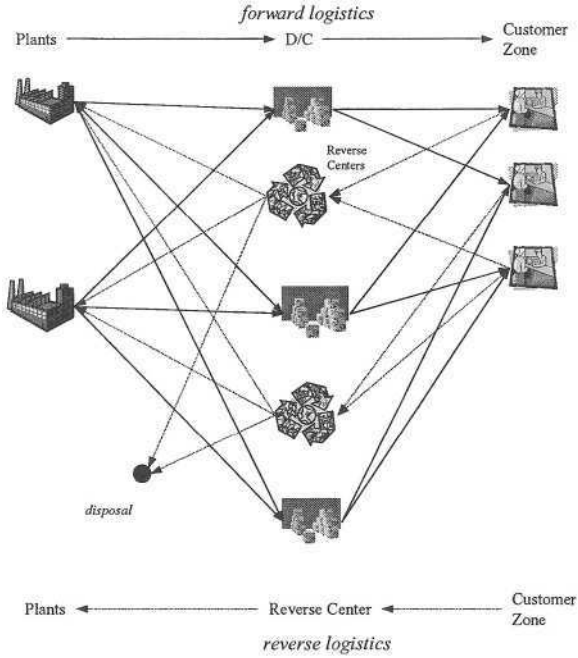


Fig. 1. Structure of the closed-loop supply chain

We assume that the customer demand for each final product is known. The plants produce the final products and deliver these products through the transportation channel to each D/C. The D/Cs also take the products from the plants according to the customer demand quantity and deliver the products to each customer zone. In each time period, we assume that there might exist a return rate for each product. This depends on 1) the final products themselves, due to their limited durability or product life cycle time, 2) the planning period, because of seasonal demand patterns or new product developments and 3) the customer zone, due to the regional propensity to consume the products. We use the return rate for each product in our model. In each reverse center, the returned products would be recycled and decomposed into sub products following the Bill of Materials (BOM) structure. And recycled sub products would be delivered to the plants that want to use them. In each transportation channel,

there would exist some constraints; volume or weight constraints and, therefore, not all of transportation modes would be possible. So we assume that there are four types of transportation mode; car, rail, ship and airplane.

In this framework, the objective is to decide the number of facilities, their locations and the allocation of the corresponding goods flow using a specific transportation mode in each planning period. We formulate this problem into an MIP (Mixed Integer Programming) optimization problem, which is similar to a traditional network design problem. We model the potential facility locations as binary variables and the quantities related to the final/sub products as continuous decision variables in each period.

Our objective is to find a solution that minimizes the sum of the transportation and operation costs. Fixed operating costs, e.g., for acquiring land, for infrastructure construction or for leasing existing facilities, will be incurred when these facilities are used. Although economies of scale might be involved in the case of the transportation costs, we presume that this would not have a significant effect on the strategic locations and the decisions affecting the design of the network. Therefore, the variable costs of transportation are assumed to be linear [7]. We consider that the capacity constraint of each facility during any period might be due to a variety of reasons e.g., availability of land or usage contract.

2.2 Mathematical Formulation

A mathematical model to find the best solution of network design for closed-loop supply chain is structured as follows.

$$Min\ Z = \sum_t \sum_m \sum_l \sum_j \sum_i pc^t_{ijlm} PX^t_{ijlm} + \sum_t \sum_m \sum_l \sum_k \sum_j dc^t_{jklm} DY^t_{jklm} + \sum_t \sum_m \sum_l \sum_n \sum_k cc^t_{knlm} CY^t_{knlm} +$$
$$\sum_t \sum_m \sum_{i'} \sum_i \sum_n rc^t_{nii'm} RX^t_{nii'm} + \sum_t \sum_i pf^t_i PZ^t_i + \sum_t \sum_j df^t_j DZ^t_j + \sum_t \sum_n rf^t_n RZ^t_n +$$
$$\sum_t \sum_l \sum_i pb^t_{it} (\sum_j \sum_m PX^t_{ijlm}) + \sum_t \sum_l \sum_j dh^t_{jt} (\sum_k \sum_m DY^t_{jklm}) + \sum_t \sum_l \sum_n rb^t_{nl} RQ^t_{nl}$$

The objective function minimizes the sum of the costs required to transfer the products from the source sites to the destination sites, the fixed costs of the operating facilities, and the production and storage costs at various sites. The transportation costs consist of transferring the final products from the plants to the distribution center (D/C), from the D/C to the customer, and of transferring the returned products from the customer to the reverse center(R/C), and the recycled sub products from the R/C to the plant. To decide which facility to use in any planning period, we consider the operating costs of all of the potential facilities and the operation costs of production and storage.

Constraints are as follows.

$$\sum_{m}\sum_{j} DY^{t}_{jklm} = d^{t}_{kl} \qquad for\ \forall k,l,t \qquad (1)$$

$$\sum_{m}\sum_{n} RX^{t}_{nil'm} \geq a^{t}_{il'} \qquad for\ \forall i,l',t \qquad (2)$$

$$\sum_{l} ps^{t}_{il}\left(\sum_{j}\sum_{m} PX^{t}_{ijlm}\right) \leq PU^{t}_{i} PZ^{t}_{i} \qquad for\ \forall i,\ t \qquad (3)$$

$$\sum_{l} ds^{t}_{jl}\left(\sum_{k}\sum_{m} DY^{t}_{jklm}\right) \leq DU^{t}_{j} DZ^{t}_{j} \qquad for\ \forall j,\ t \qquad (4)$$

$$\sum_{l} rs^{t}_{nl} RQ^{t}_{nl} \leq RU^{t}_{n} RZ^{t}_{n} \qquad for\ \forall n,\ t \qquad (5)$$

$$\sum_{l} RQ^{t}_{nl} * q_{ll'} \geq \sum_{i}\sum_{m} RX^{t}_{nil'm} \qquad for\ \forall n,l,t' \qquad (6)$$

$$\sum_{m}\sum_{k} DY^{t}_{jklm} \leq \sum_{m}\sum_{i} PX^{t}_{ijlm} \qquad for\ \forall j,l,t \qquad (7)$$

$$\sum_{m}\sum_{n} CY^{t}_{knlm} = r^{t}_{kl} * \sum_{m}\sum_{j} DY^{t}_{jklm} \qquad for\ \forall k,l,t \qquad (8)$$

$$\sum_{i} \mathrm{PZ}^{t}_{i} \leq \mathrm{PW} \qquad \text{for } \forall t \qquad (9)$$

$$\sum_{j} DZ^{t}_{j} \leq DW \qquad for\ \forall t \qquad (10)$$

$$\sum_{n} RZ^{t}_{n} \leq RW \qquad for\ \forall t \qquad (11)$$

$$\sum_{j}\sum_{l} pv^{t}_{l} PX^{t}_{ijlm} \leq PV^{t}_{im} \qquad for\ \forall i,\ m,\ t \qquad (12)$$

$$\sum_{k}\sum_{l} dv^{t}_{l} DY^{t}_{jklm} \leq DV^{t}_{jm} \qquad for\ \forall j,\ m,\ t \qquad (13)$$

$$\sum_{n}\sum_{l} cv^{t}_{l} CY^{t}_{knlm} \leq CV^{t}_{km} \qquad for\ \forall k,\ m,\ t \qquad (14)$$

$$\sum_{i}\sum_{l} rv^{t}_{l} RX^{t}_{nilm} \leq RV^{t}_{nm} \qquad for\ \forall n,\ m,\ t \qquad (15)$$

$$PZ^{t}_{i}, DZ^{t}_{j}, RZ^{t}_{n} = 0/1 \qquad for\ \forall i,j,n,t \qquad (16)$$

$$PX^{t}_{ijlm}, DY^{t}_{jklm}, CY^{t}_{knlm}, RQ^{t}_{nl}, RX^{t}_{nilm} \geq 0 \qquad for\ \forall i,j,k,l,m,t \qquad (17)$$

Constraints (1) and (2) satisfy the demand requirements of the customer and the plant for the final and sub product, respectively. Constraints (3)~(5) limit the capacity of each facility according to its type. Constraint (6) ensures that the recycled products present at the R/C are transferred to their demand points; constraint (7) reflects the flow conservation such that the inflow quantity to the D/C should exceed the outflow quantity and constraint (8) requires that the predefined return rate of final products be used as the return quantity from the final customer. Constraints (9)~(11) ensure the open limit of each facility during any time period. Constraints (12)~(15) reflect the transportation constraints, in that no transportation channel can transfer more than the prespecified quantity of product during any one planning period. The binary decision variable constraint is (16) and the continuous variable constraint is expressed in (17).

This generic formulation can be extended to reflect any other constraint. If there are some weight constraints to be considered in the transportation channel, then some supplementary constraints, similar to constraints (12)~(15), can be added. Refer to Appendix I for further explanation of the variables in the above formulas.

3 Solution Methodologies

This network design problem is an extension of the traditional uncapacitated facility location problem (UFLP). UFLP was shown to be NP-complete by Krarup [8], which implies that the problem that we are studying also belongs to the NP-complete class of problems. Therefore, we propose an efficient heuristic algorithm for solving this kind of problem using LP-based Genetic Algorithm.

3.1 LP-Based Genetic Algorithm

The more decision variables there are, especially binary variables, the more time it takes to solve an MIP problem. In the case of the closed-loop supply chain network design, there are a lot of decision variables involved, so we need to generate an efficient heuristic. If there are no binary variables, the problem becomes an LP problem. So first, we relax the binary variables in the MIP problem and solve the resulting LP-relaxed problem. Using these results, we make an initial population that forms the input data to the GA. The binary variables are converted to 0/1 values by performing the following steps: (1) set the Lower Value (LV) to the sum of the relaxed values, (2) let the Upper Value (UV) be equal to the facility open limit, (3) divide the each relaxed value by the LV and use this value as the weight and (4) assign a value of 0 or 1 to each binary variable in proportion to the weight so that the total number of 1-value variables is between the LV and the UV. In this way, we can make the initial population be closer to the feasible solution group. Based on these generated initial solutions, we can find the final solution through evolutionary process.
The proposed algorithm is as follows.

Step 1: Generate LP-relaxed formulation
Step 2: Solve the relaxed problem using LPSolver
Step 3: Set t=0
Step 4: Generate initial population using the result of the relaxed problem, P(t)
Step 5: Evaluate each of the strings in P(t)
Step 6: While (not termination condition) do
Step 7: Select two parents P1 and P2 from P(t) using selection criteria
Step 8: Apply genetic operators
Step 9: Repeat step 7 and step 8 until a new population is generated
Step 10: Evaluate this new population
Step 11: Set t=t+1 and go to Step 6
Step 12: Generate the final optimal solution

As shown in Jaramillo [5], we use two termination criteria: either the GA converged (i.e. the improvement in the objective function value fell below the tolerance limit of 10^{-5}) or it was executed for a prespecified number of interventions.

3.2 Genetic Operators

3.2.1 Representation Scheme

To design a GA for a particular problem, we first need to devise a suitable representation scheme that shows the solution characteristics. In solving the location problem using a GA, previous researchers usually defined the chromosomes as being the potential sites [5], [6]. In this paper, we developed a representation scheme that consisted of 2-dimensional binary strings as the chromosome structure. This chromosome structure represents both the potential facility sites and planning time periods. A value of 1 for the i^{th} time period and j^{th} facility site implies that the j^{th} facility is located and used in i^{th} period. The 2-dimensional binary representation of an individual's chromosome is illustrated in Fig. 2.

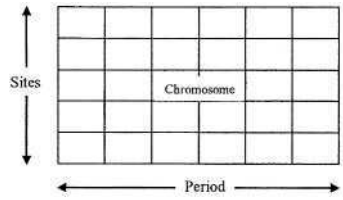


Fig. 2. Expression of a solution

3.2.2 Fitness Function and Parents Selection

The fitness function is a measure of goodness of a solution. In this paper, we use the fitness function as its objective function. For the crossover operation, we select two parent solutions. The various selection operators have one principle in common, in that the probabilistically superior solution is the one to be chosen. In this study, we used the simplest and most frequently used roulette wheel method [9].

3.2.3 Crossover Operator

Crossover combines the features of two parent chromosomes to form two similar offsprings by swapping the corresponding segments of the parents. With the application of the crossover operator, the genetic information pertaining to the selected parent solutions is passed on to the offspring in various combinations. In this study, we choose the one-point crossover operation, the most widely known and simple, in order to form better offspring that inherit superior genetic information from the parent solutions. In addition to the general crossover operation, we consider the feasibility of constraint on the number of open facilities. In any time period, there is a limit to the number of open facilities, so we use the non-linear one-point crossover operator, as shown in Fig. 3.

3.2.4 Mutation Operator

Mutation arbitrarily alters one or more genes of a selected chromosome, by a random change with a probability equal to the mutation rate. It can be used as a policy to prevent solutions from being trapped in local optima. In this study, we randomly select one of the open facilities in a certain period and transfer this state of being open

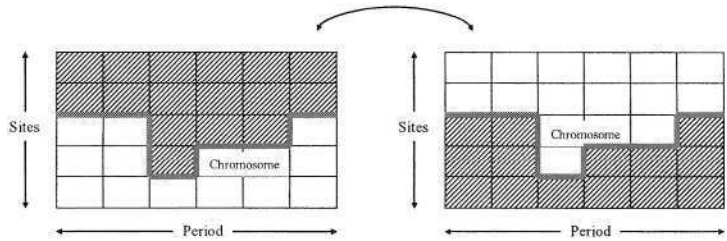


Fig. 3. Non-linear one-point crossover operator

to another facility and select the widely used non-uniform mutation as the mutation operator. This results in the mutation rate being gradually decreased as the algorithm proceeds and thereby raises the convergence speed of the solution with the unfolding evolution.

4 Computational Experiments

In performing the computational experiments, we wanted to show (1) the relative complexity of the solution in relation to the complexity of the problem, (2) the performance superiority of the proposed algorithm and (3) the statistical significance of the difference between the solution qualities.

We use the LP-based GA, so we need to run the algorithm multiple times, because GAs have stochastic characteristics. The problem data sets were generated randomly, but systematically, to reflect the system environment. We generated a set of 100 problems in order to experiment the algorithm.

The number of facility open limits in each period is specified as experimental parameters, and four periods are considered as the planning horizon. For experimental simplicity, we set the number of final products to 2. Each final product can have one or two subproducts. The number of transportation mode in each transportation channel is generated randomly from 1 to 4. The population size and the number of generations are set to one hundred and one thousand, respectively. The heuristic algorithm developed in this study was coded in C++ on the Pentium IV 1.4GHz computer. The comparison of LP-based, traditional GA with the optimization package (CPLEX 7.0) is also provided in Table 1.

The gap between the best feasible solution that was generated by the algorithm and the optimal solution that was generated by the CPLEX engine can be used as the solution quality criteria.

Table 1 led to an important insight. The LP-based GA heuristic is superior to the MIP solver in terms of the time required to find a solution. For a small size problem, the proposed heuristic has poor quality in time and quality. As the size of the problem increases, however, the MIP solver requires a much longer time to solve the problem. The proposed heuristic can produce an optimal or near-optimal solution in a reasonable time. Although the initial population generated by the LP-relaxed solution

Table 1[1]. Comparison of LP-based, Traditional GA and optimal solution

P	D	C	R	CPLEX		LP-based GA		Traditional GA		No. of
				Time(s)	Obj.	Time(s)	GAP(%)	Time(s)	GAP(%)	var.
2	3	3	2	0.2	1.66	3.32	0.39	4.33	2.25	2340
2	3	5	3	1.63	1.76	7.98	0.77	7.76	1.96	2788
2	3	10	2	10.2	1.83	2.50	0.76	1.60	2.15	3894
2	3	10	3	9.5	1.92	7.30	0.05	8.95	1.08	4770
2	5	10	2	12.2	1.99	8.01	0.99	7.02	2.94	5386
2	5	10	3	20.3	2.08	13.91	0.60	13.59	1.51	6324
3	5	10	2	15.6	2.22	15.11	0.08	14.01	2.70	5915
3	5	10	3	27.11	2.22	23.99	0.59	25.68	1.82	6822
3	5	15	2	21.41	2.36	5.73	0.62	5.96	2.81	8220
3	5	15	3	1195.09	2.54	49.19	0.81	47.49	1.41	10314
3	10	15	2	1769.56	2.64	67.4	0.42	68.83	2.22	13927
3	10	15	3	1856.25	2.79	73.07	0.34	74.98	1.87	15197
5	10	15	2	1648.27	2.83	74.29	1.32	73.81	3.59	15455
5	10	15	3	2087.26	2.84	77.31	1.85	78.87	1.13	16872
5	10	15	4	2415.25	2.87	73.35	0.20	73.99	1.43	17992
5	10	20	2	2435.54	3.00	88.32	1.58	88.44	3.33	19335
5	10	20	3	2428.45	3.03	76.1	1.21	77.81	3.70	20923
5	10	20	4	2711.07	3.14	73.27	1.91	72.07	4.01	22615
5	15	20	2	2864.50	3.15	94.04	2.92	95.12	4.52	27456
5	15	20	3	2985.42	3.26	96.03	2.54	94.77	3.05	28925
5	15	20	4	2845.89	3.29	100.7	1.50	100.81	2.52	30612
8	15	20	2	2931.06	3.45	112.37	2.19	112.42	3.31	30687
8	15	20	3	3080.23	3.57	123.45	1.64	121.13	3.13	32374
8	15	20	4	3190.97	3.69	128.06	2.04	128.86	3.50	34267
10	15	20	2	2934.26	3.79	131.64	2.09	130.98	3.06	32840
10	15	20	3	3268.29	3.93	144.18	1.61	143.64	3.86	34694
10	15	20	4	3394.19	4.00	140.94	1.83	138.17	3.40	36733
10	15	20	5	3394.19	4.12	151.91	1.93	151.70	4.34	38861
10	15	30	3	4926.30	5.18	162.29	2.08	162.18	4.61	46309

could not guarantee the optimality of the final solution, the solution obtained by the LP-relaxed GA was closer to the optimum than the traditional GA solutions.

[1] GAP = (heuristic solution value – optimal value)/optimal value* 100
P: Plant, D:D/C, C: Customer, R: Reverse Center, Obj.: Objective Value($\times 10^{+5}$).

In Fig. 4 and Fig. 5, a comparison is made between the solution time and quality for each method. As shown in Fig. 4, the more complex the problem becomes, the more time is needed to solve the problem. Fig. 5 shows that the LP-based GA is superior to traditional GA in terms of solution quality.

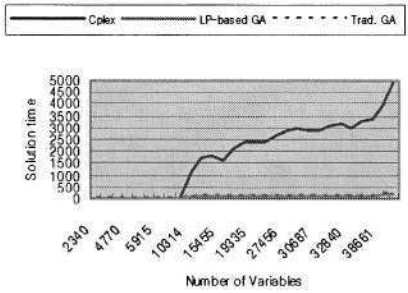


Fig. 4. Comparison of solution time

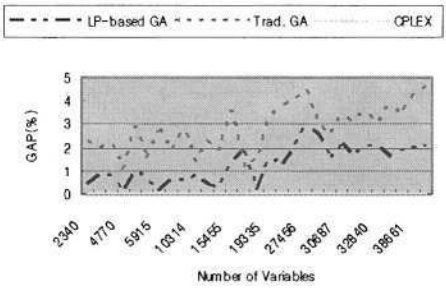


Fig. 5. Comparison of solution GAP

In table 2 and Table 3, the solution quality and time in LP-based/traditional GA and optimal solution package results are compared. In these paired t-test results, the difference in quality between LP-based GA(LGA) and MIP solver is not statistically significant. So, we can support that there is no difference in solution quality performance between them. In terms of solution time, the differences between LP-based GA and traditional GA(TGA) are not significant.

Table 2[2]**.** Paired *t*-test in solution quality

	TGA	LGA	CPLEX
TGA		S	N.S
LGA	S		N.S
CPLEX	N.S	N.S	

Table 3[2]**.** Paired *t*-test in solution time

	TGA	LGA	CPLEX
TGA		N.S	S
LGA	N.S		S
CPLEX	S	S	

In real world problem, there are much more decision variables in closed-loop supply chain network design problem. In that case, the MIP solver cannot obtain optimal solution in a reasonable time. The proposed LP-based GA can find a near-optimal solution in an affordable time.

5 Conclusion

In this paper, we consider an extended supply chain network design model that allows for the reverse logistics. To solve this problem, we propose an LP-based genetic algorithm that uses the LP-based solution and genetic operators. In order to evaluate the superiority of the proposed method, we performed an experiment. Through this

[2] Significance level = 0.05, S: significant, N.S : non-significant.

experiment, we compare the results from MIP solver (CPLEX), traditional GA (TGA) and the LP-based GA (LGA).

The result shows that the time taken to obtain optimal solution using CPLEX increases exponentially as the problem size grows while LGA and TGA reduces the influence of the problem size on the time. In terms of cost, it shows that the solution obtained by LGA is as good as the optimal solution solved by CPLEX while it is superior to the solution obtained by TGA.

In real time application aspects, the size of the problem is much larger than that of the test data. In such environments, it is infeasible to find an optimal solution in a reasonable time. Hence the proposed LP-based genetic algorithm can be usefully used in real situation.

But more studies are required to solve some weakness in LGA. In evolutionary process, genes have high correlation among them, so there can be some perturbation in the process. To reduce this perturbation, further researches should focus on the design of the operators themselves.

References

1. Fleischmann, M.: Quantitative models for reverse logistics. Springer-Verlag, Berlin (2001)
2. Stock, J. : Development and Implementation of Reverse Logistics Programs. Council of Logistics Management. USA (1998)
3. Chu, C.H., Premkumar, G.., Chou, H.: Digital data networks design using genetic algorithms. European Journal of Operational Research 127 (2000) 140-158
4. Gen, M., Cheng, R., Oren, S.S.: Network design techniques using adapted genetic algorithms. Advanced in Engineering Software 32 (2001) 731-744
5. Jaramillo, J.H., Bhadury, J., Batta, R.: On the use of genetic algorithms to solve location problems. Computers & Operations Research 29 (2002) 761-779
6. Zhou, G.., Min, H., Gen, M.: The balanced allocation of customers to multiple distribution centers in the supply chain network: a genetic algorithm approach. Computers & Industrial Engineering 43 (2002) 251-261
7. Verter, V.: An integrated model for facility location and technology acquisition. Computers & Operations Research 29 (2002) 583-592
8. Krarup, J., Pruzan, P.M.: The simple plant location problem: survey and synthesis. European Journal of Operational Research 12 (1983) 36-81
9. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. 3rd edn. Springer-Verlag, Berlin Heidelberg New York (1996)

Appendix I

Indices

I : set of possible locations for plant, indexed by $i \in I$, J: set of possible locations for D/C, indexed by $j \in J$

K : set of locations for customers(customer zones), indexed by $k \in K$, L : set of products, indexed by $l \in L$

N : set of possible locations for reverse logistic centers, indexed by $n \in N$ T : time period

M : transportation mode

Decision Variables

PX^t_{ijlm}	transportation quantity of product l from plant i to D/C j at time period t using transportation mode m
DY^t_{jklm}	transportation quantity of product l from D/C j to customer k at time period t using transportation mode m
CY^t_{knlm}	transportation quantity of product l from customer k to reverse center n at time period t using transportation mode m
$RX^t_{nil'm}$	transportation quantity of product l from reverse center n to plant i at time period t using transportation mode m
PZ^t_i	Indication as to whether a plant i is open at time period t
DZ^t_j	Indication as to whether a D/C j is open at time period t
RZ^t_n	Indication as to whether a reveres center n is open at time period t
RQ^t_{nl}	processing quantity of product l at reverse center n at time period t

Parameters

pc^t_{ijlm}	transportation cost per unit of product l from plant i to D/C j at time period t using transportation mode m
dc^t_{jklm}	transportation cost per unit of product l from D/C j to customer k to at time period t using transportation mode m
cc^t_{knlm}	transportation cost per unit of product l from customer k to reverse center n at time period t using transportation mode m
$rc^t_{nil'm}$	transportation cost per unit of product l from reverse center n to plant i at time period t using transportation mode m
pf^t_i	fixed, operating cost for plant i at time period t
df^t_j	fixed, operating cost for D/C j at time period t
rf^t_n	fixed, operating cost for reverse center n at time period t
pb^t_{il}	production cost for product l in plant i at time period t
rb^t_{nl}	disassembly cost for product l in reverse center n at time period t
dh^t_{jl}	holding cost for product l in D/C j at time period t
d^t_{kl}	demand quantity for product l in customer (zone) k at time period t
$a^t_{il'}$	demand quantity of recycled product l in plant i at time period t
ps^t_{il}	required production capacity per unit of product l in plant i at time period t
ds^t_{jl}	required inventory holing capacity per unit of product l in D/C j at time period t
rs^t_{nl}	required reverse processing capacity per unit of product l in reverse center n at time period t
PU^t_i	total available production capacity in plant i at time period t
DU^t_j	total available holding capacity in D/C j at time period t
RU^t_n	total available reverse processing capacity in reverse center n at time period t
PW	upper limit on the number of open plant
DW	upper limit on the number of open D/C
RW	upper limit on the number of open reverse center
r^t_{kl}	return rate of product l for customer k at time period t
$q_{ll'}$	BOM quantity of child product l to parent product l
$pv^t_l, dv^t_l, cv^t_l, rv^t_l$	unit volume of product l(l) at time period t
PV^t_{im}	total volume capacity of transportation mode m from plant i at time period t
DV^t_{jm}	total volume capacity of transportation mode m from D/C j at time period t
CV^t_{km}	total volume capacity of transportation mode m from customer k at time period t
RV^t_{nm}	total volume capacity of transportation mode m from reverse center n at time period t

Evolving a Roving Eye for Go

Kenneth O. Stanley and Risto Miikkulainen

Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78705
{kstanley, risto}@cs.utexas.edu
www.cs.utexas.edu/users/{kstanley, risto}

Abstract. Go remains a challenge for artificial intelligence. Currently, most machine learning methods tackle Go by playing on a specific fixed board size, usually smaller than the standard 19×19 board of the complete game. Because such techniques are designed to process only one board size, the knowledge gained through experience cannot be applied on larger boards. In this paper, a *roving eye* neural network is evolved to solve this problem. The network has a small input field that can scan boards of *any* size. Experiments demonstrate that (1) The same roving eye architecture can play on different board sizes, and (2) experience gained by playing on a small board provides an advantage for further learning on a larger board. These results suggest a potentially powerful new methodology for computer Go: It may be possible to scale up by learning on incrementally larger boards, each time building on knowledge acquired on the prior board.

1 Introduction

The performance of artificial intelligence (AI) methods in Go lags behind other board board games, which makes it a popular and challenging testbed for different techniques [1]. Since designing a strategy of a good player by hand is very difficult, machine learning (ML), i.e. letting the computer learn how to play through experience, is a popular approach [2]. The idea is that a sufficiently powerful ML algorithm can learn strategies and tactics through experience that are otherwise difficult to formalize in a set of rules.

The long-term goal is to train an AI player on the full-sized 19×19 board. However, the game is increasingly difficult on larger boards because the search space grows combinatorially and position evaluation becomes more difficult. Thus, current methods [3,4] have been successful only on smaller boards. Such results demonstrate the potential of a method, with the hope that it may be possible to scale it up to larger boards in the future.

Ideally the knowledge gained on small boards could bootstrap play on larger ones. A process that could make use of such shared knowledge would be a significant step towards creating learners that can scale.

Such a learner is presented in this paper, based on the NeuroEvolution of Augmenting Topologies (NEAT) method of neuroevolution [5], i.e. the evolution

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1226–1238, 2004.

of artificial neural networks of varying complexity. Previous neuroevolution work in evolving Go focused on networks with the number of inputs and outputs chosen for a particular board size, making it difficult or impossible to transfer to a larger board [3,4]. A different approach is taken in this paper; instead of a network that sees the entire board at once, a *roving eye* is evolved with a small visual field that can scan the board at will. While scanning, the roving eye decides when and where to place a piece. Because it has the same field size on *any* board size, a roving eye evolved on a small board can be further evolved on a larger board without loss of generality. Thus, the roving eye architecture promises to fulfill the goal of a scalable learner.

In order to demonstrate the scalability of the roving eye, a competent Go player was evolved on a 5×5 board against Gnugo, a publicly available Go playing program (available at `www.gnu.org/software/gnugo/gnugo.html`). This neural network was then further evolved against Gnugo on a 7×7 board. For comparison, other 7×7 players were separately evolved from scratch. The main result is that the networks pre-evolved on the 5×5 board improved their performance significantly faster than networks evolved from scratch, and reached a higher final level of performance. This result establishes that (1) the roving eye is a scalable architecture, and (2) scaling can lead to better performance than learning directly on a larger board.

In the next section prior work in machine learning and neuroevolution for Go is reviewed, and also prior implementations of "eyes" in AI. Section 3 then briefly describes the NEAT method for evolving neural network topologies and weights. Finally, Section 5 describes the experimental methods and results.

2 Background

While Go is difficult to master, machine learning techniques show promise. However, current methods cannot scale learning from one board size to another. A roving eye can potentially provide such a scaling capability. Although roving eye systems are not currently used in board games, they are commonly used in robotics. This section reviews the current state of machine learning and Go, and prior applications of roving eyes.

2.1 Machine Learning and Go

Go is a difficult two-player game with simple rules, making it an appealing domain for testing machine learning techniques. The standard game is played on a 19×19 grid. Black and white pieces are placed alternately at intersection points on the grid by the two players. The game ends when both players pass, which usually happens when it becomes clear that no further gains can be made. The winner is determined by the final score.

The object of the game is to control more territory than the opponent. Any area completely surrounded by one player's stones is counted as that player's territory. If the opponent's stones are completely surrounded, those stones are

lost and that area is counted towards the other player. If there is an open intersection, called an *eye*, in the middle of the opponent's stones, that intersection must also be filled in order to surround the stones. A structure with two eyes cannot be captured because it is not possible to fill both eyes at once. The *ko rule,* which forbids the same board state from occurring twice, ensures that the game progresses to a conclusion.

The rules of Go are deceptively simple. Yet unlike in many other board games, such as Othello and chess, machines cannot come close to master level performance in Go. Not only are there generally more moves possible in Go than other two-player, complete information, zero-sum, games, but it is also difficult to formulate an accurate evaluation function for board positions [1]. However, the game can be simplified by playing on a smaller board [4]. The smaller the board, the simpler the strategy of the game. At the smallest possible board size, 5×5, the game becomes largely a contest to control one side of the board. However, even at this very small scale, fundamental concepts must be applied, such as forming a line of connected pieces and defending the center. Although these concepts alone are not sufficient to play well on a larger board, they are nevertheless a foundation for more developed strategies, making smaller boards a viable platform for testing machine learning methods. While 5×5 Go may be much simpler than 19×19 Go, it is still related to 7×7 Go, which is related to 9×9 Go, and so on.

Bouzy and Cazenave [1] reviewed a number of general AI techniques not based on learning that have been applied to Go, such as goal generation, game tree search, and pattern-matching [6]. Gnugo 3.2 is a publicly available, open source Go playing program that includes many of these techniques. Gnugo is on par with current commercially available Go playing programs. However, no existing program is even competitive with an amateur shodan, which is the designation for a technically proficient human player [7].

Writing a program to play Go directly is difficult because a large amount of knowledge must be coded into the system. Therefore, machine learning methods that generate their own knowledge through experience are an appealing alternative. For example, Enzenberger [2] created a program called NeuroGo that links units in a neural network corresponding to relations between intersections on a Go board. In 2001, NeuroGo ranked in the top third of computer Go playing programs [1], showing that machine learning is competitive with hand-coded knowledge.

In this paper, neuroevolution (NE) is used to evolve neural networks to play Go. NE has been applied to Go on smaller board sizes in the past [3,4]. However, these experiments evolved neural networks for a single board size, wherein each intersection on the board was represented by a discrete set of inputs and outputs. Such representations cannot easily scale to other board sizes because the number of inputs and outputs in the network are only compatible with the single board size for which they were designed. In contrast, in this paper a roving eye neural network is evolved that uses the same number of inputs and outputs regardless

of the board size. The next sections reviews prior research on roving eye systems outside the game-playing domain.

2.2 Roving Eyes

A *roving eye* is a visual field smaller than the total relevant image area; such a field must move around the image in order to process its entire contents. They are often used in robots, where they allow successful navigation even with a limited sensory radius [8]. This type of purposeful control of a moving visual field is also sometimes called *active vision* [9].

Instead of hardcoding the control laws that guide the movement of the roving eye, Pierce and Kuipers [10] showed that they can be learned through experience in the world. This research sets a precedent for the automatic design of roving eyes when the proper control or processing rules are unknown.

Most relevant to game playing are experiments where a roving eye must learn to recognize objects that are too big to process all at once. Fullmer and Miikkulainen [11], and separately Nolfi [12], trained neural networks using neuroevolution to control situated robots that had to move around an object in order to determine its identity. Fullmer and Miikkulainen evolved simple "creatures" that move around a shape on a grid and must learn to move onto only certain shapes. In Nolfi's experiment, a robot moves around an object in order to determine if it is a wall or a type of cylinder. In both cases, both the movement control and the final classification action were evolved simultaneously as neural network outputs.

It is such neuroevolved shape-recognizing roving eyes that provide inspiration for using a roving eye in a board game. However, instead of performing a simple classification, the eye must scan the board and then decide where and when to place a piece. For such a technique to work, the controller for the eye must have memory: It must relate different parts of the board to each other even though they cannot be within its field at the same time. In neural networks, memory can be retained through recurrent connections. Of course, such recurrent structures are likely to be very complex for a game like Go, even at the smallest board size. For this reason, the NEAT method for evolving artificial neural networks, which can evolve increasingly complex network topologies [5,13], was used to develop the controller for the roving eye. NEAT is briefly reviewed in the next section.

3 NEAT: NeuroEvolution of Augmenting Topologies

Developing a Go-playing neural network can be seen as a search problem. It is not known how complex such a network needs to be or even what kind of topology it should have. Searching in too large a space, i.e. a space of highly complex networks, would be intractable while searching in too simple a space would limit solution quality. Therefore, the NeuroEvolution of Augmenting Topologies (NEAT) method [5], which automatically evolves network topology to fit the complexity of the problem, is appropriate for this task. NEAT combines the

usual search for the appropriate network weights with *complexification* of the network structure. This approach is highly effective: NEAT outperforms other neuroevolution (NE) methods, e.g. on the benchmark double pole balancing task [5,13]. In addition, because NEAT starts with simple networks and expands the search space only when beneficial, it is able to find significantly more complex controllers than fixed-topology evolution, as demonstrated in a robotic strategy-learning domain [14]. These properties make NEAT an attractive method for evolving neural networks in complex tasks. In this section, the NEAT method is briefly reviewed; see e.g. [5,13,14] for a complete description.

NEAT is based on three key ideas. First, evolving network structure requires a flexible genetic encoding. Each genome in NEAT includes a list of *connection genes,* each of which refers to two *node genes* being connected. Each connection gene specifies the in-node, the out-node, the weight of the connection, whether or not the connection gene is expressed (an enable bit), and an *innovation number,* which allows finding corresponding genes during crossover. Mutation can change both connection weights and network structures. Connection weights mutate as in any NE system, with each connection either perturbed or not. Structural mutations, which allow complexity to increase, either add a new connection or a new node to the network. Through mutation, genomes of varying sizes are created, sometimes with completely different connections specified at the same positions.

Each unique gene in the population is assigned a unique innovation number, and the numbers are inherited during crossover. Innovation numbers allow NEAT to perform crossover without the need for expensive topological analysis. Genomes of different organizations and sizes stay compatible throughout evolution, and the problem of matching different topologies [15] is essentially avoided.

Second, NEAT speciates the population, so that individuals compete primarily within their own niches instead of with the population at large. This way, topological innovations are protected and have time to optimize their structure before they have to compete with other niches in the population. The reproduction mechanism for NEAT is *explicit fitness sharing* [16], where organisms in the same species must share the fitness of their niche, preventing any one species from taking over the population.

Third, unlike other systems that evolve network topologies and weights [17, 18] NEAT begins with a uniform population of simple networks with no hidden nodes. New structure is introduced incrementally as structural mutations occur, and only those structures survive that are found to be useful through fitness evaluations. This way, NEAT searches through a minimal number of weight dimensions and finds the appropriate complexity level for the problem.

4 Experimental Methods

The experiments are designed to answer two questions: (1) Is the roving eye a scalable architecture, i.e. can it play on more than one board size? (2) If so,

can learning to play on a small board facilitate further evolution on a larger board? Specifically, does a proficient 5×5 player provide a better starting point for evolving a 7×7 player than evolving from scratch?

4.1 Evolving Against Gnugo

Roving eye neural networks were evolved to play black against Gnugo 3.2 with Japanese scoring. Gnugo is deterministic; it always responds the same way to the same moves. Thus, our solutions were not evolved to be general-purpose players (although they did evolve some general skills), but rather to defeat Gnugo. While it is possible to devise a more general-purpose competition, e.g. by using coevolution, playing Gnugo allows easy interpretation of results and clear illustration of scalability against a known benchmark.

Another advantage of Gnugo is that it estimates the score for every move of the game, as opposed to only the last one. This estimate makes fitness calculation more effective, because the quality of play throughout the game can be taken into account, instead of only at the end. Fitness was calculated from the cumulative score estimate as well as the final score as follows:

$$f = 100 - \left(\frac{2\sum_{i=1}^{n} e_i}{n} + e_f\right), \tag{1}$$

where e_i is the score estimate on move i, n is the number of moves before the final move, and e_f is the final score. This fitness equation weighs the average estimated score twice as much as the final score, emphasizing the performance over the course of the entire game over the final position. Such a fitness allows selecting promising networks even early in evolution when the network is likely to lose all its pieces. Because Gnugo returns positive scores for white, they must be subtracted from 100 to correlate higher fitnesses with greater success for black.

Neural networks were evolved to control a roving eye in both 5×5 and 7×7 evolution. In half the 7×7 evolution runs, the champion of the 5×5 run was used to seed to initial population, i.e. the 5×5 champion's connection weights were slightly mutated to form the initial population for 7×7 evolution.

A special modification had to be made in order make Gnugo a useful opponent: Because Gnugo will pass as soon as it determines that it cannot win, Gnugo will pass in 5×5 Go as soon as black plays in the middle. Therefore, to force a continuing game, white was forced to play the intersection directly adjacent and right of center on its first move. That way, Gnugo would play a full game. In order to be consistent, white was forced to make the same initial move on the 7×7 board as well. This modification does not affect the generality of the results since the aim is to encourage the roving eye to improve by playing full games against Gnugo, which was effectively accomplished.

4.2 Roving Eye

The roving eye is a neural network evolved with NEAT. The neural network's sensors are loaded with the visual field of the roving eye, and its outputs de-

termine how the eye should move, or stop moving and decide where to place a piece.

The state of the roving eye consists of its position and heading. It can be positioned at any intersection on the board, and be heading either north, south, east, or west. The ability to see the board from different headings allows the roving eye to process symmetrical board configurations the same way. In other words, unlike full-board neural network players, the roving eye does not need to evolve separate components for symmetric positions, greatly reducing the burden on evolution. Instead, the eye can simply turn around to see the board from an identical perspective.

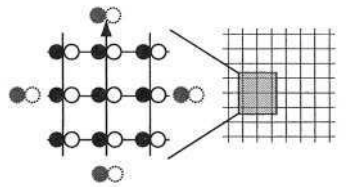


Fig. 1. The Roving Eye Visual Field. At each of the nine intersections visible to the roving eye, it has one black sensor and one white sensor, depicted as filled circles. In addition, the dotted circles represent long-range front, left, right, and back sensors, which only tell how many pieces of each type are in each of those four zones outside the visual field. The arrow shows the eye's current heading. The gray square on the 7×7 board shows the size of the visual field relative to the board. The roving eye also receives 2 inputs representing absolute position, an illegal move sensor, and a bias. This architecture allows the same roving eye to operate on any board size, making scalable Go players possible.

The visual field includes nine intersections with the eye positioned at the center (figure 1). For each intersection, the eye receives two inputs. If the intersection is empty, both are zero. For a black or white stone, the first or second input is fully activated. For a border, both inputs are active. In addition, to help the eye decide where to look, it is given a count of how many white and black pieces are outside its visual field to its front, left, right, and back. The eye is also fed two inputs representing its absolute position on the board, a single input that specifies whether playing in the current position would be illegal due to ko, and a constant bias. Thus, in total, the eye receives 30 inputs regardless of board size.

Five outputs determine the eye's next action. It can place a piece at its current position, move forward in its current heading, turn left, turn right, pause (which may give it more time to "think," i.e. process recurrent activations), or pass. If any of the movement outputs are above 0.5, the eye will move regardless of the other output values, and the movements are combined. For example, the eye can move forward and turn left at the same time. Otherwise, the greatest output over 0.5 is chosen. If no output is sufficiently active, the eye pauses for one time

step. If after 100 time steps the eye still does not make a choice, it is forced to pass. Finally, if the roving eye attempts to place a piece on top of another piece, the move is considered a pass.

In general, the roving eye scans the board by moving around until it finds an acceptable location, and then places a piece. In this way, the roving eye analyzes the board similarly to humans, who also focus attention on specific areas of the board while considering a move rather than processing the entire board at once.

4.3 NEAT System Parameters

Because population dynamics can be unpredictable over hundreds of generations, a target of 20 species in the population of 400 networks was assigned to NEAT evolution. The champion of each species with more than five networks was copied into the next generation unchanged. The interspecies mating rate was 0.05. The probability of adding a new node was 0.0045 and the probability of a new link mutation was 0.1. These parameters were found through systematic experimental search. Except when the 5×5 champion was used to start 7×7 evolution, the starting genome had several sensors initially disconnected. That way, NEAT could start in a lower-dimensional space and decide on its own which sensors to use. Specifically, the out-of-range sensors all started out disconnected, and the front-left, front-right, back-left, back-center, and back-right sensors were disconnected from the network. However, these sensors were still present in the genome and were frequently eventually connected to the network by NEAT over the course of evolution.

5 Results

In every run, the roving eye was playing black and Gnugo white. Ten runs of 5×5 evolution were performed. The champion of a typical run, with fitness 99, was chosen as the seed genome for 15 runs of 7×7 evolution. Another 15 runs of 7×7 evolution were started from scratch, without the seed.

5.1 5×5 Champion

NEAT improved significantly against Gnugo in 5×5 evolution. In early generations, NEAT rarely placed a piece without losing it soon after. By the 400th generation, NEAT was able to control the center of the board and thereby capture more area than Gnugo. NEAT learned the general principle of connectedness, and also the importance of maintaining a forward front. The champion roving eye, whose game is shown in figure 2, was used as the seed for further evolution on the 7×7 board.

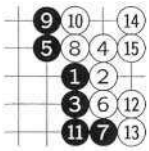
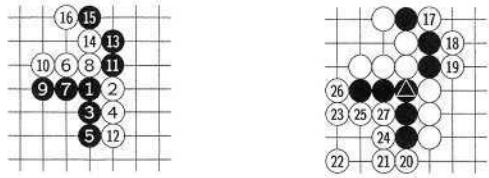


Fig. 2. A Game by the 5×5 Champion. The roving eye (black) is able to control more of the board by pushing its border as far to the right as possible against Gnugo. Gnugo is unable to mount an attack, and instead reinforces its position by creating two eyes. This roving eye was used as the starting point for evolution on the larger 7×7 board.



(a) Black attempts a border (b) Black is surrounded

Fig. 3. The 5×5 Champion Plays Gnugo on a 7×7 Board. (a) The roving eye (black) attempts to form a border as it did in 5×5 Go (figure 2). However, because the board is larger, it only partially succeeds. (b) Never having experienced such a conclusion, black does nothing but pass while Gnugo easily eliminates its opponent. The game shows that the 5×5 roving eye is able to apply some of its knowledge to 7×7 Go, although its former strategy is no longer successful.

Figure 3 shows what happens when the 5×5 champion plays directly against Gnugo on a 7×7 board. Interestingly, the champion shows some of its 5×5 capabilities, forming a semi-contiguous line. However, when its line fails to connect, and is not sufficient to cover the larger 7×7 board, the roving eye is quickly surrounded by Gnugo without making any attempt to respond. This behavior makes sense because in the 5×5 game, as soon as the roving eye finished building its border, Gnugo would not attack and the game would end (see figure 2). However, in 7×7 Go, this strategy is no longer sufficient. The question is whether the knowledge contained in the 5×5 champion will benefit further NEAT evolution on the 7×7 board.

5.2 Evolving 7×7 Players

Evolution on the 7×7 board is indeed significantly more effective starting from the pretrained 5×5 champion than starting from scratch (figure 4). Not only does initial fitness rise significantly faster over the first few generations, but it remains higher throughout the run, suggesting that starting raw may never reach the performance level of a roving eye that has already learned about Go on a smaller board. This result establishes both that (1) the roving eye can be used on more than one board size, and (2) evolving on a smaller board captures information that is useful on a larger board.

During 7×7 evolution, NEAT was able to reorganize and expand the original 5×5 strategy in order to form an effective border (figure 5). While evolving from

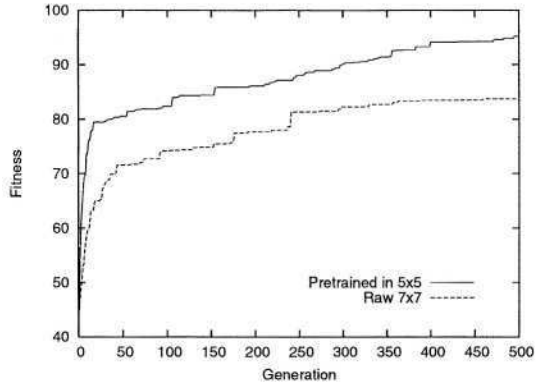


Fig. 4. Average Fitness on a 7×7 Board over Generations. The average fitness (equation 1) of the best roving eye in each generation is shown over 15 runs of raw 7×7 evolution, and 15 runs of 7×7 evolution pretrained in 5×5 Go. Pretrained evolution has significantly $(p < 0.05)$ higher fitness in generations 2–237, 383–451, and 495–500, showing that pre-evolving on a smaller board leads to both a higher start and end to evolution.

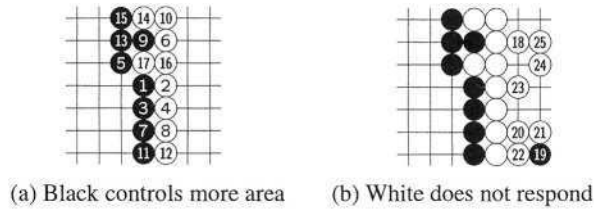


Fig. 5. Typical 7×7 Champion Pretrained in 5×5. The figure shows two halves of a game played by a 7×7 champion with fitness 104 that descended from the 5×5 champion (figure 2). (a) The roving eye has learned to form a clean border with more territory enclosed than Gnugo. (b) The roving eye plays one piece in the corner, because it has evolved to exploit Gnugo's tendency to occasionally place unnecessary pieces (which lowers its score under Japanese rules). Gnugo finishes by creating two eyes, but cannot mount an attack on the roving eye. Thus, the eye finishes with more territory.

scratch required NEAT to discover the capability to connect contiguous intersections, this capability was present from the start when using the pretrained 5×5 champion. In fact, on a larger board, discovering the same early principles takes longer because the game is significantly more complex. Therefore, the raw-starting roving eye was at a disadvantage throughout evolution. In contrast, the pretrained roving eye quickly rises within 25 generations to a level of play that takes raw-starting evolution ten times longer to achieve!

6 Discussion and Future Work

The roving eye allows scaling skills learned on smaller boards to larger boards. This is an important result because current techniques do not succeed in learning to play on larger boards. Therefore, roving eye neuroevolution could turn out to be an important component of a competent learning system for Go.

The level of play demonstrated in this paper is not at championship level. Since black has the first move, it is not surprising that it ultimately controls more area. In addition, it has only learned to play against Gnugo, and would likely fail against a more skilled or significantly different opponent. An important question for the future is how one might evolve a world-class Go player using such a scaling technique. That is, can we apply the roving eye methodology in a way that more general, robust strategies would emerge?

First, the roving eye needs to be evolved on significantly larger boards. There are substantial differences between 7×7 and 19×19 Go: For example, the larger space allows larger patterns to be formed. Evolution will take longer to make use of them, however, rudimentary skills such as the ability to surround enemy pieces or place a contiguous line should still constitute a useful starting point for future learning.

Second, to encourage better generalization, roving eyes could be coevolved instead of evolving them only against Gnugo [7]. Gnugo was used as an opponent in this paper because it is a well-known benchmark, but in the future roving eyes should play against each other to force them to handle a variety of opponent strategies. Well-founded coevolution methodologies such as Host-Parasite, Hall of Fame, or Pareto coevolution should be used to develop the most well-rounded players possible [19,20]. The combination of increasing complexity in NEAT and competitive coevolution has been shown to lead to levels of sophistication unreachable through evolving neural networks of fixed-topology [14], and should work well also in the Go domain.

Third, the roving eye can be combined with game tree search techniques such as α-β. While the state space of Go is too large to be searched directly, a roving eye may help prune the search by acting as a filter that approves or disapproves of searching different positions [21]. In this manner, the search can be made significantly more efficient, and the network does not have to attempt to look ahead to the end of the game. Such hybrid techniques constitute a most promising direction of future research in the long run.

7 Conclusion

The roving eye architecture is an appealing approach to Go because it is the same for any board size. It is also powerful because it can turn to face different directions, allowing it to process symmetrical configurations with the same connections. When compared with evolving from scratch, a 7×7 eye pre-evolved in a 5×5 board achieved significantly faster learning, and significantly higher final fitness. This result establishes that (1) The roving eye can indeed play

on different board sizes, and (2) the roving eye aids incremental evolution on increasingly large boards. Thus, the roving eye is a potentially important component of learning systems that aim to perform well on larger boards even when learning directly on such large boards is prohibitively complex.

Acknowledgments. This research was supported in part by the National Science Foundation under grant IIS-0083776 and by the Texas Higher Education Coordinating Board under grant ARP-003658-476-2001. Specials thanks to Darren Izzard for sharing his NEAT-based shape-recognizing roving eye.

References

1. Bouzy, B., Cazenave, T.: Computer Go : An AI oriented survey. Artificial Intelligence Journal **132** (2001) 39–193
2. Enzenberger, M.: Evaluation in go by a neural network using soft segmentation. In: Proceedings of the 10th Advances in Computer Games conference. (2003) 97–108
3. Lubberts, A., Miikkulainen, R.: Co-evolving a go-playing neural network. In: Coevolution: Turning Adaptive Algorithms Upon Themselves, Birds-of-a-Feather Workshop, Genetic and Evolutionary Computation Conf. (GECCO-2001). (2001)
4. Richards, N., Moriarty, D., Miikkulainen, R.: Evolving neural networks to play Go. Applied Intelligence **8** (1998) 85–96
5. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary Computation **10** (2002) 99–127
6. Cazenave, T.: Generation of patterns with external conditions for the game of Go. Advance in Computer Games **9** (2000) 275–293
7. Bayer, A., Bump, D., Denholm, D., Dumonteil, J., Farneback, G., Traber, T., Urvoy, T., Wallin, I.: GNU Go 3.2. Free Software Foundation, Cambridge, MA (2002)
8. Hershberger, D., Burridge, R., Kortenkamp, D., Simmons, R.: Distributed visual servoing with a roving eye. In: Proceedings of the Conference on Intelligent Robots and Systems (IROS). (2000)
9. Dickinson, S.J., Christensen, H.I., Tsotsos, J.K., Olofsson, G.: Active object recognition integrating attention. Computer Vision and Image Understanding **67** (1997) 239–260
10. Pierce, D., Kuipers, B.: Map learning with uninterpreted sensors and effectors. Artificial Intelligence **92** (1997) 169–227
11. Fullmer, B., Miikkulainen, R.: Using marker-based genetic encoding of neural networks to evolve finite-state behaviour. In Varela, F.J., Bourgine, P., eds.: Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life. MIT Press, Cambridge, MA (1992) 255–262
12. Nolfi, S.: Evolving non-trivial behavior on autonomous robots: Adaptation is more powerful than decomposition and integration. In Gomi, T., ed.: Evolutionary Robotics: From Intelligent Robots to Artificial Life. AAI Books, Ontario, Canada (1997)
13. Stanley, K.O., Miikkulainen, R.: Efficient reinforcement learning through evolving neural network topologies. In: Proceedings of the Genetic and Evolutionary Computation Conf. (GECCO-2002), San Francisco, CA, Morgan Kaufmann (2002)

14. Stanley, K.O., Miikkulainen, R.: Competitive coevolution through evolutionary complexification. Journal of Artificial Intelligence Research **21** (2004) In press.
15. Radcliffe, N.J.: Genetic set recombination and its application to neural network topology optimization. Neural computing and applications **1** (1993) 67–90
16. Goldberg, D.E., Richardson, J.: Genetic algorithms with sharing for multimodal function optimization. In Grefenstette, J.J., ed.: Proceedings of the 2nd Intl. Conf. on Genetic Algorithms, San Francisco, CA: Morgan Kaufmann (1987) 148–154
17. Gruau, F., Whitley, D., Pyeatt, L.: A comparison between cellular encoding and direct encoding for genetic neural networks. In Koza, J.R., Goldberg, D.E., Fogel, D.B., Riolo, R.L., eds.: Genetic Programming 1996: Proceedings of the First Annual Conference, Cambridge, MA, MIT Press (1996) 81–89
18. Yao, X.: Evolving artificial neural networks. Proceedings of the IEEE **87** (1999) 1423–1447
19. Rosin, C.D., Belew, R.K.: New methods for competitive evolution. Evolutionary Computation **5** (1997)
20. Ficici, S.G., Pollack, J.B.: Pareto optimality in coevolutionary learning. In Kelemen, J., ed.: Sixth European Conference on Artificial Life, Berlin; New York: Springer-Verlag (2001)
21. Moriarty, D.E., Miikkulainen, R.: Evolving neural networks to focus minimax search. In: Proceedings of the 12th National Conference on Artificial Intelligence, San Francisco, CA: Morgan Kaufmann (1994) 1371–1377

Comparing Discrete and Continuous Genotypes on the Constrained Portfolio Selection Problem

Felix Streichert, Holger Ulmer, and Andreas Zell

Centre for Bioinformatics Tübingen (ZBIT), University of Tübingen,
Sand 1, 72076 Tübingen, Germany,
{streiche, ulmerh, zell}@informatik.uni-tuebingen.de
http://www-ra.informatik.uni-tuebingen.de/

Abstract. In financial engineering the problem of portfolio selection has drawn much attention in the last decades. But still unsolved problems remain, while on the one hand the type of model to use is still debated, even the most common models cannot be solved efficiently, if real world constraints are added. This is not only because the portfolio selection problem is multi-objective, but also because constraints may turn a formerly continuous problem into a discrete one. Therefore, we suggest to use a Multi-Objective Evolutionary Algorithm and compare discrete and continuous representations. To meet constraints we apply a repair mechanism and examine the impact of Lamarckism and the Baldwin Effect on several instances of the portfolio selection problem.

1 Introduction

One prominent problem in financial engineering is portfolio selection, i.e. the problem how to invest money most profitable in multiple assets available. In this paper we investigate the application of a Multi-Objective Evolutionary Algorithm (MOEA), a heuristic that is virtually independent of the underlying portfolio selection model used. We investigate the impact of several coding schemes and the application of a repair mechanism together with Lamarckism on the constrained portfolio optimization problem.

First, we give a short introduction to the portfolio selection problem in sec. 1.1 and the related work in sec. 1.2. Then we explain details of the MOEA, the repair mechanism and the different coding schemes we applied in sec. 2. Results on several problem instances are shown in sec. 3 and finally conclusions and an outlook on future work are given in sec. 4 and sec. 5, respectively.

1.1 The Portfolio Selection Problem

The Markowitz mean-variance model [11,12] gives a multi-objective optimization problem, with two output dimensions. A portfolio p consisting of N assets with specific volumes for each asset given by weights w_i is to be found, which:

$$\text{minimizes the variance of the portfolio} : \sigma_p = \sum_{i=1}^{N} \sum_{j=1}^{N} w_i \cdot w_j \cdot \sigma_{ij}, \quad (1)$$

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1239–1250, 2004.

$$\text{maximizes the return of the portfolio :} \quad \mu_p = \sum_{i=1}^{N} w_i \cdot \mu_i, \tag{2}$$

$$\text{subject to :} \quad \sum_{i=1}^{N} w_i = 1 \quad \text{and} \tag{3}$$

$$0 \leq w_i \leq 1 \tag{4}$$

where $i = 1, .., N$ is the index of the asset, N represents the number of assets available, μ_i the estimated return of asset i and σ_{ij} the estimated covariance between two assets. Usually, μ_i and σ_{ij} are to be estimated from historic data.

While the optimization problem given in equ. 1 and equ. 2 is a quadratic optimization problem for which computationally effective algorithms exist, this is not the case if real world constraints are added:

Cardinality constraints restrict the maximal number of assets used in the portfolio, $\sum_{i=1}^{N} \text{sign}(w_i) = K$.

Buy-in thresholds give the minimum amount that is to be purchased, i.e. $w_i \geq l_i \quad \forall \quad w_i > 0;\ i = 1, .., N$.

Roundlots give the smallest volumes c_i that can be purchased for each asset, $w_i = y_i \cdot c_i; \quad i = 1, .., N$ and $y_i \in \mathbb{Z}$.

These constraints are often hard constraints, i.e. they must not be violated. Other real world constraints like sector/industry constraints, immunization/duration matching and taxation constraints can be considered as soft constraints and should be implemented as additional objectives, since this yields the most information. While we do consider the above hard constraints, we currently do not include soft constraints in our experiments, but plan to examine their impact in our future work.

1.2 Related Work

One of the first groups to apply Genetic Algorithms (GA) on the portfolio selection problem were Tettamanzi et al. [1,10,9]. They transformed the multi-objective optimization problem (MOOP) into a single-objective problem by using a trade-off function. They used multiple GA populations with individual trade-off coefficients to identify the complete Pareto front. More recently, Crama et al. applied Simulated Annealing (SA) to the portfolio selection problem [5]. They especially pointed out that SA and similar heuristics like GA have the major advantage that they can be easily applied to any kind of portfolio selection model with arbitrary constraints without much modification. For the same reason Beasley et al. compared Tabu Search, SA and GA on the portfolio selection to evaluate their performance [4]. They solved the MOOP by interpreting one objective as constraint and optimizing the other one. The constraint was altered iteratively to get the complete Pareto front. As a conclusion they found that no individual heuristic performed better than the other ones and that only a pooled result of all three heuristics produced a satisfying Pareto front.

Unfortunately, the papers using Evolutionary Algorithms (EA) did not apply multi-objective EAs (MOEA) to the portfolio selection problem, although MOEA have shown to be very useful on similar multi-objective optimization problems [8,6,16].

2 Multi-objective Evolutionary Algorithm

Our MOEA strategy uses a generational GA population strategy with a population size of 500 individuals. We apply tournament selection with a tournament group size of 8 together with objective space based fitness sharing with a sharing distance of $\sigma_{share} = 0.01$ [7]. The selection prefers individuals that are better than other individuals in at least one objective value, i.e. which are not dominated by other individuals. To maintain the currently known Pareto front we use an archive of 250 individuals and use this archive as elite to achieve a faster speed of convergence. Details of this MOEA strategy can be found in [13]. We use one-point mutation with a mutation probability of $p_m = 0.1$ and a discrete 3-point-crossover with $p_c = 1.0$ on all genotypes. For binary genotypes bit-flip mutation is used and in case of the real-valued genotype a gaussian random number with $\sigma = 0.05$ is added to a random decision variable. These parameters for the operators were selected to allow a fair comparison. The general parameters were found in preliminary experiments [14].

As representations we decided to compare bit-string based genotypes using binary or gray-coding to a real-valued genotype. On discretized problem instances, caused by additional roundlot constraints, we also investigated the size of the bit-string from a 32bit 'continuous' and a 7bit 'discrete' representation.

Preliminary experiments indicated that pareto-optimal solutions for the portfolio selection problem are rarely composed of all available assets, but only a limited selection of the available assets, especially in case of cardinality constraints, see Fig. 2. This selection problem resembles a one-dimensional binary knapsack problem, which has already been addressed by means of EA using a binary representation. Therefore, we suggest to use the very same representation in addition to the vector of decision variables **W**, see Fig. 1. Each bit of the bit-string **B** determines whether the associated asset is an element of the portfolio or not, so that the actual value of the decision variable is $w'_i = b_i \cdot w_i$. This is the value that is processed by the following repair algorithm. With this hybrid representation it is much easier for the GA to add or remove the associated assets simply by mutating the additional bit-string. The hybrid representation is altered by mu-

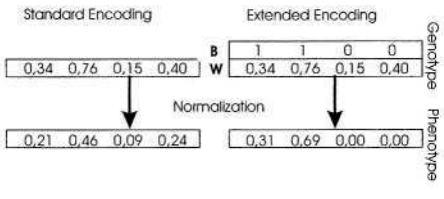


Fig. 1. Comparing the standard representation to the hybrid representation.

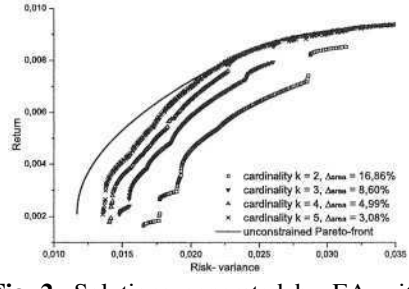


Fig. 2. Solutions generated by EA with the hybrid representation on the *DAX* data set with 81 assets as given in [2].

tating/crossing each genotype element (**B** and **W**) separately from each other. The extended GA is abbreviated KGA (Knapsack-GA).

The GA implementation used encodes each decision variable in the desired range, $w_i \in \{0,1\}$, but especially the additional constraints given in sec. 1.1 are rather restrictive. Therefore, it is impractical to outright reject all infeasible solutions. This is the reason why we applied a repair algorithm, which searches for the next feasible solution.

To do so the repair algorithm first removes all surplus assets from the portfolio to meet the cardinality constraints by setting the $N - K$ smallest values of w_i to zero and also those assets whose weights are below the given buy-in threshold. For those $w_i > 0$ remaining, the weights are normalized such that $w_i' = l_i + \frac{w_i - l_i}{\sum(w_i - l_i)}$. To meet round-lot constraints the algorithm rounds the $w_i > 0$ to the next round-lot level, $w_i'' = w_i' - (w_i' \mod c_i)$, after cardinality repair, buy-in repair and normalization was applied. The remainder of the rounding process, $\sum_i (w_i' \mod c_i)$, is spent in quantities of c_i on those w_i'', which had the biggest values for $w_i' \mod c_i$ until all of the remainder is spent. Since the repair algorithm is deterministic, an individual is always assigned to the same phenotype after repair if the genotype did not change.

Since in a basic implementation the repair mechanism would only determine the phenotype of a GA individual, we compare the performance of the GA with and without Lamarckism to further examine the effect of the repair mechanism. With Lamarckism alters genotype of a GA individual is altered by coding the phenotype back onto the genotype.

3 Experimental Results

The comparison of the different GA implementations is performed on a public benchmark data set provided by Beasley [2]. The numerical results presented here were performed on the *Hang Seng* data set with 31 assets. On this data set we use several combinations of real world constraints to compare the performance of the different GA representations. First, we compare the cardinality constrained portfolio selection problem without and with use of Lamarckism. In a second set of experiments we also add real-world constraints like buy-in thresholds and roundlot constraints.

To compare the performance of the MOEAs we use the S-metric that calculates the hyper volume under the Pareto front [17]. We take the percentage difference (Δ_{area}) between the hyper volume of the Pareto front found by the MOEA and a reference solution of the unconstrained portfolio selection problem, compare Fig. 2, Δ_{area} is to be minimized.

To obtain reliable results we repeat each GA experiment for 50 times for each parameter setting and problem instance. A single GA run is terminated after 100,000 fitness evaluations. We then calculate the mean value, the standard deviation, the maximum and minimum values and the 90 % confidence intervals of the Δ_{area} value to evaluate the performance of each GA setting.

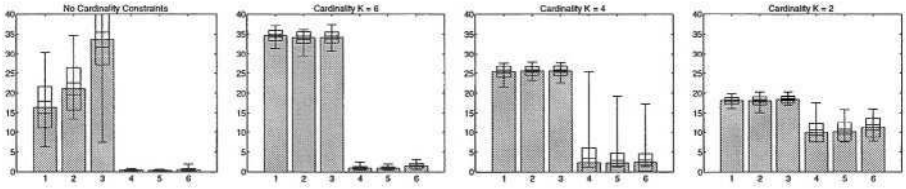


Fig. 3. Δ_{area} for the experiments on the Hang Seng data set $l_i = 0$ and $c_i = 0$ (1: GA binary-coding, 2: GA gray-coding, 3: GA real-valued, 4: KGA binary-coding, 5: KGA gray-coding, 6: KGA real-valued)

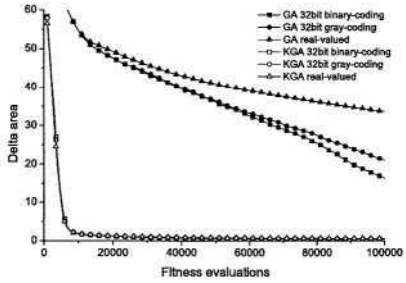


Fig. 4. Δ_{area} on the Hang Seng data set with $K = N$, $l_i = 0$ and $c_i = 0$

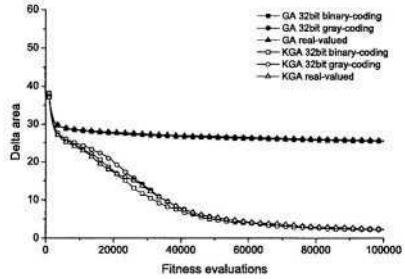


Fig. 5. Δ_{area} on the Hang Seng data set with $K = 4$, $l_i = 0$ and $c_i = 0$

3.1 Results without Additional Constraints

In our experiments we distinguish further between experiments with and without Lamarckism. On the one hand Lamarckism is said to cause premature convergence, while the Baldwin effect on the other hand leads to a neutral search space, which may enable the GA to escape local optima, see [15] for further details. We show that the applied repair mechanism has a quite unexpected result on the constrained portfolio selection problem if Lamarckism is not applied.

Without Lamarckism. On the simplest problem instance without additional constraints the behavior of the hybrid KGA representation clearly outperforms the standard representation on all problem instances, see Figs. 3-5. Without cardinality constraints the hybrid KGA nearly instantly converges to very good values of Δ_{area} independent of the coding scheme used for the genotype. Only in case of $K = 2$ the real-valued KGA performs slightly worse than the bit-string based KGAs.

When the standard GA is used on the portfolio selection problem without cardinality constraints, the different genotype coding schemes can be clearly distinguished, see Fig. 4. Here the real-coded GA performs worst, while the binary-coding is better than the gray-coding. But when cardinality constraints are used no such distinctions can be made anymore. This is due to the combined effect of cardinality constraints and the applied repair mechanism. The repair algorithm

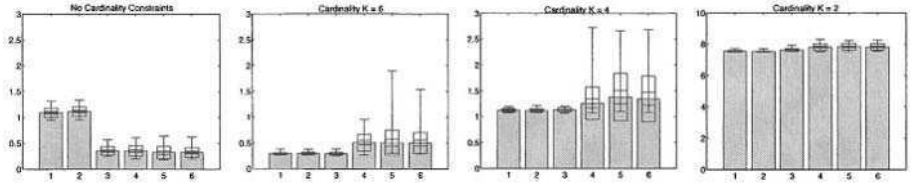


Fig. 6. Δ_{area} for the experiments on the Hang Seng data set with Lamarckism, $l_i = 0$ and $c_i = 0$ (1: GA binary-coding, 2: GA gray-coding, 3: GA real-valued, 4: KGA binary-coding, 5: KGA gray-coding, 6: KGA real-valued)

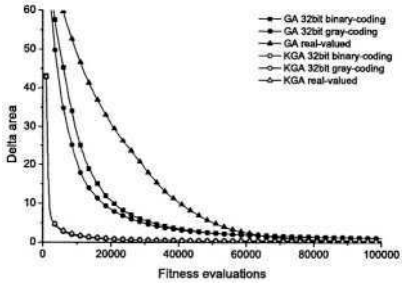


Fig. 7. Δ_{area} on the Hang Seng data with Lamarckism, $K = N$, $l_i = 0$ and $c_i = 0$

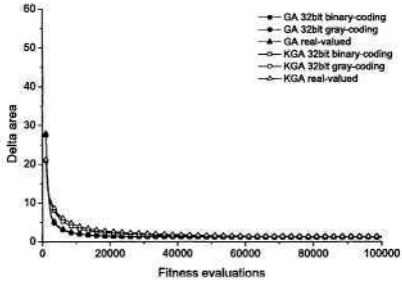


Fig. 8. Δ_{area} on the Hang Seng data with Lamarckism, $K = 4$, $l_i = 0$ and $c_i = 0$

always selects the K biggest w_i to be part of the portfolio. The remaining w_i are normalized to values of $w_i' \approx 1/K$. The other $N - K$ asset weights are subject to genetic drift, since there is no selection pressure toward sparse vectors $\mathbf{W}$. If any of the previously selected w_i drops out of the portfolio due to mutation or crossover, the biggest of the $N - K$ asset weights takes its place and the values are again normalized to $w_i' \approx 1/K$. This way the standard GA only searches the subspace of portfolio of size K with weights $w_i \approx 1/K$.

With Lamarckism. With cardinality constraints and Lamarckism the standard GA inherits some properties of the hybrid KGA. Since the repair mechanism removes the surplus assets from the portfolio and Lamarckism removes them from the genotype, the standard GA also acts on a sparse vector of $\mathbf{W}$ like the hybrid KGA. This way the standard GA can add and remove assets to and from the portfolio as easily as the hybrid KGA. Now the standard GA is also able to explore the complete subspace of possible portfolio combinations, see Fig. 6. The standard GA even outperforms the KGA reading speed convergence and reliability of the results for $K < N$.

Without cardinality constraints this effect is not as strong, although the results of the standard GA are much better and the speed of convergence is increased notably, see Fig. 7. Here Lamarckism removes neutrality from the search space, which enables the standard GA to remove surplus assets more efficiently and thereby the standard GA converges much faster, compare Fig. 7 to Fig. 4.

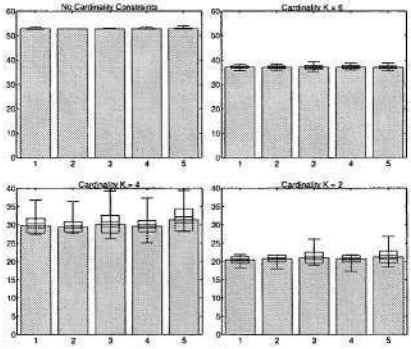


Fig. 9. Δ_{area} for the experiments on the Hang Seng data set with $l_i = 0.08$ and $c_i = 0.008$ (1: GA 32bit binary-coding, 2: GA 7bit binary-coding, 3: GA 32bit gray-coding, 4: GA 7bit gray-coding, 5: GA real-valued)

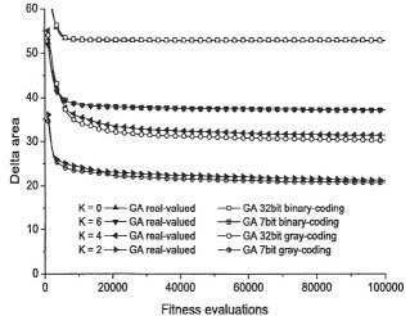


Fig. 10. Δ_{area} on the Hang Seng data set with several cardinality constraints, $l_i = 0.08$ and $c_i = 0.008$

But also the performance of the KGA is increased due to Lamarckism. Although no better results are found the KGA converges significantly, faster especially with increasing cardinality constraints, compare Fig. 8 to Fig. 5.

Unfortunately, all the GA representations perform so well on this problem instance with $K < N$ that no clear distinctions can be made. Only for $K = N$ the real-valued GA converges slower than the bit-string based GA, see Fig. 7, but outperforms both bit-string based standard GAs regarding the quality of the results, see Fig. 6. But these differences also vanish with the application of the hybrid KGA.

3.2 Results with Additional Constraints

With additional real-world constraints the previously continuous portfolio selection problem becomes a discrete one. Therefore, we extend the group of examined representations with an additional discrete representation using a bit-string limited to 7bit instead of 32bit.

To increase comprehensibility we examine the results separately, first for the standard GA and then for the hybrid KGA representation.

Standard GA without Lamarckism. Here the very same effect as in sec. 3.1 occurs: the standard GA implementation suffers from premature convergence, see Fig. 9 and Fig. 10. Again this is due to the neutrality of the search space caused by the repair mechanism. But now it applies to all problem instances since even without cardinality constraints the additional buy-in threshold acts like a cardinality constraint of $K = 12$. The neutral search space causes the GA

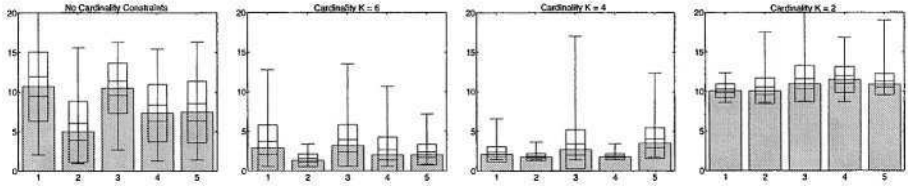


Fig. 11. Δ_{area} for the experiments on the Hang Seng data set with Lamarckism, $l_i = 0.08$ and $c_i = 0.008$ (1: GA 32bit binary-coding, 2: GA 7bit binary-coding, 3: GA 32bit gray-coding, 4: GA 7bit gray-coding, 5: GA real-valued)

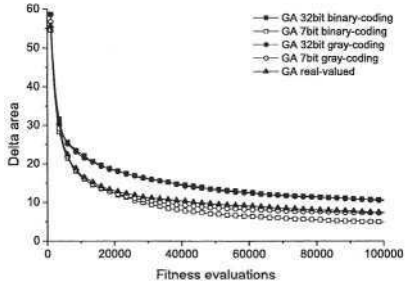


Fig. 12. Δ_{area} on the Hang Seng data set with Lamarckism, $K = N$, $l_i = 0.08$ and $c_i = 0.008$

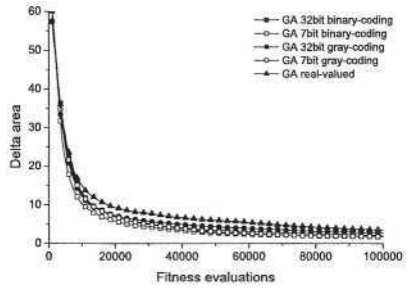


Fig. 13. Δ_{area} on the Hang Seng data set with Lamarckism, $K = 4$, $l_i = 0.08$ and $c_i = 0.008$

to search a subspace of the true search space and again the subspace consists only of portfolios of size K with weights $w_i \approx 1/K$.

Fig. 10 shows the convergence behavior on each problem instance. On each problem instance a bit-string based representation is compared to the real-valued representation. Basically, they all converge to the very same local optimum in the previously described subspace, but the real-valued representation performs slightly worse than the bit-string based representations.

Standard GA with Lamarckism. Again with Lamarckism the negative effect of the neutral search space is removed, see Fig. 11. And again the standard GA becomes much more efficient, since it is able to search the space of sparse portfolios more efficiently. The convergence speed of the standard GA once more matches the behavior of the hybrid KGA in the previous examples, see Fig. 12 and Fig. 13.

Regarding the different coding schemes the real-valued coding performs slightly better than the 32bit codings. But comparing the 7bit coding to the 32bit coding, the 7bit coding performs much better than the 32bit coding and also outperforms the real-valued representation, see also Fig. 12 and Fig. 13. Most likely this is due to the reduced search space of the 7bit coding and the greater impact of the mutation operator. While the confidence intervals for the different representations are clearly separated for $K = N$ and $K = 4$, the differences decrease with increasing cardinality constraints.

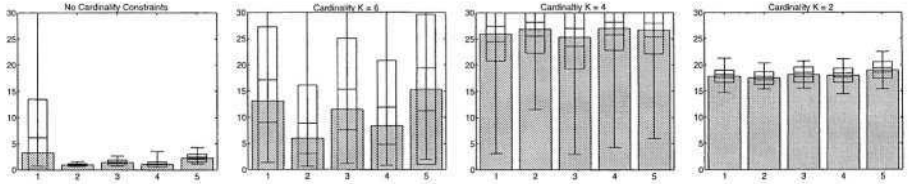


Fig. 14. Δ_{area} for the experiments on the Hang Seng data set with $l_i = 0.08$ and $c_i = 0.008$ (1: KGA 32bit binary-coding, 2: KGA 7bit binary-coding, 3: KGA 32bit gray-coding, 4: KGA 7bit gray-coding, 5: KGA real-valued)

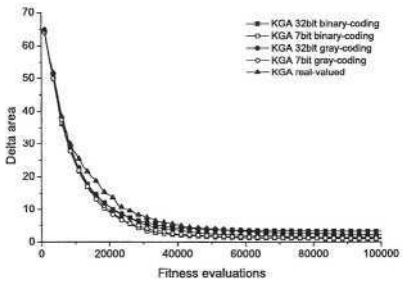


Fig. 15. Δ_{area} on the Hang Seng data set with $K = N$, $l_i = 0.08$ and $c_i = 0.008$

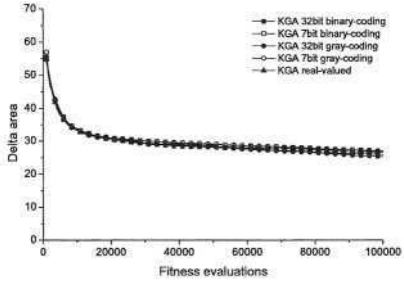


Fig. 16. Δ_{area} on the Hang Seng data set with $K = 4$, $l_i = 0.08$ and $c_i = 0.008$

Hybrid KGA without Lamarckism. Even without Lamarckism the hybrid KGA is not prone to the same premature convergence as the standard GA, compare Fig. 14 to Fig. 11. But while without cardinality constraints the hybrid KGA performs rather well, see Fig. 15, this is not the case with increasing cardinality constraints, see Fig. 16. Although the mean results of the hybrid KGA are still better than the results of the standard GA and the best runs of the hybrid KGA are considerably better, the overall results of the KGA without Lamarckism can be rejected as being too bad and also too unreliable.

When comparing the different coding schemes, again the real-valued KGA performs worst on all problem instances, see Fig. 14. The 32bit codings usually perform slightly better than the real-valued representation, except for some extreme outliers in case of the 32bit binary-coding for $K = N$. But the confidence intervals between the 32bit codings and the real-valued coding are not as clearly separated. But the confidence intervals for 32bit and 7bit coding are clearly separated at least for weak cardinality constraints, $K = N$ and $K = 6$, and show that the 7bit coding outperforms the 32bit coding. With increasing cardinality constraints these differences are again leveled out. Regarding the comparison between binary and gray-coding no reliable conclusions can be made, since the confidence intervals have a significant overlap.

Hybrid KGA with Lamarckism. The application of Lamarckism gives the driving edge to the hybrid KGA, see Fig. 17. In some instances the results are so good, that we believe the fixed size of the archive population may become a limiting element.

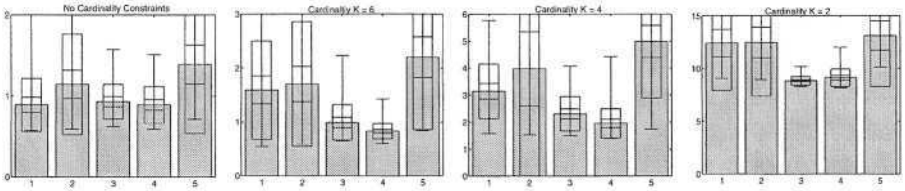


Fig. 17. Δ_{area} for the experiments on the Hang Seng data set with Lamarckism, $l_i = 0.08$ and $c_i = 0.008$ (1: KGA 32bit binary-coding, 2: KGA 7bit binary-coding, 3: KGA 32bit gray-coding, 4: KGA 7bit gray-coding, 5: KGA real-valued)

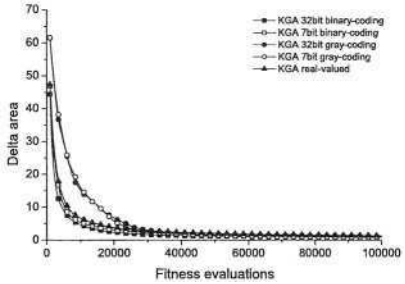


Fig. 18. Δ_{area} on the Hang Seng data set with Lamarckism, $K = N$, $l_i = 0.08$ and $c_i = 0.008$

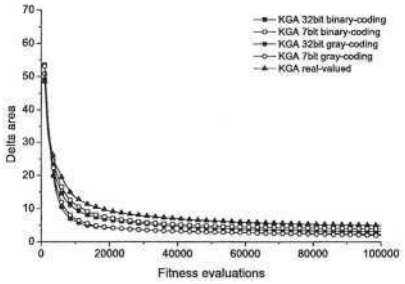


Fig. 19. Δ_{area} on the Hang Seng data set with Lamarckism, $K = 4$, $l_i = 0.08$ and $c_i = 0.008$

Comparing the different representations the real-valued representation performs again worst. Second best is the binary-coding, but astonishingly the previously observed advantage of the 7bit coding is reversed in this case. Gray-coding on the other hand performs best on all problem instances and again the confidence intervals indicate a significant advantage of the 7bit gray-coding over the 32bit gray-coding. In this case the general advantage of the gray-coding is even maintained for increasing cardinality constraints and actually becomes more and more obvious.

Regarding the speed of convergence the gray-coding is slightly slower in the beginning for $K = N$, see Fig. 18, but it catches up and finally produces the best results. With increasing cardinality constraints the gray-coding performs better and outperforms the other coding schemes regarding speed of convergence and the quality of the final result, see Fig. 19 and Fig. 17.

4 Discussion

There are several conclusions that can be drawn from the experimental results presented in this paper. First, we were able to prove that the proposed hybrid KGA representation performed better than the standard GA on this problem class regardless of the problem instance and the genotype representation used for **W**. The KGA produced better results and converged faster than the standard GA. We could support the argument, that the advantage of the hybrid

KGA is based on the efficient removal of surplus assets, by reproducing the very same effect for the standard GA on the problem instances without additional constraints, with cardinality constraints and Lamarckism. Although the positive effect of Lamarckism on the standard GA was not as strong if real-world constraints were added.

We also showed that the standard GA without Lamarckism is prone to premature convergence, since the neutrality of the search space causes the GA to get stuck in an suboptimal subspace. The KGA on the other hand was not prone to such premature convergence even without Lamarckism.

Regarding the different coding schemes we were able to show that on average the real-valued coding performed worst on all problem instances. But there were only negligible differences between the binary and the gray-coding if no additional constraints were applied. We could also prove that with additional constraints the 'discrete' 7bit coding performed better than the 32bit coding on both bit-string based codings, most likely because the mutation and crossover operators become more effective.

Overall, the hybrid KGA with 7bit gray-coding and Lamarckism turned out to be best on the most interesting problem instances with additional real-world constraints.

5 Future Work

Our future work will concentrate on evaluating the performance of alternative MOEA implementations on the portfolio selection problem. We believe that the choice of the MOEA strategy will become crucial, if more real-world constraints are added like sector/industry constraints, immunization/duration matching and taxation constraints, which may increase the output dimension of the portfolio selection problem.

Another area of improvement could be the application of more sophisticated local search heuristics. There are numerous alternatives to the simple search for feasible solutions, but they have to be carefully evaluated regarding their ability to handle real-world constraints.

Finally, we plan to extend our experiments to other models for portfolio selection like the Black-Litterman model [3], since the Markowitz mean-variance model suffers from two major drawbacks: first, it is rather complicated to gather the necessary data and estimate μ_i and σ_{ij} from historic data and secondly, the Markowitz model is very sensitive to estimation errors of μ_i and σ_{ij}.

References

1. S. Arnone, A. Loraschi, and A. Tettamanzi. A genetic approach to portfolio selection. *Neural Network World, International Journal on Neural and Mass-Parallel Computing and Information Systems,* 3:597–604, 1993.
2. J. B. Beasley. OR-Library: distributing test problems by electronic mail. *Journal of the Operational Research,* 8:429–433, 1996.

3. F. Black and R. Litterman. Global portfolio optimization. *Financial Analysts Journal,* pages 28–43, September-October 1992.
4. T.-J. Chang, N. Meade, J. B. Beasley, and Y. Sharaiha. Heuristics for cardinality constrained portfolio optimization. *Computers and Operations Research,* 27:1271–1302, 2000.
5. Y. Crama and M. Schyns. Simulated annealing for complex portfolio selection problems. Working paper GEMME 9911, Université de Liège, 1999.
6. K. Deb, S. Agrawal, A. Pratab, and T. Meyarivan. A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, editors, *Proceedings of the Parallel Problem Solving from Nature VI Conference,* pages 849–858, Paris, France, 2000. Springer. Lecture Notes in Computer Science No. 1917.
7. D. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In Grefenstette, editor, *Proceedings of the 2nd International Conference on Genetic Algorithms,* pages 41–49, 1987.
8. J. Knowles and D. Corne. The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation. In P. J. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, and A. Zalzala, editors, *Proceedings of the Congress on Evolutionary Computation,* volume 1, pages 98–105, Mayflower Hotel, Washington D.C., USA, 1999. IEEE Press.
9. A. Loraschi and A. Tettamanzi. An evolutionary algorithm for portfolio selection in a downside risk framework. *Working Papers in Financial Economics,* 6:8–12, June 1995.
10. A. Loraschi, A. Tettamanzi, M. Tomassini, and P. Verda. Distributed genetic algorithms with an application to portfolio selection problems. In D. W. Pearson, N. C. Steele, and R. F. Albrecht, editors, *Artificial Neural Networks and Genetic Algorithms,* pages 384–387, Wien, 1995. Sringer.
11. H. M. Markowitz. Portfolio selection. *Journal of Finance,* 1(7):77–91, 1952.
12. H. M. Markowitz. *Portfolio Selection: efficient diversification of investments.* John Wiley & Sons, 1959.
13. N. Srinivas and K. Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation,* 2(3):221–248, 1994.
14. F. Streichert, H. Ulmer, and A. Zell. Evolutionary algorithms and the cardinality constrained portfolio selection problem. In D. Ahr, R. Fahrion, M. Oswald, and G. Reinelt, editors, *Operations Research Proceedings 2003, Selected Papers of the International Conference on Operations Research (OR 2003), Heidelberg, September 3-5, 2003.* Springer, 2003.
15. D. L. Whitley, V. S. Gordon, and K. E. Mathias. Lamarckian evolution, the baldwin effect and function optimization. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Parallel Problem Solving from Nature – PPSN III,* pages 6–15, Berlin, 1994. Springer.
16. E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Technical Report 103, Gloriastrasse 35, CH-8092 Zurich, Switzerland, 2001.
17. E. Zitzler and L. Thiele. Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE Transactions on Evolutionary Computation,* 3(4):257–271, 1999.

Learning Environment for Life Time Value Calculation of Customers in Insurance Domain

Andrea Tettamanzi[1], Luca Sammartino[2], Mikhail Simonov[2], Massimo Soroldoni[2], and Mauro Beretta[3]

[1] Università degli Studi di Milano, Dipartimento di Tecnologie dell'Informazione
Via Bramante 65, I-26013 Crema (CR), Italy, andrea.tettamanzi@unimi.it
[2]Nomos Sistema S.p.A., Viale Monza 259, I-20126 Milano (MI), Italy,
{simonov, soroldoni, sammartino}@nomos.it
[3]Genetica S.r.l., Via San Dionigi 15, I-20139 Milano (MI); Italy
beretta@genetica-soft.com

Abstract. A critical success factor in Insurance business is its ability to use information sources and contained knowledge in the most effective way. Its profitability is obtained through the Technical management plus Financial management of the funds gathered on the market. The profitability of a given customer can be evaluated through its Life Time Value (LTV). We aim at applying evolutionary algorithms to the problem of forecasting the future LTV in the Insurance Business. The Framework developed within the Eureka co-funded research projects HPPC/SEA and IKF has been adapted to the Insurance Domain through a dedicated Genetic Engine. The solution uses RDF and XML-compliant standard. The idea of using evolutionary algorithms to design fuzzy systems date from the beginning of the Nineties and a fair body of work has been carried out throughout the past decade. The approach we followed uses an evolutionary algorithm to evolve fuzzy classifiers of the data set.

1 Introduction

A critical success factor for an Insurance business today is its ability to use information sources and contained knowledge in the most effective way. This strategic use of data can result from opportunities presented by discovering hidden, previously undetected, and frequently extremely valuable facts about consumers, retailers and suppliers, business trends, and direction and significant factors. Knowing this information, an organisation can formulate effective business, marketing, and sales strategies; precisely target promotional activity; discover and penetrate new markets; and successfully compete in the marketplace from a position of informed strength.

Ontologies can play an important role in industrial systems by enabling access to legacy resources in a transparent and distributed way. Since ontologies are developed in order to provide a machine-processable semantics of information that is exchanged between different agents, either humans or software, the same mechanism can be used

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1251–1262, 2004.

for Legacy Service discovery and for automated reasoning about the content of the answers obtained from such a system. However, the only possibility to offer the self-learning environment and DSS system is the use of data mining. The technology aimed at achieving this strategic advantage is known as 'advanced data mining'.
The Framework developed within the Eureka co-funded research projects HPPC/SEA and IKF has been adapted to the Insurance Domain through a dedicated Genetic Engine. The solution uses an RDF- and XML-compliant standard according to W3C recommendations for data representation.
One interesting application of data mining is customer modeling, whose practical uses in Finance range from proactive marketing to fraud detection. We will focus on a particular real-world application of the above methods within the Insurance Domain, namely the calculation of the Life Time Value of customers.
This document is organised as follows: Section 2 introduces the problem and its formulation; Section 3 provides a sketch of the system; Sections 4 and 5 present a descriptions of adopted business and fuzzy modeling; Section 6 illustrates an optimization engine based on evolutionary algorithm for the problem; Section 7 provides a sample application of the proposed approach and the results obtained; finally, Section 8 outlines at a glance the main achievements of the project.

2 Calculating the Customer Lifetime Value

Profitability in the Insurance Business is obtained through the Technical management plus Financial management of the funds gathered on the market. Since interest rates have dropped dramatically in the recent past, current financial management cannot anymore guarantee the profitability of the whole business by itself. As a consequence, the only way to run such a business properly is to increase the profitability of the Technical Management of insurance policies. The Insurance Business is customer-dependent and it is obvious that there are "good" and "bad" customers in terms of profitability. The typical portfolio of an insurance company contains both of them. The profitability of a given customer can be evaluated through its Life Time Value (LTV). While the past LTV is known and the present LTV can be calculated on the fly, the expected future LTV can only be forecasted. We illustrate an application of evolutionary algorithms to the problem of obtaining an accurate and reliable forecast of the future customer LTV.
The insurance business still relies heavily on legacy applications, where several millions of customers plus several insurance policies for each of them are managed. The huge amount of data makes it impossible to manage an individual customer in a personalized fashion. This is a typical business problem in the insurance domain which can be successfully solved by applying advanced data mining techniques.

3 Overview of the System

All back-office management activities in the Insurance domain are performed by huge and well-tested legacy systems. Through an intranet, an insurance agent deals with a legacy system in order to obtain needed information or to answer typical queries from

customers that visit the agency or call it over the phone. The agent deals with the back office when more complex questions are raised or real-time calculations are required. This process is time-consuming and automation through semantic capabilities and data mining software would be helpful. Unfortunately, software with such a mix of automated reasoning and advanced data mining with self learning capabilities is not available in commercial systems.

On the other hand, users can deal with web-based applications during the purchase of an insurance product or when the "automated" customer relationship management (CRM) is contacted for frequently asked questions. Managing such relations is not straightforward, because it requires accessing a legacy system, understanding user queries, knowledge of insurance regulations and guidelines with user data or even profiles, etc. By providing such capabilities, also known as customer services, the insurer invests money and expects to increase the profitability of the main business. Accordingly, it is very important to be able to predict the Life Time Value of a customer in order to ensure the best possible customer service with respect to the expected profit.

Industrial applications must process some specific calculations in order to come up with a customer model and apply it to specific customer data. In order to allow these calculations, some back-office activities must be performed and especially the model must be created, updated, and adapted to new realities emerging from a continuously changing customer portfolio.

The system is implemented as a client-server solution, made up of a Genetic Engine and a Client controlling it. The communication mechanism is based on raw sockets in order to be fully portable between different platforms (Windows, Unix, Linux etc.), and can be deployed on parallel machines (MPI libraries are used). The enterprise solution is implemented in C++ and the communication between client and server is done in XML according to W3C recommendations. A wrapping layer is implemented in order to allow the call of the whole system from the Java environment, such as a distributed web application. The IKF vertical applications[1] use this interface to perform knowledge extraction from structured data sources. A specific HTML (JSP) application was also implemented to allow the front office to call the application to evaluate a model for a given customer and obtain a prediction of the future LTV.

4 Business Modelling

In the Insurance Domain the central point of any modelling is the customer. Accordingly, the Customer is modelled as the owner of relationships with the Insurer and other people. Furthermore, the relation between the Customer and his/her Properties is modelled, since Risks are generated by Properties. Risks can be the object of the Insurance Contract and the latter can generate profits or losses.

The specific Property named Life can be also modelled in order to expand the system towards Life Insurance contracts and investments in general (we observe that investments do not imply insurance-relevant life risks since the only risk in investment is financial).

[1] see www.ikfproject.com for further details.

People become customers because they own some properties. Properties generate risks and people wish to be sure that losses can be repaid in order to have the possibility to reconstruct their patrimony, i.e. people have Markov's needs. Markov's needs are considered goods to be sold on the Insurance Market. But the owners of Markov needs are not all equal: there are people who carry more risks and people who carry fewer risks, people who will generate profits, no profit, or losses. Moreover, there are people who generate losses in the short period but generate a high profit over the life time relation. Having said this, it is very important to be able to calculate the Life Time Value of a customer (LTV) for the past and present, and to be able to predict the LTV for the short- and medium-term future, as well as for the entire residual lifetime.

The application offers (Fig. 1):

- an automated model development as a support for a back office activity
- a concrete system for customer classification in order to define segments with customers having high expected LTV, medium LTV and low LTV able to generate in future high profit, medium profit and low profit respectively
- a "teller machine" able to apply the developed model to users data, useable even in a distributed web environment
- an advanced data mining genetic engine able to perform data discovery relevant for a given business
- XML/RDF compliant results generation accordingly W3C recommendations able to deliver not only the results but the semantic meaning of results

Setting intuition to the minimum, an Insurance business could be modelled according to the following schema[2]:

- Properties –implies– Risks (1..*)
- Life (is-a Property) –implies– Risks (1..*)
- Customer –has– Properties (1..*)
- Customer –has– Policy (1..*)
- Policy –has– Claims (0..*)
- Policy –has– Receipts (1..*)
- Claim –has– Payments (1..*)
- Customer - has - Relationship with the Insurer (1..1)

The schema means that a customer has at least one policy, for which at least one receipt is expected (incoming cash flow for Insurer), and at least zero claims can be expected. A claim expects at least one payment (cash outflow). Following such schema, user data are represented in a physical legacy database (DB2 is widely adopted in the use case).

Our application relies on the legacy database, but it also provides other ways to access insurance-related information, and combines legacy data with that information by integrating IES domain ontology with the legacy db.

The domain modelling was performed as follows:

Object	DOLCE-Lite + Type	Informal description	Informal constraints

[2] The above mentioned modelling comes from the IKF-IES application developed within Eureka co-funded project named IKF, see **http://www.ikfproject.com**. IKF develops frameworks to allow knowledge management in various business settings.

Policy	Non-physical object	A policy is issued by an insurer for an insured and covers certain losses given certain risks	Held_by Insured Issued_by Insurer Covers Insured
Insurer	Non-physical object	An insurer issues a policy to an insured which covers losses given certain risks	Issues policy Pays or disclaims claims
Insured	Non-physical object	An insured is covered by a policy against certain losses given certain risks	Covered by a policy
Claim	Activity	An insured makes a claim against a policy for a loss	Made by insured Paid or disclaimed by Insurer
Risk	Non-physical object	Risk is the likelihood of a certain loss for a certain insured	Contemplated by a policy
Loss	Non-physical object	A loss exists when an insured suffers some injury to person or property. The loss is said to be covered if it was part of the risk contemplated by the policy	Suffered by insured Covered (or not) by a policy
Disclaimer	Activity	A disclaimer is made by an insurer when a loss is not covered by a policy, usually when not a contemplated risk	Made by insurer

Fig. 1. A table used for experts' drafting of the core ontology

In order to develop the model, a task force of domain experts, software engineers, and ontology specialists has reached a design agreement. The model was developed and the genetic engine was specially engineered to match the domain.
The Management Framework is responsible for generating a model (back-office activity), evolving a model evolution to keep it updated (back-office activity), and evaluating a customer (front-office activity) during the policy subscription or call center contacts (even during self help use through web channels).

5 Fuzzy Rules Model

The task of generating predictive models of customer behaviour, or classifiers, can be (and has been) approached in many ways and using many techniques, including decision trees, neural networks, linear classifiers and non-linear statistical classifiers, among the most popular ones. All techniques have their advantages and drawbacks. However, one thing that is often required in practical applications is the interpretability, or explanatory power, of a model.
This is the main motivation for evolving classifiers made of decision rules. Moreover, because reality is not sharp and clean and in general decisions taken using crisp

thresholds are dangerous in a number of real-world applications, we have moved our attention to fuzzy rules, for their intrinsic interpolative behaviour.
The idea of using evolutionary algorithms to design fuzzy systems date from the beginning of the Nineties [5, 12] and a fair body of work has been carried out throughout the past decade [6, 7, 4]. The approach we followed is largely based on the development of previous work on the evolution of fuzzy controllers [11], which has been already validated on standard machine learning benchmarks [10].
Based on that work, we define a classifier as a rule base, of up to 256 rules, which should be more than enough to express even very complicated models, each rule comprising up to four antecedents and one consequent clause. Up to 256 input variables and one output variable can be handled, described by up to 16 distinct membership functions each. Membership functions for input variables are trapezoidal, while membership functions for the output variables are triangular.

6 Genetic Engine

Our approach uses an evolutionary algorithm to evolve fuzzy classifiers of the data set. Evolutionary algorithms are a broad class of optimisation methods inspired by Biology, which build on the key concept of Darwinian evolution. It is assumed that the reader is already familiar with the main concepts and issues relevant to evolutionary algorithms; good reference books are [1, 3, 8, 2].

6.1 The Algorithm

An island-based distributed evolutionary algorithm is used to evolve classifiers. An island-based algorithm maintains several populations (islands) which evolve separately exchanging individuals from time to time. The sequential algorithm executed on every island is a standard generational replacement, elitist evolutionary algorithm. Crossover and mutation are never applied to the best individual in the population.

Encoding
Classifiers are encoded in three main blocks: a set of membership functions for the input variables, a set of symmetric membership functions, represented by (area, center of mass) pairs for the output (or classification) variable, and a set of rules.
A single input variable membership function is defined by four fixed-point numbers, each fitting into a byte. Output variable membership functions are assumed to be symmetrical, and thus can be described using just two parameters: their area and center of mass. A rule is a list of up to four conjoint antecedent clauses (the IF part) and a consequent clause (the THEN part). A clause is represented by a pair of indices referring respectively to a variable and to one of its linguistic values, i.e., a membership function.

Initialization

The population can be seeded either with hand-written or otherwise already existing classifiers or with new random ones. A new random individual is created according to the following algorithm:

Each input variable has at least one linguistic value; the number of additional values is determined by sampling from a truncated exponential distribution with mean three;

The shapes of the membership functions for the input variables are determined by randomly extracting a center *C* with uniform probability over the variable definition interval, and a spread σ from an exponential distribution with mean *1/4* of the variable range. The four numbers defining the trapezoid, *a*, *b, c* and *d* are assigned as follows:

$$\begin{aligned} a &= C - 2/3\sigma, \\ b &= C - 1/3\sigma, \\ c &= C + 1/3\sigma, \\ d &= C + 2/3\sigma \end{aligned} \qquad (1)$$

The exponential distribution is used to determine the spread because it is the zero-information probability distribution for a non-negative random variable. At least two output membership functions have to be present for each output variable; the number of additional linguistic values for each output variable is determined by sampling from a truncated exponential distribution with mean three. The centers of mass for the output variable are randomly extracted in the [0, *M*-1] interval; the areas are extracted at random such that they correspond to a triangular membership function whose base is entirely contained in that range. At least *M* rules have to be present, using at least *M* different output linguistic values; the number of additional rules in the rule base is determined by sampling from a truncated exponential distribution with mean six. The rules are generated according to the following algorithm:

for each input variable, a fair coin is flipped to decide whether to include it in the antecedent part, not exceeding four variables;

for each selected input variable, a linguistic value is extracted among those defined;

an output variable and its linguistic value are extracted for the consequent part of the rule.

Recombination

The recombination operator is designed to preserve the syntactic legality of classifiers. A new classifier is obtained by combining the pieces of two parent classifiers. Each rule of the offspring classifier can be inherited from one of the parent programs with probability *1/2*. When inherited, a rule takes with it to the offspring classifier all the referred linguistic values with their membership functions. Other linguistic values can be inherited from the parents, even if they are not used in the rule set of the child classifier, to increase the size of the offspring so that their size is roughly the average of its parents' sizes. This sort of recessive genes might fall back into use because of mutations, and it has been empirically observed that their presence enhances the performance of the algorithm.

Mutation

Like recombination, mutation also produces only legal classifiers. Mutation can result in one or more of the following changes, with probability given by the mutation rate, p_m, identical and independent for each component of the genotype:

– a new linguistic value with a random membership function is added to an input variable;
– a linguistic value whose membership function is not used in the rules is removed from an input variable;
– a membership function is perturbed as follows: each of the four points defining the trapezoid can be moved, with probability p_m, to a new random position between the previous and next points;
– a new linguistic value, with random area and center of mass, is added to an output variable;
– a linguistic value whose area and center of mass are not used in the rules is removed from an output variable;
– an area-center of mass pair is perturbed as follows:
  1. a standard deviation σ for the perturbation is extracted from an exponential distribution;
  2. a new center of mass is extracted from a truncated normal distribution with mean the old center of mass and standard deviation σ;
  3. a new area is extracted from a truncated exponential distribution with mean the old area, such that it corresponds to a triangular membership function entirely contained in the range of the relevant output variable;
– a new random rule is added to the rule set; the new rule is generated as follows:
  1. for each input variable, a fair coin is flipped to decide whether to include it in the antecedent part, not exceeding four variables;
  2. for each selected input variable, a linguistic value is extracted among those defined;
  3. an output variable and its linguistic value are extracted for the consequent part of the rule;
– a rule is removed from the rule set;
– a rule gets a random antecedent clause predicating an input variable not yet used added to it;
– an antecedent clause is removed from a rule;
– the predicate of an antecedent clause is modified by randomly extracting one of the linguistic values defined for the relevant input variable;
– the predicate of the consequent clause of a rule is modified by randomly extracting one of the linguistic values defined for the relevant output variable.

Migration

Migration is responsible for the diffusion of genetic material between populations residing on different islands. At each generation, with a small probability (the migration rate), a copy of the best individual of an island is sent to all connected islands and as many of the worst individuals as the number of connected islands are replaced with an equal number of immigrants.

Moreover, an immigrant whose fitness is lower than the lowest fitness in the island population is always discarded.

6.2 Fitness

The fitness function is basically a logarithmic likelihood function, which measures how well the predictions made by the model match with the dataset used for learning. A particular care has been taken to assign a relative weight to false-positive and false-negative predictions, to reflect the fact that the loss incurred as a consequence of mistaking a bad customer for a good one are in general a multiple of the loss caused by missing a good prospective customer.

6.3 Selection

Elitist linear ranking selection, with an adjustable selection pressure, is responsible for improvements from one generation to the next. Overall, the algorithm is elitist, in the sense that the best individual in the population is always passed on unchanged to the next generation, without undergoing crossover or mutation.

7 Case Study and Experimental Results

The system has been tested in a real world application in order to validate its results; in order to complete this task, a real dataset coming from an insurance company has been collected, and a special hardware and software configuration has been set up. In the next paragraph we illustrate the experimental results achieved.

Case Study
As mentioned above, a dataset of customers obtained from an insurance company has been collected with the objective of predicting customer values to minimize risk for the company. That is, first of all we collected insurance company database containing relevant customer data required for a comprehensive analysis of each customer profile (historical personal data); starting from there, we extracted from the database only those fields considered relevant to our analysis, obtaining a dataset of 725,035 records (customers), containing 20 fields each.
A rating (*a posteriori* evaluation of each customer expressed by a number in the [0, 1] interval, where 0 is worst and 1 is best) has been added as the last row of this dataset; such value has been computed using the following equation:
Normalized Score = (Historical customer value + Upper bound of Historical customer value) / (Upper bound of Historical customer value + Lower bound of Historical customer value).
Once this task has been completed, we have analyzed the resulting dataset in order to examine the distribution of the rating; the results of our analysis can be described by the following tables:

Dataset analysis - observed value by range					
min		max	min	max	occurrency
-	10.000	- 8.400	-	0,1	4.160
-	8.400	- 6.800	0,10	0,2	706
-	6.800	- 5.200	0,20	0,3	1.055
-	5.200	- 3.600	0,30	0,4	1.864
-	3.600	- 2.000	0,40	0,5	3.857
-	2.000	- 400	0,50	0,6	14.694
-	400	1.200	0,60	0,7	661.309
	1.200	2.800	0,70	0,8	29.067
	2.800	4.400	0,80	0,9	6.493
	4.400	6.000	0,90	1,0	3.655

Rating distribution	
hyst. value	scoring
-5000	0,3125
-4000	0,3750
-3000	0,4375
-2000	0,5000
-1000	0,5625
-500	0,5938
-200	0,6125
-100	0,6188
-50	0,6219
-10	0,6244
-5	0,6247
0	0,6250
5	0,6253
10	0,6256
50	0,6281
100	0,6313
200	0,6375
500	0,6563
1000	0,6875
2000	0,7500
3000	0,8125
4000	0,8750
5000	0,9375

Fig. 2. Dataset analysis - observed value by range

A first consideration has been made: the occurrence of customers with a value within the [0.6-0.7] interval make our dataset strongly unbalanced, and this situation could compromise the reliability of our predictions. Therefore, our next step has consisted in sampling the records to obtained a more balanced dataset; once this operation has been completed, records within the [0.6-0.7] interval have been reduced to 46,291 (instead of 661,309), giving us a final dataset composed by 110,016 total records.
At this point, a classical machine learning protocol has been adopted by splitting the dataset in three different subsets; first of all, a **training** dataset of 20% records has been extracted in order to calculate individual fitness. Then, it has been extracted a test set of 70% records to which each single generation best model calculated by evolutionary algorithms has been applied, in order to obtain a 'test fitness value'.
As long as this test fitness increases, the evolution is carried on; as the test fitness begins to decrease (due to overfitting), evolution has been automatically stopped and the last best model saved. Finally, a **validation** dataset of 10% of records has been extracted in order to validate the whole approach by applying it to the best model previously produced with a specific goal: to validate the resulting best model with customer data never used either for training or test.

Experimental Setting

Once the dataset has been prepared, the physical environment to perform session tests has been set up; eight bi-processors server equipped with Pentium IV 2.4GHz and 1 GB RAM running under Windows 2000 Server communicating by fast Ethernet TCP-IP protocol has been used as evolutionary algorithm server stations.
Here we provide the details of the evolutionary software environment adopted:

- Dataset name: Reale.xml (consisting of 110,016 records: 20% used for training, 70% for test and 10% for validation purposes);

- Initial population: random;
- Number of islands: 16 (2 for each server station);
- Island population size: 50 individuals;
- Crossover rate: 60%; Mutation rate: 15%; Migration rate: 1%;
- Linear ranking selection pressure: 1.8; Environment islands topology: ring.

Results

The evolutionary algorithm automatically stopped its computation when 1,232 generations had been completed on Server #4 (this is the highest generation of the whole environment) due to satisfaction of the self-termination condition with a total computation time of 22 hours and 46 minutes approximately.
The results reached are listed below:

Dataset	Average fitness	Standard deviation
Training	0.44	0.018
Test	0.39	0.023
Validation	0.37	0.027

In order to complete exhaustively our experimentation, a further statistical analysis has been performed on the incidence of fields in the best model (in terms of their weights inside the best model fuzzy rules); the following table shows the main fields report:

Field name	Incidence on model
Number of claims	48.48%
Policy type code	15.12%
Age of customer	12.21%
Duration of relationship w/ customer	9.01%
Profession	6.24%
County of abode	6.03%
Number of COI instalments	3.02%

A brief consideration on these results is in order: Even if the field with greatest incidence could be easily predictable (Number of claims), the other results are quite interesting (especially the second place of Policy type code) and has been considered extremely interesting by the insurance company, because they give (combined with a deep analysis of best model fuzzy rules) a powerful instrument for customer value calculation.

8 Conclusions

It is important to be able to calculate correctly the expected future life time long value of LTV in order to run a sustainable business and to retain customer fidelity for a long period of time. Even by calculating the future LTV and by selecting the only profitable (during short time) customers, the above mentioned customers may result in losses in the long run. Our experiments clearly showed that the availability of advanced data mining tools in the insurance domain is the only way to assist the Technical Management in their effort to obtain maximum profitability.

Acknowledgments. Some of the achievements described in the article are the result of participation in Eureka co-funded projects HPPC/SEA and IKF. The work described in this paper draws upon the contribution of many people, to whom the authors are indebted. Of course the authors are the sole responsible of any possible mistake. The authors would like to thank the IKF Consortium for their support.

References

[1] T. Bäck. Evolutionary algorithms in theory and practice. Oxford University Press, Oxford, 1996.
[2] C. Darwin. On the Origin of Species by Means of Natural Selection. John Murray, 1859.
[3] L. Davis. Handbook of Genetic Algorithms. VNR Computer Library. Van Nostrand Reinhold, New York, 1991.
[4] R. Dawkins. The blind Watchmaker. Norton, 1987.
[5] L. J. Fogel, A. J. Owens, and M. J. Walsh. Artificial Intelligence through Simulated Evolution. John Wiley & Sons, New York, 1966.
[6] D. E. Goldberg. Genetic Algorithms in Search, Optimization & Machine Learning. Addison-Wesley, Reading, MA, 1989.
[7] J. H. Holland. Adaptation in Natural and Artificial Systems. The University of Michigan Press, Ann Arbor, 1975.
[8] J. R. Koza. Genetic Programming: on the programming of computers by means of natural selection. The MIT Press, Cambridge, Massachussets, 1993.
[9] Z. Michalewicz. Genetic Algorithms + Data Structures = Evolution Programs. Springer-Verlag, Berlin, 1992.
[10] M. Beretta, A. Tettamanzi. Learning Fuzzy Classifiers with Evolutionary Algorithms. Proceedings of the 4th Italian Workshop on Fuzzy Logic (WILF 2001), Physica Verlag, 2002.
[11] A. Tettamanzi. An Evolutionary Algorithm for Fuzzy Controller Synthesis and Optimization. IEEE International Conference on Systems, Man and Cybernetics, Vancouver, Canada, 1995.
[12] A. Berson, Stephen J. Smith, Data Warehousing, Data Mining & OLAP, McGraw Hill; New York, 1997.

Multiple Species Weighted Voting – A Genetics-Based Machine Learning System

Alexander F. Tulai and Franz Oppacher

Computer Science Department, Carleton University,
Ottawa, Ontario, Canada, K1S 5B6
alex.tulai-rocaeng@rogers.com
oppacher@scs.carleton.ca

Abstract. Multiple Species Weighted Voting (MSWV) is a genetics-based machine learning (GBML) system with relatively few parameters that combines N two-class classifiers into an N-class classifier. MSWV uses two levels of *speciation,* one manual (a separate species is assigned to each two-class classifier) and one automatic, to reduce the size of the search space and also increase the accuracy of the decision rules discovered. The *population size* of each species is calculated based on the number of examples in the training set and each species is trained independently until a *stopping criterion* is met. During testing the algorithm uses a *weighted voting system* for predicting the class of an instance. MSWV can handle instances with unknown values and post pruning is not required. Using thirty-six real-world learning tasks we show that MSWV *significantly* outperforms a number of well known classification algorithms.

1 Introduction

Data classification, as a supervised learning task, has been one of the most researched subjects and the progress in this area has translated into a large variety of supervised learning algorithms. Among the genetics-base machine learning (GBML) systems, the more successful ones have been the learning classifier systems (LCS) with XCS being the main example [10, 11]. While Multiple Species Weighted Voting (MSWV) algorithm qualifies as a GBML because of its use of populations of individuals and two genetic operators, it is not an LCS.

The use of N 2-class classifiers in N-class classification tasks, that MSWV uses (discovered independently), is also the standard method used by Support Vector Machines [9]. There are also other algorithms using $N(N-1)/2$ SVM classifiers (one for each pair of classes) for an N-class classification task [5].

In section 2 we introduce a terminology and present a background of the classification theory while in section 3 we present the important techniques used by the algorithm. In section 4 we present the experimental results and compare them with well known competing techniques followed by conclusions in section 5.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1263–1274, 2004.

2 Background

A dataset D is a subset of $\mathcal{X} \times \mathcal{C}$ where $\mathcal{X}$ is the input space, also called the feature space, and $\mathcal{C}$ is the class space. In general $\mathcal{X} = X_1 \times \cdots \times X_n$ but in many practical applications $X_i = R$ so $\mathcal{X} = R^n$ and the feature space is n-dimensional. $\mathcal{C} = \{c_1, \ldots, c_N\}$, usually a discrete set of small cardinality N, is the class space. A record (instance or sample) of a data set, is a point $p = \{\mathbf{x}, c_i\} = \{x_1, \ldots, x_n, c_i\}$ where $\{x_j\}_{j=1,n}$ are the feature values and c_i is the class of the record (sometimes called the record tag).

Classification is the problem of predicting the value of an output variable $y \in \mathcal{C}$ based on a given value of the input variable $\mathbf{x} = \{x_1, \ldots, x_n\}$.

An attribute is a Boolean valued function $h_i : X_i \rightarrow \{0,1\}$. A decision rule r is a function $r : \{0,1\}^n \rightarrow \mathcal{C}$ usually built as a conjunction or disjunction of attributes. Decision rules are used in classification and they are usually constructed by a learning algorithm from a training set of L points $p_i = \{\mathbf{x}_i, y_i\}, i = 1, L$ of known input and output values. For a given point $\{\mathbf{x}, c_i\}$ with $\mathbf{x} = \{x_1, \ldots, x_n\}$ a correct classification occurs when $r(\{h_1(x_1), \ldots, h_n(x_n)\}) = \hat{y}(\mathrm{x}) = c_i$, otherwise $(\hat{y}(\mathbf{x}) \neq c_i)$ it is considered an error. MSWV uses decision trees with internal nodes storing Boolean valued attributes and the leaf nodes containing a tag. Because the path from the root node to a leaf node is a conjunction of Boolean attributes, a decision tree can be seen as a disjunction of a limited number of conjunctions.

In general almost all learning algorithms are procedures for obtaining estimates $\{\hat{f}_i(x)\}_1^N$ of the set of conditional probabilities:

$$\{f_i(x) = \Pr(y = c_i \mid x)\}_1^N \tag{1.1}$$

There are two methods for obtaining these probability estimates [5], the density estimation approach and a regression methodology. The density estimation approach uses Bayes' theorem. A well known example of a classification algorithm using the density estimation paradigm is Naïve Bayes [6] but as referenced in [5] there are many others.

The second approach attempts to directly estimate the conditional probabilities (1.1). Examples of algorithms using the regression methodology are the decision tree induction methods [4, 8], the nearest neighbor methods [1] and others. MSWV also belongs to this class of algorithms. In the case of the algorithms using decision tree representations, f would be nothing else but a function built on the Boolean valued attributes $h(x)$ (like a decision rule). MSWV uses a GA to modify these decision trees and search for those trees that contain the best decision rules r giving us the best probability estimates $\{\hat{f}_i(\mathbf{x})\}_1^N$.

The predicted class $\hat{i}$ may be obtained from the probability estimates using:

$$\hat{i}(\mathbf{x}) = \arg \max_{1 \leq i \leq N} L_i \hat{f}_i(\mathbf{x}) \tag{1.2}$$

Very often $L_i = 1$ (see [5]) which is equivalent to assigning the predicted class to be the most probable for a given x. In section 3.7, the equivalent of equation (1.2) will be given by equation (1.6) showing how the decision is made in the case of MSWV.

3 Algorithm Characteristics

In this section we detail the main characteristics of the MSWV system.

3.1 Representation and Species Definition

MSWV uses populations of individuals to create 2-class classifiers. The genotype and phenotype of an individual is a decision tree with internal nodes carrying Boolean valued attributes and leaf nodes carrying one of two possible class labels. What distinguishes MSWV from other algorithms using decision trees is the constraint we impose that each internal node must carry a Boolean valued attribute constructed from a *feature distinct* from the other internal nodes in the tree. This restriction implies that a decision tree can have a maximum of n internal nodes (where n, following the terminology in section 2, is the number of features) and a maximum of $n+1$ leaf nodes. Because the path from the root node to a leaf node represents a decision rule (as a conjunction of Boolean valued attributes) each decision tree can carry a maximum of $n+1$ decision rules. This restriction significantly reduces the size of the search space.

The decision trees are randomly initialized. Each internal node stores a Boolean valued attribute as a triplet ($\boldsymbol{feature\#, featuretype, values}$). The possible features and the attribute functions built on them are:

- *Boolean features (type 1).* $X_i = \{a, b\}$. An internal node stores $(i, 1, v)$ with $v \in \{a, b\}$, $h_i(x_i) = 1 \Leftrightarrow x_i = v$.
- *Set features (type 2).* $X_i = \{x_i^1, \cdots x_i^k\}$. An internal node stores $(i, 2, A)$ with $A \subseteq X_i$, $h_i(x_i^j) = 1 \Leftrightarrow x_i^j \in A$
- *Numerical features (Integer type 3, Floating point type 4).* $X_i = [x_i^{\min}, x_i^{\max}]$, $x_i \in Z \mid R$. A node stores $(i, 3 \mid 4, \{v_1, v_2\})$, $[v_1, v_2] \subseteq X_i$, $h_i(x_i) = 1 \Leftrightarrow x_i \in [v_1, v_2]$

Each 2-class classifier is represented by a population of decision trees with leaf nodes carrying one of two possible class labels, either a class $c_i \in C$ or the "don't know" class that we tag as "?"(i.e. $? = C - \{c_i\}$). A population of individuals recognizing the same two classes $\{c_i, ?\}$ forms a species, so each species is a 2-class classifier. For a N-class classification task, MSWV creates N species.

The training is done using the 1-v-r (one-versus-rest) method. For a given training set T the N classes induce a partition $T = T_1 \cup \ldots \cup T_N$ with T_i containing only the records tagged with c_i. The *i*-th species (corresponding to the *i*-th 2-class classifier)

is trained with the examples in T_i as positive examples and all other examples $(T - T_i)$ as negative examples (treated as the "don't know" class).

The size of the *i*-th species population is automatically calculated with the formula:

$$m_i = \begin{cases} 20 & \text{if } card(T_i)/2 < 20 \\ 200 & \text{if } card(T_i)/2 > 200 \\ card(T_i)/2 & \text{otherwise} \end{cases} \quad (1.3)$$

The reason behind using a population size of $card(T_i)/2$ is linked to the assumption that in the worst case the training subset T_i may contain just 2 examples for each possible classification rule that needs to be discovered in which case the maximum number of rules the *i*-th species should discover (although there is no guarantee they will be discovered) is $card(T_i)/2$. The upper limit of 200 on the population size is imposed by computational considerations. The lower limit of 20 is imposed by training time considerations and will be discussed in section 3.6.

3.2 Mutation Operators

In its current version MSWV uses only mutation and selection (described in the next section). Every generation all the individuals of a species (the species could be trained independently and in parallel) are mutated with probability 1, followed by selection. MSWV implements 6 mutation operators but with equal probability uses only one when mutation is applied. The six mutation operators are shown in Table 1. The term "randomly" refers to a uniform distribution random variable.

3.3 Rule Discovery and Automatic Speciation

One can classify the decision rules into *weak* and *strong* rules depending on whether they cover a small or a large number of training examples. There are datasets governed by one (or a very small number of) strong classification rule(s) in which case placing all the individuals in one large population and having them compete with each other by applying the selection operator seems to be the right way to go. Copies of the fittest individual will propagate through the population of decision trees resulting in an accelerated search for the strongest classification rule.

On the other hand, in general, the training examples are covered by a mixture of strong and weak rules [3]. This would suggest that following a simple hill climbing strategy by applying selection only between a parent and its offspring (we call this *restricted* selection) while slowing down the process of rule discovery, would ensure that both the strong and the weak rules were pursued by the evolving population.

MSWV achieves a compromise between the two extreme cases. Initially, each species starts with a population of hill climbers, individuals that mutate and use restricted selection to maintain the diversity of the evolved rules. Periodically the algorithm checks for *similar* individuals and places them together forming subspecies. The similarity is defined not in terms of genetic makeup but in terms of proficiency in

Table 1. The mutation operators used by MSWV.

Operator	Description
Add node	Adds a new internal node (and a new attribute)
Remove node	Randomly selects a leaf node and removes it and its parent node
Swap two leaves	Randomly selects a pair of sibling leaves and swaps them.
Swap internal nodes	Randomly selects two internal nodes and swaps them
Swap two branches	Randomly selects two internal nodes and swaps the sub-trees headed by them
Attribute change	*Boolean features*: the value used in the node is replaced by its complement
	Set features: With equal probability either adds one element to the set stored in the node or removes one
	Numerical features: Randomly either shift the range up | down or shrink | expand the range by a random gain

classifying positive and negative examples. The speciation process is formally described as follows. Let S_i represent the set of all individuals in species i, trained with positive examples from the set T_i and negative examples from the set $T - T_i$. We introduce a pre-order relation on the elements of S_i. If $s_1, s_2 \in S_i$ are two individuals from species i and by $s_1(T_i) \subseteq T_i$, $s_1(T - T_i) \subseteq T - T_i$, $s_2(T_i) \subseteq T_i$ and $s_2(T - T_i) \subseteq T - T_i$ we denote the sets of positive and negative examples correctly classified by s_1 and s_2 we say that

$$s_1 \geq s_2 \Leftrightarrow s_1(T_i) \supseteq s_2(T_i) \;\&\; s_1(T - T_i) \supseteq s_2(T - T_i) \tag{1.4}$$

In other words, an individual is greater than or equal to another individual (from the same species) if and only if the first individual can correctly classify all the positive and all the negative training examples the second individual classifies. It can be easily shown that the relation "$\geq$" is reflexive and transitive but not antisymmetric. MSWV uses the pre-order relationship on the elements of S_i to create subspecies of individuals that are allowed to compete with each other and accelerate the search for specific decision rules. Because it is computationally intensive this operation (that we call *automatic speciation*) is not performed every generation but at regular intervals. All the experiments described in section 4 have used a speciation period of 50 generations. When the speciation operation is performed the second (or subsequent) time subspecies may already exist in the population. In this case only the fittest individuals from each subspecies participate in the speciation process in which case the pre-order relation between individuals will result in mergers between their respective subspecies. The use of the fittest individual of a subspecies is justified by the lack of diversity that characterizes the subspecies, as a result of full selection

pressure. We would like to point out that individuals of a subspecies may also be allowed to mate if an effective crossover operator is devised in the future.

3.4 Rule Confidence and Rule Weight

As previously mentioned, a decision tree encodes as many decision rules as there are leaf nodes in the tree. Although not mentioned in section 3.1, associated with each leaf there is some additional information built during the learning phase.

- A counter P that counts all the positive examples that have exercised the tree path (i.e. decision rule) ending at this leaf.
- A counter N that counts all the negative examples that have exercised the tree path ending at this leaf.
- A rule confidence coefficient calculated, following training, as follows:

  $\mu = \frac{P}{P+N}$ (also called the true positive rate) for leaves tagged with c_i

  $\mu = \frac{N}{P+N}$ (also called the true negative rate) for leaves tagged with ?

The rule confidence coefficients are used to weight the individual vote during the testing phase (see section 3.7) and remove the need for post pruning.

Datasets often contain instances with unknown values, and learning algorithms must be able to deal with them with minimal degradation of performance.

MSWV treats the unknown (missing) values similar but not identical to other practical learning schemes like C4.5 [8] and PART [4]. When a record is evaluated (during training or testing) a path weight variable pW, initially set to 1, is associated with the path taken by the record through the decision tree. Associated with each internal node of a decision tree there are 4 variables (p_l, p_r, n_l, n_r) counting how many positive or negative examples have gone left or right. If an instance cannot be assigned deterministically to a branch because of an unknown value we assume that it may go on both branches and consequently two paths are created from that node down. The path weights of these two paths are updated using the 4 variables stored in the node with separate equations depending on whether we are testing or training.

During testing, the left and right path weight are given by $pW \bullet \rho_l$ and $pW \bullet \rho_r$ where

$\rho_l = (p_l + n_l)/(p_l + p_r + n_l + n_r)$, $\rho_r = (p_r + n_r)/(p_l + p_r + n_l + n_r)$

During training after calculating the path weight updating factors ρ_l and ρ_r we also have to (on-line) update the variables (p_l, p_r) or (n_l, n_r):

$\rho_l = p_l/(p_l + p_r)$, $\rho_r = p_r/(p_l + p_r)$, $p_l = p_l + pW_l$, $p_r = p_r + pW_r$

$\rho_l = n_l/(n_l + n_r)$, $\rho_r = n_r/(n_l + n_r)$, $n_l = n_l + pW_l$, $n_r = n_r + pW_r$

Because a path from the root node to a leaf node defines a rule, a path weight is nothing else but a rule weight. In section 3.5 we show how the path weights (or rule weights) are used during training (in fitness calculation) and in section 3.7 we show how they are used during testing (in vote weighting).

3.5 Individual Fitness Calculation

The fitness of an individual is calculated only once, when the individual is born, using the training data set. When during the process of evaluating a decision tree shaped chromosome on a training example a leaf node is reached the following updates take place. If the leaf has a tag c_i that matches the class of the example we increment the positive examples counter P and give a reward, otherwise we increment the negative examples counter N and give no reward. If the leaf is tagged " ? " and the example is tagged c_i we increment the positive examples counter P and give no reward, otherwise we increment the negative examples counter N and give a reward.

The reward score for correctly classifying an example depends on the path weight. However we need to correct for the imbalance that may exist in the data set. If a class has very few representatives we need to give a higher reward for correctly classifying samples from this class. Consequently, the reward for correctly classifying a positive example $x \in T_i \subset T$ is $r_p(x) = \sum_{all\ paths} pW \cdot card(T)/card(T_i)$. The reward for classifying a negative example $x \in T - T_i$ (at a leaf node tagged with the unknown class "?") is given by the equation: $r_n(x) = \sum_{all\ paths} pW \cdot card(T)/card(T - T_i)$.

The sum over all paths is required for handling unknown values as explained in section 3.4. With these definitions, the fitness of an individual $s \in S_i$ is given by: $g_i(s) = (\sum_{x \in T_i} r_p(x) + \sum_{x \in T - T_i} r_n(x))/2$ (therefore $g_i(s) \in [0, card(T)]$).

One of the major concerns in machine learning is the generalization power of the rules learned. To encourage the emergence of short rules with a high generalization potential, MSWV reduces an individual's fitness by a penalty factor defined as: $\delta = 0.0005(n-1)L$ where n is the number of internal nodes in the tree (and distinct features used in the decision tree), L is the number of records in the training data set and the constant 0.0005 has been experimentally determined. Assuming a training dataset of 1000 instances, this small penalty factor implies that adding a new internal node to a decision tree must be justified by an increase in fitness greater than 0.5.

3.6 Training Time

Unlike other GBML systems that use a fixed, experimentally determined training time parameter [2, 3], MSWV uses a dynamic stopping criterion. The reasons why a stopping criterion is superior to a fixed training time method are:

- if the training phase stops too early the achieved rule accuracy may be too low
- a training phase that is too long may encourage data overfitting
- the "optimum" training time may greatly vary from dataset to dataset
- it improves the algorithm efficiency (stopped species do not consume CPU time)

Every Δ generations MSWV calculates the average fitness of all the individuals of a species. If the average fitness of all individuals of species i at time $t = j\Delta$ is given by $\bar{g}_i(j\Delta)$, the training will stop at time $j\Delta$ if the following condition is met

$$\frac{\overline{g}_i(j\Delta) - \overline{g}_i((j-3)\Delta)}{3} \leq 0.02 \max_{k \geq 1}(\overline{g}_i(k\Delta) - \overline{g}_i((k-1)\Delta)) \tag{1.5}$$

MSWV system uses a value of 10 for Δ. Put in words, a species is evolved until the increase in the (smoothed over 2Δ) average fitness of its individuals falls below 2% of the highest jump in average fitness over Δ generations. Because the highest jump in fitness usually occurs in the first Δ generations for almost all datasets it is reasonable to ensure the same behavior in the case of those datasets that have a very small number of examples for a certain class (these datasets are very rare but they do exist in the real-world). It is because of these special cases that we established a lower limit on population of 20 (see section 3.1) to increase the probability of positive evolutionary changes and a non zero average fitness increase early in the training.

3.7 Testing

Given an instance $\mathbf{x}$ from a testing dataset with N classes, MSWV uses a voting scheme to predict its class. Because the individuals in a subspecies are all very similar (due to the strong selection pressure) only the fittest individual in the subspecies is allowed to vote. Experiments performed with a very large number of datasets have shown there is no difference in results if all are allowed to vote or only the best is allowed to vote (but this method is more efficient). Let's assume species i finishes the training phase, with its population distributed in M_i subspecies. Let's also denote by M the maximum number of subspecies in any of the N species (i.e. $M = \max_{i=1,N} card(M_i)$). The outcome of the weighted voting process is given by the following formula (also see equation 1.2 for reference):

$$\hat{i}(\mathbf{x}) = \arg\max_{1 \leq i \leq N} \frac{M}{card(M_i)} \sum_{j=1}^{M_i} v_{ij}(\mathbf{x}) \tag{1.6}$$

where $v_{ij}(\mathbf{x})$ is the confidence and path (rule) weighted vote of the fittest individual from the j-th subspecies of the i-th species and is given by the formula:

$$v_{ij}(\mathbf{x}) = \sum_{all\ paths} (-1)^{1(leafTag == "?")} \mu \bullet pW, 1(x) = \begin{cases} 1 \; if \; x = true \\ 0 \; if \; x = false \end{cases} \tag{1.7}$$

where μ is the confidence coefficient calculated as explained in section 3.4, pW is the path weight and the summation over all possible paths is required for handling instances with unknowns (as explained in section 3.4). Confidence weighting removes the need for post pruning (for example, the vote of a rule with confidence 0, is 0).

4 Experimental Results

The performance of MSWV has been evaluated on a set of thirty-six standard datasets available from UCI at: www.ics.uci.edu/~mlearn/MLRepository.html. The thirty-six datasets used for the study exhibit a large variety of characteristics in terms of number of instances, number and diversity of features, number of classes, missing feature values, etc. The datasets and their characteristics are listed in Table 2.

Table 2. The datasets used for experiments (unknown values given as a percentage of total).

Dataset	Id	Inst	Unknown	#feat	#nom	#num	#class
Audiology	aud	226	2.0	69	69		24
Balance	bal	625	-	4		4	3
Breast-w	brw	699	0.3	9		9	2
Bupa	bpa	345	-	6		6	2
Contraceptive	cmc	1473	-	9	4	5	3
Credit-austral.	cra	690	0.6	15	9	6	2
Credit-german	crg	1000	-	20	13	7	2
Glass	gls	214	-	9		9	6
Heart-c	h-c	303	0.2	13	7	6	2
Heart-h	h-h	294	20.5	13	7	6	2
Hepatitis	hep	155	5.6	19	12	7	2
Ionosphere	ion	351	-	34		34	2
Iris	irs	150	-	4		4	3
Labor	lbr	57	35.7	16	8	8	2
Pima-indians	pmi	768	-	8		8	2
Primary-tumor	prt	339	3.9	17	17		22
Sonar	snr	208	-	60		60	2
Soybean	soy	683	9.8	35	35		19
Vehicle	veh	846	-	18		18	4
Vote	vot	435	5.6	16	16		2
Vowel	vow	990	-	13	3	10	10
Wine	wne	178	-	13		13	3
Yeast	yst	1484	-	8		8	10
Zoo	zoo	101	-	16	16		7
Adult	ADL	48842	0.9	14	8	6	2
Hypothyroid	HTH	3772	5.5	29	22	7	4
Kr-vs-kp	KRK	3196	-	36	36		2
Led (10%noise)	LED	6000	-	7	7		10
Letter	LTT	20000	-	16		16	26
Mushrooms	MUS	8124	1.4	22	22		2
Optical-digits	ODG	5620	-	64		64	10
Satellite-image	SAT	6435	-	36		36	6
Segment	SEG	2310	-	19		19	7
Sick	SIC	3772	5.5	29	22	7	2
Splice	SPL	3190	-	60	60		3
Waveform+noise	WVF	5000	-	40		40	3

The performance of MSWV is compared to six other learning algorithms: NB [6], IB1 and IB3 (IBk or Instance-Based learner [1] that we use with k=1,3) , C4.5 (revision 8, an induction tree algorithm, [8]) , PART (an algorithm for inferring rules by repeatedly generating partial decision trees, [4]) and SMO (Sequential Minimal Optimization, a SVM classifier system [7]), all of them run using the Weka v3.4 package [12] available at: www.cs.waikato.ac.nz/ml/weka

The datasets are divided in two groups, twenty-four small (under 2000 instances) datasets and twelve large (over 2000 instances) datasets.

On the group of small datasets (lower case IDs) the comparison between classifiers is based on averaging ten ten-fold cross-validation runs. To test the statistical significance of the differences between classifiers on this group of datasets we used a paired two-sided *t*-test at 99% confidence level.

On the group of large datasets (upper case IDs) the comparison is based on the holdout estimate, where 33.3% of the instances are used for training and 66.6% are used for testing. The statistical significance of the differences between classifiers is performed using a test for the differences of two proportions at 99% confidence level. The large datasets have been shuffled offline and we have ensured that all classification systems have used the same partitions for training and testing.

Finally, the paired Wilcoxon signed rank test is used to calculate the statistical significance of the overall observed differences between two learning systems.

The accuracy rates (given as percentage of correct classifications) of the classification systems are summarized in Table 3. Results for the six learning systems used in the study are marked with ○ if they show significant improvement over the corresponding results for MSWV, and with ● if they show significant degradation.

The results presented in Table 3 can serve as basis for several observations. The first observation is that on this set of thirty-six datasets MSWV significantly outperforms 4 other classification systems (NB, IB1, C4.5 and PART), slightly outperforms IB3 and it is outperformed by SMO. This conclusion is based on the results presented in the row labeled W-L-T (wins-losses-ties) and also from the outcome of the paired Wilcoxon signed rank test as shown in the row labeled "Confidence". For example, MSWV significantly outperforms C4.5 on 17 datasets and it is significantly outperformed by C4.5 on 7 datasets. Using the results of the Wilcoxon test we can say that on this group of datasets MSWV improves C4.5 with 97.6% confidence but it is also improved by SMO with 54.7% confidence. The comparison with the other classification systems is done similarly.

Using a similar methodology and the experimental results on thirty datasets, [2] reported that XCS and their own UCS (both are Learning Classifier Systems) were outperformed by C4.5, PART and IB3. The SMO classifier used in our study could handle nominal and numerical valued features (different from the version used in [2]). We can indirectly conclude that MSWV may outperform XCS and UCS on a subgroup of datasets. In addition to that, MSWV does not exhibit the deterioration in performance reported in [2] on datasets with high number of classes like *aud, soy* or *vow* (see Table 2 for IDs and Table 3 for MSWV's results).

As shown in Table 3, MSWV is outperformed by all other six classifier systems on one single dataset, *ADL* that happens to be the dataset with the highest number of instances. Future experiments on datasets with very large number of instances (> 20,000) will help clarifying whether the performance degrades for very large datasets.

MSWV has outperformed all other classifiers on two datasets, *gls* and *yst* that have continuous numerical features and more than two classes. This may suggest that either MSWV does particularly well on this kind of datasets or, alternatively, the performance of the other six classifiers on datasets with continuous numerical features does not match their performance on datasets with integer or nominal features (this second alternative may be particularly true for the SMO classifier).

Table 3. Accuracy rates (%) of the classifiers used in the study. The row labelled W-L-T counts the wins-losses-ties of one-on-one comparison between MSWV and another classifier. The last two rows show the z-value and the confidence level of the Wilcoxon signed rank test.

	MSWV	NB		IB1		IB3		C4.5		PART		SMO	
aud	76.86	72.61	•	75.00		78.41		77.26		79.42	○	80.75	○
bal	86.62	90.53	○	79.28	•	86.74		77.82	•	83.17	•	87.42	○
brw	96.91	96.07	•	95.44	•	96.61		95.01	•	94.69	•	96.75	
bpa	67.39	54.87	•	62.20	•	62.49	•	65.83		65.25	•	58.00	•
cmc	52.99	49.72	•	42.49	•	45.11	•	52.73		50.17	•	49.82	•
cra	85.72	77.86	•	81.57	•	84.96	•	85.57		84.45	•	84.88	•
crg	73.59	75.16	○	71.88	•	72.21	•	71.25	•	70.54	•	75.15	○
gls	72.01	49.44	•	69.95	•	70.00	•	67.62	•	68.74	•	57.34	•
h-c	82.44	83.33	○	76.04	•	81.82		76.93	•	77.99	•	83.89	○
h-h	83.40	83.95		78.33	•	82.07	•	80.20	•	81.12	•	82.79	
hep	84.26	83.81		81.35	•	80.84	•	79.23	•	79.74	•	85.68	○
ion	91.03	82.17	•	87.09	•	86.01	•	89.74	•	90.83		88.06	•
irs	94.80	95.53		95.40		95.20		94.73		94.20		96.20	○
lbr	84.39	93.51	○	84.21		92.81	○	78.77	•	77.89	•	92.98	○
pmi	74.61	75.76	○	70.62	•	73.87		74.49		73.46		76.80	○
prt	44.96	49.71	○	34.34	•	44.99		41.39	•	40.86	•	46.96	○
snr	80.53	67.69	•	86.15	○	83.75	○	73.61	•	77.40	•	76.49	•
soy	91.40	92.81	○	90.07	•	91.07		91.65		91.27		92.97	○
veh	67.55	44.68	•	69.59	○	70.21	○	72.28	○	72.21	○	74.08	○
vow	87.47	62.90	•	99.05	○	96.99	○	80.20	•	77.67	•	70.59	•
vot	95.08	90.02	•	92.44	•	93.08	•	96.57	○	95.98	○	95.77	○
wne	96.01	97.47	○	95.11		95.84		93.20	•	92.08	•	98.76	○
yst	58.76	57.99	•	52.61	•	55.09	•	56.61	•	54.81	•	57.10	•
zoo	93.86	94.95		96.53	○	95.54		92.57		93.37		96.04	○
ADL	75.91	83.38	○	78.75	○	81.74	○	85.92	○	84.80	○	84.98	○
HTH	93.52	95.63	○	90.66	•	92.64		99.20	○	99.20	○	93.83	
KRK	94.60	84.61	•	87.56	•	92.07	•	99.01	○	98.22	○	94.23	
LED	74.33	74.08		62.48	•	74.25		74.28		74.23		74.03	
LTT	76.29	64.20	•	93.02	○	91.55	○	81.33	○	82.42	○	79.18	○
MUS	99.89	94.43	•	100.0		99.91		100.0		99.88		100.0	
ODG	92.93	90.71	•	97.84	○	98.05	○	86.55	•	87.70	•	97.12	○
SAT	61.91	61.42		69.04	○	69.53	○	59.62		61.35		69.63	○
SEG	92.98	79.22	•	95.52	○	94.35		93.50		93.90		90.71	
SIC	97.10	93.04	•	95.30	•	95.94		98.25	○	97.89		93.76	•
SPL	95.63	94.54		73.01	•	77.39	•	92.05	•	90.79	•	93.04	•
WVF	82.27	79.75		71.29	•	75.88	•	73.45	•	76.63	•	85.99	○
W-L-T		18-10-8		22-9-5		13-8-15		17-7-12		19-7-10		10-19-7	
z-value		2.31		2.2		0.31		2.26		2.26		-0.75	
Confid.		97.91		97.22		24.34		97.62		97.62		-54.67	

The performance of MSWV on two large datasets that have noise artificially added, *LED* and *WVF,* is also good, as shown in Table 3.

5 Conclusions

MSWV uses a GA to evolve populations of decision trees (each population a 2-class classifier) but uses a calculated population size and a training stopping criterion rather than some fixed values. Other variables (like speciation period or fitness averaging interval) have been kept constant in all our experiments supporting the claim that MSWV is a relatively parameter free GBML system. An innovative speciation mechanism ensures that strong and weak rules are pursued by the evolving population at the best pace possible. The restriction that the decision tree be constructed on attributes built on distinct features clearly helps the search but seems to introduce a limit on the complexity of the decision rules that can emerge in such trees and to negatively impact the classifier accuracy. This potential problem is successfully compensated by the statistical power of a voting system, making classification a collective task. Tested on a large number of real world classification tasks the algorithm significantly outperforms a number of well known classification algorithms. MSWV performs well on noisy datasets, can handle records with unknown values and does not require post pruning. Based on the experimental results we believe that at this moment MSWV is the best GBML system reported in the EC literature.

References

1. Aha, D., Kibler, D., Albert, M.: Instance-based learning algorithms. Machine Learning vol. 6, (1991) 37-66
2. Bernadó-Mansilla, E., Garrell-Guiu, J.M.: Accuracy-based learning classifier systems: models, analysis and applications to classification tasks. Evol. Comput.11(3) (2003) 209-238
3. Carvalho, D.R. , Freitas A.A.: A genetic algorithm with sequential niching for discovering small-disjunct rules. Genetic and Evolutionary Computation Conference proceedings. Morgan Kauffman publishers, San Francisco, CA (2002).
4. Frank, E., Witten, I.H.: Generating accurate rule sets without global optimization. In: Shavlik, J., editor, Machine Learning proceedings of the 15-th international conference. Morgan Kaufmann (1998) 144-151
5. Friedman, J.H., Another approach to polychotomous classification. Technical report, Stanford department of Statistics [www-stat.stanford.edu/reports/friedman/poly.ps.Z] (1996)
6. John, G., Langley, P.: Estimating Continuous Distributions in Bayesian Classifiers. In: P. Besnard and S. Hanks (Eds.) 11-th Annual Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann Publishers, San Mateo (1995)
7. Platt, J.: Fast training of support vector machines using sequential minimal optimization. In: Advances in Kernel Methods – Support Vector Learning. Cambridge, MA:MIT Press (1998)
8. Quinlan, J.R. C4.5: Programs for Machine Learning, Morgan Kaufmann Publisher (1993)
9. Vapnik, V.N.: Statistical Learning Theory. Wiley, New York, NY, USA (1998)
10. Wilson, S.W.: Classifier fitness based on accuracy. Evolutionary Computation (1995), 3(2) 149-175
11. Wilson, S.W.: Generalization in the XCS classifier system. In: Genetic Programming Proceedings of the Third Annual Conference, J. Koza et al., eds., San Francisco, CA: Morgan Kaufmann (1998) 665-674
12. Witten, I.H., Frank, E.: Data mining practical Machine Learning tools and techniques with Java implementations. Morgan Kauffman Publishers (2000)

Object Oriented Design and Implementation of a General Evolutionary Algorithm

Róbert Ványi

Department of Theoretical Computer Science
Friedrich-Alexander University, Erlangen-Nürnberg
Martensstr. 3, D-91058 Erlangen, Germany
vanyi@cs.fau.de

Abstract. There are many types of evolutionary algorithms, just like genetic algorithms, evolution strategies or genetic programming. Thus there are a lot of implementations of different EAs as well. One can find many tools, programming libraries written in a wide variety of programming languages, implementing several features of Evolutionary Algorithms. However, many of them misses one or other feature by restricting the algorithm to a certain type. The aim of the work presented in this paper is to introduce a general, object oriented EA model, and to present an implementation of it. One of the most important advantages of this system is that it uses a generalized view of EAs and evolutionary operators, thus any combination of different types of evolutionary operators can be used with lots of parameters. The system is also easy to extend, and still easy to understand and use. If one wants to use the already implemented data types and operators, implementing an EA is an easy ride. For example an evolutionary algorithm solving a polynomial approximation problem for real-valued polynomials can be implemented with only 100 lines of C++ code.

1 Introduction

When someone decides to use evolutionary algorithms to solve a problem, a decision is also made, what kind of EA is used. This usually determines the possibly used operators, and the representations. The most common types are

- evolution strategies [Rec73] using real vectors, and operators on real numbers (adding a random number, averaging, and so on),
- genetic algorithms [Hol75] [Gol89] with bitvector representations and bit operators, and
- genetic programming [Koz92] with tree data structures, and tree operators.

However, one may want to combine several properties of these methods and thus many home-brew systems cannot be considered as pure GAs, ESs or GPs. The available programming environments, however, usually consider one of these models, therefore they are not general enough.

In this paper a general EA model is constructed and an object oriented implementation in the programming language ANSI C++ is presented. Advantages of this object oriented C++ implementation are

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1275–1286, 2004.

- *easy to understand:* a small number of clearly defined classes,
- *easy to use:* simple problems can be solved with 100 lines of code,
- *easy to extend:* new representation types and operators can be implemented using inheritance
- *general:* a general view of EAs and operators is used
- *fast:* combines OO and the efficiency of C

1.1 Related Work

There are many implementations of evolutionary algorithms. Many of them was examined, but no general EA implementation was found. SGA-C [SGE94] and GENESIS [Gre84] were implemented in C, thus the advantages of the object oriented programming cannot be used. SGA-C was not meant to be a general tool, and GENESIS concentrates only on GAs as well. Evolvica [Evo] and its predecessor eaLib [eaL] are implemented in Java, and though sometimes Java programs may be fast, the author still prefers C++ over Java. Furthermore eaLib have many classes, making its usage difficult, though there are many examples in the documentation. Evolvica intends to be a programming framework, so it is not only a library. From the libraries implemented in C++ three were examined, GAlib [Wal96], EO [Mer] and GEA [Tót00]. GAlib has many nice features, can handle different individuals and operators, and these can also be extended. But it still does not use a general approach either for the algorithm or for the operators. EO is also a fully-featured implementation, based on C++ templates available for many platforms. However, it also lacks a general EA implementation and the general approach to evolutionary operators, though different types of algorithms can be implemented by subclassing a given algorithm class. GEA was developed to overcome some disadvantages of these systems, but it does not consider giving a general model for EAs or operators either.

As it can be seen, none of these libraries, systems use a general EA view, and all of them use more or less the usual approach for operators, that is having a crossover (recombination) followed by a mutation. To create a system having these missing features is the aim of the work presented here. On the other hand these systems have many nice features, not considered by the general EA design and implementation introduced in this paper, but the detailed comparison of these systems is beyond the scope of this paper.

We proceed as follows. First a general evolutionary algorithm model is introduced. In Section 3 an object oriented design is given for the general EA model. In the pseudocode an object oriented notation is used, that is attributes and methods of objects are written as *objectname.attr,* and *objectname.func*(). This design is discussed further with concrete implementation details in Section 4. In Section 5 some examples are given how a specific problem can be solved with the implemented general EA. Finally in Section 6 conclusions are drawn and some future plans are mentioned.

2 General Evolutionary Algorithm

To design a general evolutionary algorithm a very simple approach is used. The EAs store a set of objects, do something with them, and create a new set of objects. Doing something with the objects means that several evolutionary operators and selections are applied on them. Sometimes selections are handled as operators, but in our case they are distinguished from each other. When searching in literature several types of selections can be found. One can select individuals to be parents, one can select individuals from the generated individuals, that is from the offsprings to be survivors. One can also directly copy individuals into the offsprings or even into the survivors. The latter one is called elitism. Sometimes there are not enough survivors to make up a new population, so new individuals may be created randomly. The selections use a fitness function to measure the goodness. To create a general evolutionary algorithm, it is allowed for each selection to have an own fitness function.

Using all these components (4 selections with fitness functions and 2 operators) a general evolutionary loop can be constructed that can be seen in Figure 1. In this figure circles represent populations, that is sets of individuals. Squares are selections and operators, which take a set of individuals and return a new set of individuals. These individuals may be only filtered (selections) or newly generated (operators). The selections have a goodness measure, that is a fitness function, according to which they filter the populations. These fitness functions are F_{ps}, F_{cp}, F_{el} and F_{ss} for the different selections.

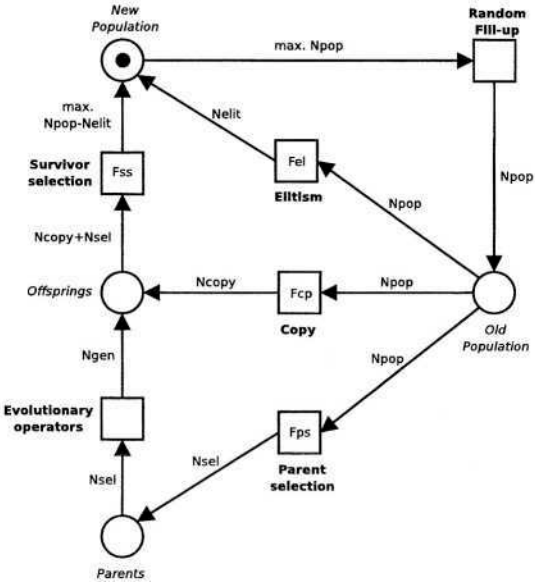


Fig. 1. General EA loop

Before the algorithm starts, several constants are defined. These are N_{pop}, N_{sel}, N_{gen}, N_{copy} and N_{elit}, showing the size of the population, the number of selected, generated and copied individuals and the size of the elitism respectively. The loop begins with a new population marked with a dot. If there are less than N_{pop} elements, the population is filled up with randomly generated individuals. Parent selection using fitness function F_{ps} selects N_{sel} individuals. Then the evolutionary operators are applied on these individuals and N_{gen} new individuals are created, and inserted into the offspring population. From the original population N_{copy} elements are added to this population using copy selection with fitness function F_{cp}. The survivors are selected from the offspring population using survivor selection with fitness function F_{ss} into the new population. Meanwhile N_{elit} individuals are inserted directly from the old population into the new population by the elitism using fitness function F_{el}. This loop can be implemented as Algorithm 1 shows.

Algorithm 1 General EA loop

```
GENERAL_EA_LOOP(new_pop)
 1  while new_pop.size < N_pop
 2  do new_pop.insert(Random_Individual())
 3  old_pop ← new_pop
 4  parents ← parent_sel.select(old_pop, N_psel)
 5  offsprings ← operators.apply(parents, N_gen)
 6  offsprings += copy_sel.select(old_pop, N_copy)
 7  N_surv ← min{N_copy + N_sel, N_pop − N_elit}
 8  new_pop ← surv_sel.select(offsprings, N_surv)
 9  new_pop += elitism.select(old_pop, N_elit)
10  return new_pop
```

The experienced reader may have recognized that this model does not consider the phenotype and the genotype. This model simply forgets the phenotype. It is only needed for fitness evaluation, but the fitness function can be considered as a composition of the genotype decoding function and the phenotype evaluation function, thus the phenotype never appears in the computer point of view.

2.1 Generalized Evolutionary Operator

In a common evolutionary algorithm it is usual that first recombination is applied on the population and then mutation is applied on some individuals. This can be generalized, and more operators may be allowed. That is several operators can be applied on the population sequentially. However, one may have more mutations, and want to choose between them. Thus not only a sequential, but also a parallel application has to be allowed. By generalizing this idea the *operator groups* can be defined. An operator group can be a single operator or a set of operators to be applied sequentially or parallelly. So the operators and operator groups can

be organized into a tree-like structure, where the leafs are operators and the internal nodes are operator groups. The three types of groups can be seen in Figure 2 and are the following

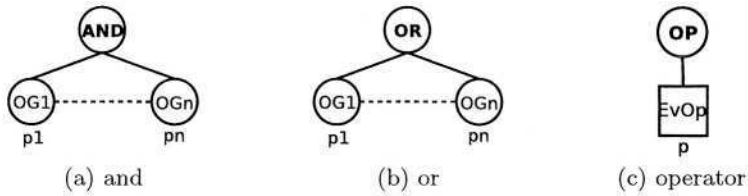


Fig. 2. Types of operator groups

- AND – the operator groups are applied sequentially from 1 to n on the whole population, using probability p_i.
- OR – the operator groups are applied parallelly on the population, and the result is the union of the resulted populations. The probability of inserting an individual created by group i into the result is p_i.
- OP – the evolutionary operator is applied on the whole population.

The apply method of the operator group takes two populations – an input and an output population – and a size, that is the number of individuals to be generated. The output population does not have to be empty, the new objects are inserted to the end of it. The AND group takes the output of an operator group as the input of the next one. The input of the first one is the input population, and the output of the last one is inserted into the output population, as it can be seen in Algorithm 2.

Algorithm 2 Applying an AND group

```
APPLY(input_population, output_population, size)
1  temporary_population1 ← input_population
2  temporary_population2 ← empty_population()
3  for i ← 1 to n
4  do og[i].apply(temporary_population1, temporary_population2, size)
5     temporary_population1 ← temporary_population2
6     temporary_population2 ← empty_population()
7  op.merge_with(temporary_population1)
```

The OR group first partitions the required size into n partitions using the probabilities of the operator groups. Then it uses the operator groups to create individuals according to the partitions. The created individuals are inserted directly into the output population. This method can be seen in Algorithm 3.

Algorithm 3 Applying an OR group

```
APPLY(input_population, output_population, size)
1  sizes ← weighed_partitions(size, probabilities)
2  for i ← 1 to n
3  do og[i].apply(input_population, output_population, sizes[i])
```

3 Object Oriented EA

In this section an object oriented evolutionary algorithm is designed using the model and ideas from the previous section. The central object in the system is of course the evolutionary algorithm itself, but first the most important building block is designed, and it is the individual, that is the representation of the problem.

3.1 Individuals

The most common way to implement individuals is to create an evolvable object, and then use its descendants (subclasses) as representations for the problem, as outlined in Figure 3(a) The Evolvable class gives the common interface, so the

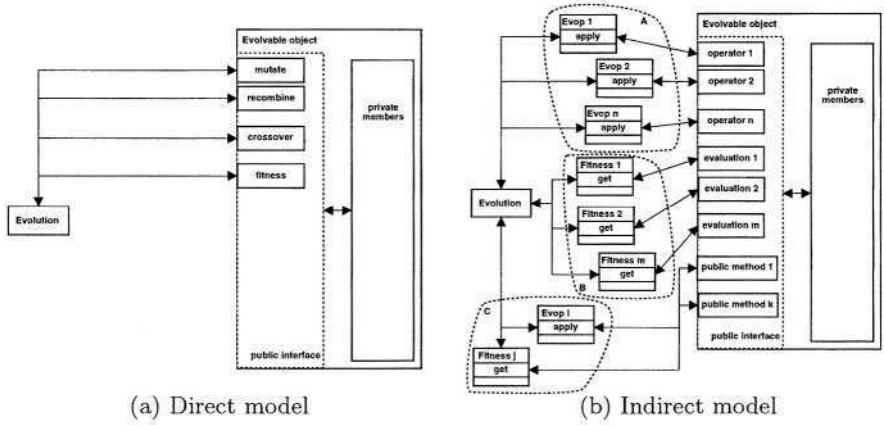


Fig. 3. Different models for Evolvable objects

evolutionary algorithm can modify the objects, and get their fitnesses. This will be referred to as *direct access,* since the EA knows the interface of the Evolvable class, and directly modifies the object. The direct model is used by the previously mentioned C++ implementations of EA (GAlib, EO, GEA).

However, in this paper another approach is followed. Since it is not known, how many operators the user wants to use, or how the fitness is calculated, the Evolvable subclasses cannot be forced to implement a given number of operators and/or a fitness function. Thus the EA class has no knowledge of the

Evolvable class, these objects cannot communicate directly and therefore an object is needed to connect them. This approach can be seen in Figure 3(b), where the contact between the EA and the individual is established by Evolutionary Operator and Fitness objects.

In this case the Evolvable objects may have several operators and fitness functions, but it is also possible to modify and evaluate them using other, not EA specific public methods. The disadvantage of this approach is that one has to implement small Evolutionary Operator and Fitness Function classes. However, operators have to be implemented anyway, so the only difference is that they are now in separate classes. The advantage is generality. The Evolvable subclass do not have to implement a certain number of operators. It can implement more, less, or even none. The only required methods are randomization, cloning and printing. The first two functions are needed because during the EA new individuals have to be constructed. Another advantage of this approach, is that it is also easier to parameterize the operators, since the Evolutionary Operator classes may have private fields to hold several parameters.

3.2 Population

The individuals make up the population, and it can be implemented as a vector, or as a set using standard arrays or the Standard Template Library. To construct a versatile evolutionary algorithm, it can be very useful to separate the population from the EA, and use an abstract class with a well defined interface. Thus the EA does not have to care what is inside of the population. It may be a simple array, or it may be in a file, in a database or even on a remote machine somewhere in the Internet. What the EA needs is the possibility to apply operators and selections on the population to create new populations. And of course inserting and removing individuals.

3.3 Operators

As mentioned previously, the operators are the link between the EA and the individuals. Furthermore, they can be organized into operator groups. The only functionality needed from operators and from operator groups is that they must be able to be applied on populations. That is operators and operator groups take populations and create new populations.

3.4 Selection

Selections are similar to operators. They also need a fitness function, according to which they filter the individuals. As it can be seen from the general EA model, the EA class does not have to know the fitness function either, only the selection method.

3.5 Algorithm

As discussed previously, the evolutionary algorithm itself knows the populations, an operator group and four selections. Later it will be easier and more efficient, to consider the populations created during an evolution loop as *subpopulations,* that is parts of one population. Thus, the EA object has exactly one Population, one Operator Group, and four Selections. The Population includes Evolvable objects, the Operator Groups are composed of other groups and Operators, and each selection has its own Fitness Function.

4 Implementation of the OOEA

Using the ideas and waypoints mentioned in the previous section, one can implement a general EA. It is not the purpose of this paper to give a detailed description of the implementation, but the important points are discussed in this section. The class diagram of the system can be seen in Figure 4. The classes and their most important functions are the following:

- **EA:** implements an evolutionary algorithm. Contains a Population, Operator Group and four Selections. Furthermore it has 5 parameters, which were given at the general EA model. Besides the functions to set the parameters and other components, it has a method to carry out one or more evolutionary steps.
- **Population:** contains Evolvable objects, and has several set-like operators. To be fast, only a pointer is given back, when an individual is requested, but it is constant for security. The Population has one or more subpopulations numbered from one, which can be used to store temporary populations, like the set of parents, or the set of offsprings.
- **Evolvable:** the individual in an EA. It can be cloned and printed, and it can generate a random individual.
- **SelectionMethod:** the most important method of this class is `apply`, which takes a Population, and inserts the selected individuals into a second Population. Usually it has a Fitness Function.
- **FitnessFunction:** it has a method called `get` that returns the fitness value of an individual or the fitness values for a whole population. There is a special descendant of this class called FunctionFitness. It stores a function pointer, and uses it for fitness evaluation.
- **OperatorGroup:** contains a tree of OperatorGroups and/or Evolutionary Operators. The method `apply_on` is the most commonly used function. It takes a population of parents, applies the group on it as described previously, and inserts the new individuals into another population.
- **Evolutionary Operator:** the evolutionary operator. Its method `apply` is the counterpart of the `apply_on` method for operator groups.

These classes can be divided into three groups. EA and OperatorGroup are *fixed classes* for any evolutionary algorithm, the user need not and can not modify them. The Population and the SelectionMethod classes are *customizable classes,*

they may be reimplemented, but it is not necessary, if the user does not need special subclasses. There are already several selection and population implementations, they can be used with many problems. Evolvable, EvolutionaryOperator and FitnessFunction are *problem specific classes,* thus in many cases they have to be implemented by the user. However, bitstrings and real vectors, and the appropriate operators are already implemented.

5 Examples

Using this general implementation many types of evolutionary algorithms can be implemented. Some examples are listed in the following.

- *ES with , or + strategy:* real vector implementation can be used, and in case of + strategy the copy selection can be used to insert the parents into the offspring population.
- *Traditional GA:* the bitstring is already implemented in the system with simple operators.
- *Island Model:* an evolvable population may be implemented as a subclass of classes Evolvable and Population, and the fitness evaluation can include an evolutionary algorithm.
- *Meta-ES:* similar to the Island Model
- *Parallel EA:* more populations and more EA objects with the same parameters may be created to implement this.
- *Distributed EA:* using a derivative of the Population object the individuals may be stored on different machines. It is also possible to distribute the individuals only for fitness evaluation, by implementing an appropriate `get` method.
- *Co-evolution:* the second `get` method of the fitness function takes the whole population, so it can be used to evaluate the individuals during a co-evolution.

5.1 Using the Library

When the representation form is already implemented, to use evolutionary algorithms is very easy. With 100-200 lines of C++ code even complicated problems can be solved. An example is detailed in the following.

1. Implement a fitness function

```
double fitness(const Evolvable &i){ /* fitness calculation */ }
```

2. Create a new population then a random individual, and insert this individual into the population

```
void test_ea(){
  ArrayPopulation *pop = new ArrayPopulation();
  ArrayPopulation results;
  EvolvableRealVector evreal(4);
  pop->insert(evreal.random());
```

3. Create a fitness function object

```
FunctionFitness *ff = new FunctionFitness();
ff->set_func(fitness);
```

4. Create the used selection methods, and set their fitness function, if necessary

```
SequentialSelection *copy = new SequentialSelection();
BestSelection *bs = new BestSelection(500);
FitPropSelection *fps = new FitPropSelection(500);
bs->set_fitness(*ff); fps->set_fitness(*ff);
```

5. Create the operator objects

```
EvOpRealMutate eo_mut;
EvOpRealRecombine eo_rec(false);
```

6. Combine the operator objects into an operator group

```
OperatorGroup og, rec, mut;
rec.set_op(&eo_rec); rec.set_prob(800);
mut.set_op(&eo_mut); mut.set_prob(500);
og.set_and(rec); og.add_group(mut); og.set_prob(1000);
```

7. Create an evolutionary algorithm by giving the parameters. Then set the population, the selections and the operator

```
EA *myEA=new EA();
myEA->set_params(1000,50,10,10,980);
myEA->set_population(pop);
myEA->set_psel(fps);
myEA->set_copy(bs); myEA->set_elitism(bs);
myEA->set_ssel(copy);
myEA->set_operator(&og);
```

8. Run the EA! After each step you can get individuals from the population, read their fitnesses, print them, examine halting criteria, and so on

```
for (int i=0; i<100; ++i){ myEA->step(); /* save, print */ }
```

9. At the end delete the objects

```
delete myEA; delete fps; delete bs; delete copy; delete pop;
}
```

5.2 Extending the Library

It is easy to extend the library, only the following points have to be followed.

– A subclass of class Evolvable has to be implemented with the `clone`, `random` and `print_on` functions. The class may contain special methods for evolutionary operators or for fitness evaluation.

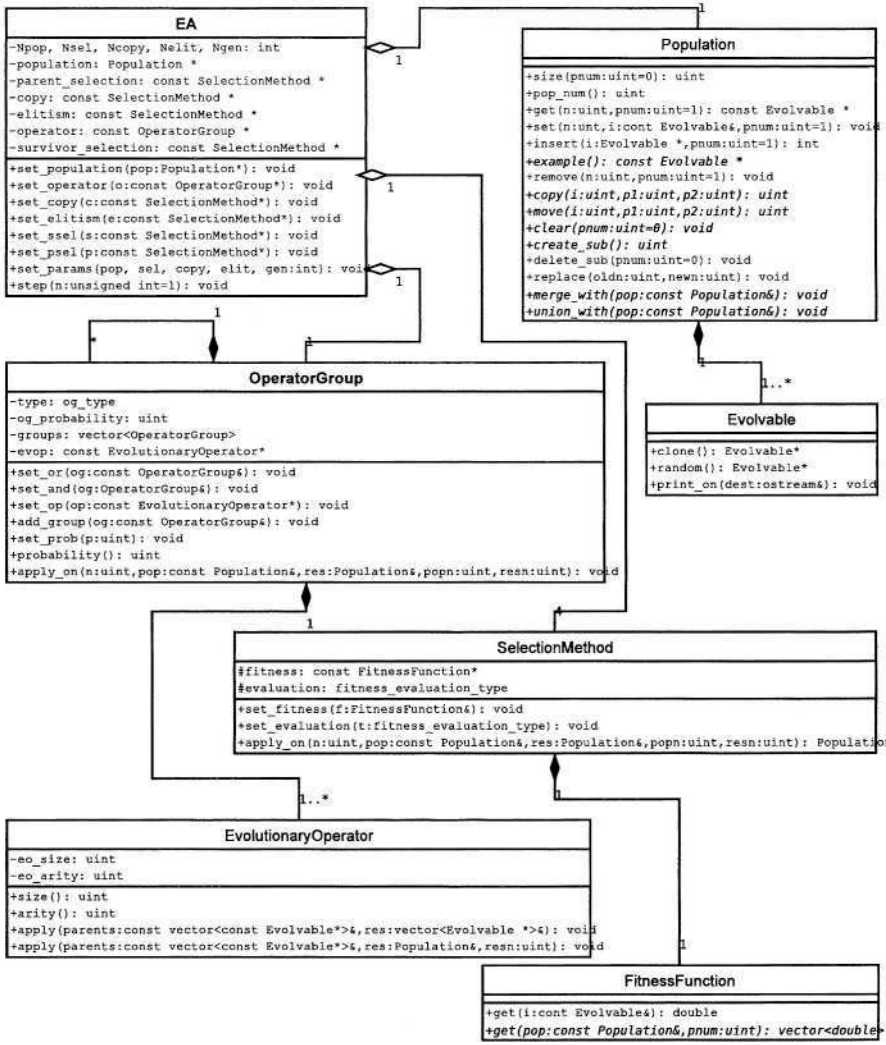


Fig. 4. UML class diagram of the general EA library

- The evolutionary operators have to be implemented. The most important function is `apply`, which can use the public methods of the Evolvable class.
- The fitness function has to be implemented. It can be either a subclass of FitnessFunction, or just a function as shown in the example.
- For special needs new selection methods may be implemented or the Population class may be overloaded.

6 Conclusion, Future Work

In this paper a general evolutionary algorithm was designed, and the steps of its implementation were shown. Using this general EA many special types of evolutionary algorithms can be implemented. The constructed system is simple and still versatile, easy to use and easy to extend. It was not among the aims of this paper to compare different programming libraries. However, it is planned to compare some of them with respect to CPU and memory usage.

Some features would be nice to be added to the library like a logging function, or error checking. It is also planned to change the randomization to be similar to the evolutionary operator, that is separated from the Evolvable class. When this is done, it may be considered to replace the inheritance based design to a template based one. This would mean even higher level of generality.

References

[eaL] eaLib – a Java Evolutionary Computation Toolkit. Technical report, Technical University Ilmenau, Germany. http://www.evolvica.org/ealib/index.html.

[Evo] Evolvica – Evolutionary Algorithms Framework in Java. Technical report, Technical University Ilmenau, Germany. http://www.evolvica.org/.

[Gol89] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Reading, MA, 1989.

[Gre84] J J Grefenstette. Genesis: A system for using genetic search procedures. In *Proceedings of a Conference on Intelligent Systems and Machines,* pages 161–165, 1984.

[Hol75] John H. Holland. *Adaption of Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, Michigan, 1975.

[Koz92] John R Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, Massachusetts, 1992.

[Mer] J J Merelo. Eo evolutionary computation framework. http://eodev.sourceforge.net/.

[Rec73] Ingo Rechenberg. *Evolutionsstrategien: Optimierung Technischer Systeme nach Prinzipen der Biologischen Evolution.* Fromman-Holzboog, Stuttgart, 1973.

[SGE94] R. Smith, D. Goldberg, and J. Earickson. Sga-c: A c-language implementation of a simple genetic algorithm, 1994.

[Tót00] Zoltán Tóth. *Generic Evolutionary Algorithms Programming Library.* Master's thesis, University of Szeged, Hungary, 2000.

[Wal96] M Wall. Galib: A C++ library of genetic algorithm components. Technical report, Massachusetts Institute of Technology, Mechanical Engineering Department, April 1996. http://lancet.mit.edu/ga/.

Generating Multiaxis Tool Paths for Die and Mold Making with Evolutionary Algorithms

Klaus Weinert and Marc Stautner

Department for Machining Technology Baroper Str. 301,
44227 Dortmund, Germany
{Weinert,Stautner}@isf.de
http://www.isf.de

Abstract. Defining tool paths for multiaxis milling is a difficult task that requires knowledge about the process and the behavior of the machine tool. Because of the high risks of multiaxis movements within a real machine, the resulting paths are safe but normally sub-optimal. This article introduces a new method to obtain tool paths, which are free of collisions and optimal in consideration, the machine and its kinematics. The proposed method combines evolutionary algorithms and a newly developed efficient multiaxis milling simulation. The evolutionary algorithm uses multiple criteria for the optimization of a tool path. These criteria are minimized axial movement of the machine, minimal cutting forces and the exclusion of collisions between the workpiece and the non-cutting parts of the cutting tool. The approach uses multi-objective algorithms because minimal or no movement of a machine axis can lead to high cutting forces or multiple collisions. Along with the evolutionary algorithm, this article introduces a simulation for multiple axis milling. This simulation is able to compute the fitness of the given tool paths by considering the engagement condition and collision situation of a cutting tool.

1 Problem Description

In multiaxis milling of free formed surfaces for die and mould making, one of the main tasks is a near optimal definition of the tool path. This definition is placed under various constraints, which need to be fulfilled simultaneously. One point is the avoidance of collisions between the non-cutting parts of the cutting tool and the work piece. A second one that is exclusive to multiaxis milling is the task of finding a tool orientation that fulfills all constraints resulting from the used machine. Different milling machines have different constraints imposed upon the movement of their additional axis, which are needed for the more than three axis movement. This can be seen as an additional collision condition between tool and possible movement area of the machine in addition to the collision between tool and free form surface.

The third constraint is not exclusively given in multi axis milling; it is also valid for all milling processes. In milling processes, the cutting force on the cutting edge is

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1287–1298, 2004.

an important factor for a stable and safe process control. The programming of tool paths must lead to a process that is fast and safe to be cost efficient.

Resulting from these constraints multiaxis tool path strategies are often designed as an addition to the existing three axis strategies. These paths are planned as a three-axis movement with an additional two-axis movement, which changes the orientation of the tool to the work piece. This additional rotational movement has to be free of collisions. Therefore, plenty of user experience is needed to design these tool paths. There are some software tools on the market that deliver some kind of collision detection but they are often restricted to three axis milling or only to collision detection against the final geometry and not the actual process geometry of the free form surface. So there is a demand for solutions which provide functions to develop tool paths which are free of collisions and use the possibilities of multi axis tool path programming to gain a fast and safe process course. This article introduces a first prototype of a tool that enables the user to convert given three axis tool paths to collision free multiaxis tool paths.

2 General Idea

The main concept consists of two steps. In the first step, a user starts designing a conversion solution for a given three-axis tool path and a given workpiece. In a second step, a software tool optimizes the user's parameters considering all constraints and optimizing facilities. After this second step, the resulting tool path leads to an efficient multi axis milling process, which is free of collisions and fulfills all machine restrictions.

The general idea for converting three axis milling paths into multiaxis paths, is to keep the position of the tool tip and to add a varying the orientation to the tool. The resulting surface remains the same when only ball end mills are used. Only the quality of the surface will improve resulting from the more advantageous engagement conditions. Several studies have shown that this simple tool tilt strategy leads to useful and efficient multi axis paths with minimal amount of manual user input.

2.1 Manual Step

The user determines a tilt point in 3D space. The multiaxis path is then generated by tilting the tool so that the main axis of the tool, shank and holder lead through an axis determined by the tilt point and the point on the tool path. This strategy is illustrated in Fig. 1. This tilt point strategy has several benefits: it is easy to develop and the resulting tool paths lead to smooth movements of the machine axis, because these movements can often be executed by tilting in one machine axis and then by rotating the second machine axis. One disadvantage is the indirect change of the tool paths by changing a point in 3D space. It is easy to find or see a useful position for this point, but it is difficult to determine that this point is the optimal one that leads to a minimal movement of the tool axis and is free of collisions.

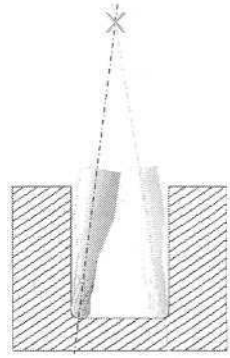


Fig. 1. Strategy for generating a tilt position for every cutter location by tilting through an axis given by a point in 3D space and the cutter position

2.2 Evolutionary Step

The optimization of this tilt point is done in the second step of the strategy by an evolutionary algorithm. The evolutionary strategy benefits from the factor that only a few (three) parameters need to be evolved for the tilt point. For further implementations of this technique the optimization of more then one tilt point is planned. Additionally the optimization of further parameters such as the form of the area in which the orientation is changed by one tilt point should be tested.

The difficulty in this evolutionary step lies not in the number of parameters. The main problem of this evolutionary algorithm is a fast computation of the fitness function. Therefore a new system for the simulation of five-axis milling has been developed. This simulation provides a tool to analyze the real process quality of the evolved individuals.

3 Techniques

The inputs in the process chain are a three axis-milling path with additional user parameters, these parameters are an area in which the path has to be tilted and an orientation of the tilting axis. After that, the evolution step searches for the optimal additional parameters, here the position of the tilt point and evaluates the fitness. The outputs of this process are optimized tilting positions and accordingly efficient multiaxis tool paths. The schematic view on this process chain is shown in Fig. 2.

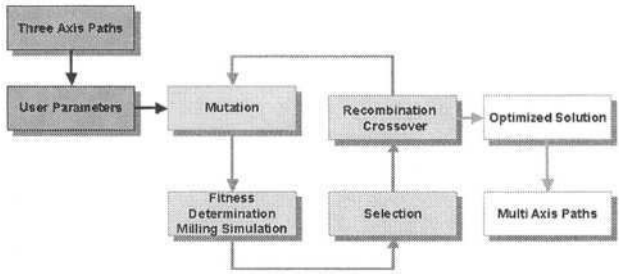


Fig. 2. Proposed two step process chain

3.1 Discrete Milling Simulation

Several products on the market allow a simulation of milling processes. Most of them [1] are not capable of simulating multiaxis milling processes. There are simulations that can perform this task but these techniques contain disadvantages, which prevent utilization for the fitness determination.

There is the group of exact simulation approaches. Kawashima [2] describes an accurate technique called "Grafttree" to speed up a solid modeling approach. The workpiece and the arbitrarily shaped tool are modeled by a CSG-tree. Constructive Solid Geometry (CSG) is an object-construction method used in several 3D-CAD systems. Objects are basically constructed by combining simple solid objects with Boolean operators, such as union or intersection as described by Foley et al. [**3**]. Sourin and Pasko [4] developed another approach; they simulate the cutting tool analytically with the use of procedurally defined time-dependent defining functions. Both approaches, the "Grafttree" and the one made by Sourin and Pasko are exact but time consuming.

A second group of techniques is the approximate approaches. One technique is the polygonal approach described by Glaeser [5]. Here a polygonal representation of the workpiece, called "Γ-buffer", was processed with a polygonal representation of the swept volume of the tool, which was determined by differential geometry. This approach can lead to large mesh sizes that are reduced for display by a recursive subdivision scheme.

Another group of approximate techniques is the "dexel" approach. Hook [6] converts the workpiece into a discrete "dexel structure" which is derived from the z-buffer approach in computer graphics [**3**]. One problem of Hook's approach is that, similar to the z-buffer algorithm, the dexel structure has to be aligned with a viewing vector. Therefore, after changing the viewing position, the whole simulation process has to be redone. A determination of fitness values through the simulation is not possible.

Extensions to this approach were made by Huang [7] and Weinert [8]. These extensions are independent of the viewer's position and process the input NC-file by modeling the cutting operation at each discrete position on a given path. To achieve

high accuracy with low memory usage, these discrete simulations do not have the ability to simulate undercuts, thus they are only able to simulate a three-axis milling process.

For the usage in these evolutionary algorithms a new simulation of milling processes based on these dexel techniques has been developed. This new simulation allows free scalable level of simulation resolution to reduce simulation time and provides a fast collision testing and extra interface functions to derive a fitness of simulated tool path. The simulation is able to detect and benchmark the amount of collisions and to take account of the restrictions resulting from the different mechanical capabilities of the used machine.

3.2 The Evolutionary Algorithm

As shown by Castelino et al. [9] evolutionary algorithms, which are used in process planning for nc-machining are primarily, specialized in the optimization of general tool movement. These solutions refer to algorithms developed for the traveling salesman problem or the lawnmower problem. Due to their large non-linear search spaces, both problems are often used for evolutionary algorithm. An overview on this subject is presented by Alander [10]. The algorithms in this area of problems do not focus on the machine abilities and the engagement conditions.

In our algorithm, the problem that needs to be solved is a simple three dimensional search. Therefore we can use a rather simple (μ, λ) evolutionary strategy ES. This leads to good results without optimized evolutionary parameters. This is a consequence of the simple three dimensional evolution problems, which have to be solved. A tournament selection operator defines two groups of individuals that compete with each other. Every survivor of a tournament replaces the inferior individual. A survivor of a tournament is the individual that has a higher fitness value compared to its competitor.

As is shown in the results section, simple problems can be solved in less than 20 generations with values for μ of 50 and a λ of 25. This behavior is important the usefulness of the introduced system. The determination of the fitness of a single individual can be very time consuming. Milling processes have real running times of up to 20 hours. State of the art milling simulations can be 10 times faster than the real process. So defining a fitness of a single individual phenotype can be very time consuming.

3.2.1 Fitness Determination

The fitness function uses three different types of fitnesses that were determined separately and result from different constraints and collisions.

1. Collision with the space bound by not reachable axis values of the used machine.
2. Collisions of the workpiece with the non-cutting parts of the tool such as shank or holder. All collisions in a simulation run are added up to a single value. This value shows how much volume of the material collides with the tool. Therefore,

slight touching collisions result in different values as full collisions. This is a measurement for the amount of collisions with the machine.

3. The third value is the amount of harmony in the resulting movement of the machine. The simulation returns a value that represents how easy the machine may follow the suggested movement. This value has a direct influence on the real process milling times and the resulting surface quality.

These three values represent multiple criteria for the fitness determination. A combination is used for the selection of the individuals.

3.2.2 Selection Method

The algorithm uses a truncation or (μ, λ) selection [11]. The selection schema for the λ best individuals is in pseudo code.

```
for (c=0;c<[NumberOfFitnessValues];c++){
    if (I1.f[c] > I2.f[c]) return true;
    if (I1.f[c] < I2.f[c]) return false;};
```

Where I1 and I2 are the current individuals with their respective arrays of fitness values. In our algorithm, there are three different fitness values. The ranking depends on the order of the fitness values in the array. In our algorithm the order is

1. machine constraints,
2. amount of collisions and
3. movement harmony.

Therefore, at first a tool path needs to be free of both types of collisions and after that the algorithm searches for the position that leads to the most harmonic movement of the machine axis.

3.2.3 Recombination and Crossover

The Recombination is done via a two individual crossover. Two individuals are randomly chosen from λ selected ones (see Sect. 3.2.2) ignoring their fitnesses. The crossover randomly chooses values from both individuals. This crossover reduces the fitness pressure so that searches in the solution space are more efficiently.

4 First Results

First tests were carried out on specially designed workpieces, which allow a fast and simple design of the evolutionary algorithm. The use of these test cases allows verifying the design of the EA and helps to reduce unwanted time losses from errors in the implementation of the algorithms. The test workpieces are designed with the CAD/CAM software CATIA to ensure the functionality of the complete process chain.

4.1 Test Workpieces

The tests should verify the algorithm on both opposed possibilities that occur in real workpieces. Therefore, the first workpiece is a deep cavity. This is a typical form in die and mould making. Cavities are difficult to mill if they are of a certain depth and a small diameter, because this necessitates tools with long thin tool shanks. The use of long shanks causes undesirable effects on the surface quality and increases the high risk of tool breakage. To overcome these effects the process speeds need to be reduced. The second workpiece is a dome that has a steep rising edge in the middle of the workpieces Fig. 3. (right). This workpiece needs to be milled with long shank as well to enable the processing of the small diameter (1 mm) fillet at the base of the dome.

Although both test workpieces have their ideal tool tilt point in the center of the workpieces they have different constraints. The collision spaces are completely different so that the collision avoidance leads to different demands on the evolution progression. The simulation can only deliver a value of how many collisions have occurred. As a consequence the fitness landscapes are different, and a parameterization which is suitable for both cases should be suitable for real workpieces as well.

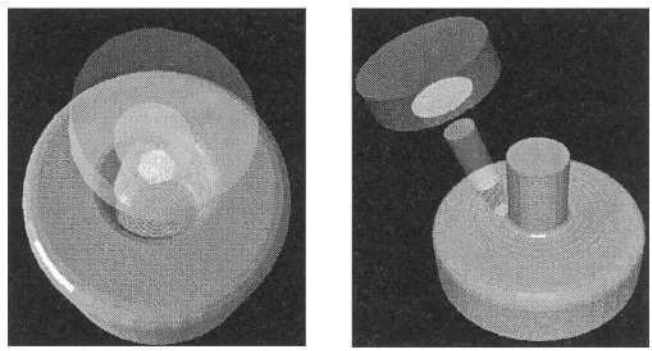


Fig. 3. Test workpieces and their tooling paths. Negative Cavity (left), Positive Dome (right)

4.1.1 Tool-Shank-Holder

The employed, simulated tool is a ball end mill with a diameter of 2 mm and an edge length of 1 mm. This part of the tool cannot lead to collisions. To ensure an increased surface quality after the evolution a shorter, thicker tool shank was use for the evolution. This tool cannot be used on the original three axis paths. The colliding part of the tool is the shank, which has a cone as basic form and the holder. In our test workpiece, only collisions between workpiece and shank are to be expected because of the length of the shank.

4.1.2. The Cavity Workpiece

The initial tool paths for the cavity are some short finishing paths on the ground of the cavity (see Fig. 3). At this point of the milling process, the chance of a collision is most likely. The milling of these paths is not possible with a three-axis strategy and the chosen shank-holder combination.

The initial search space for the positioning of the tool points is from -40 mm to +40 mm along all axes of the global object space. There are 50 initial individuals that are spread stochastically over this 80x80x80 mm search space. To obtain an overview of the behavior of the algorithm this test run is repeated 20 times.

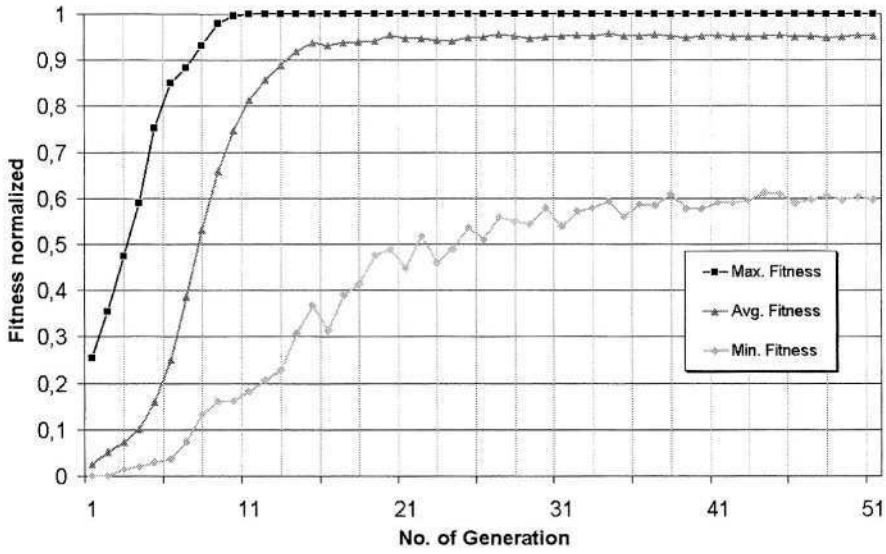


Fig. 4. Average fitnesses from 20 evolution runs with 50 individuals in each population. (Cavity Workpiece)

The diagrammed fitness in Fig. 4. is the accumulated, normalized fitness of both collision constraints. As can be seen, the maximal fitness reaches 1 in approx. 10 generations. This shows that collision free tilt points can be reached and there is enough time for a more difficult search for the optimal tool point.

4.1.3 The Dome Workpiece

The tool path on the second workpiece is different to the path introduced before. The collision here is most likely at the inner paths where edge of the dome is approached, see Fig. 5. Because of the similar dimensions of the workpiece all parameters remain the same. This allows obtaining an overview of the behavior of the algorithm on different parts without the use of extra knowledge.

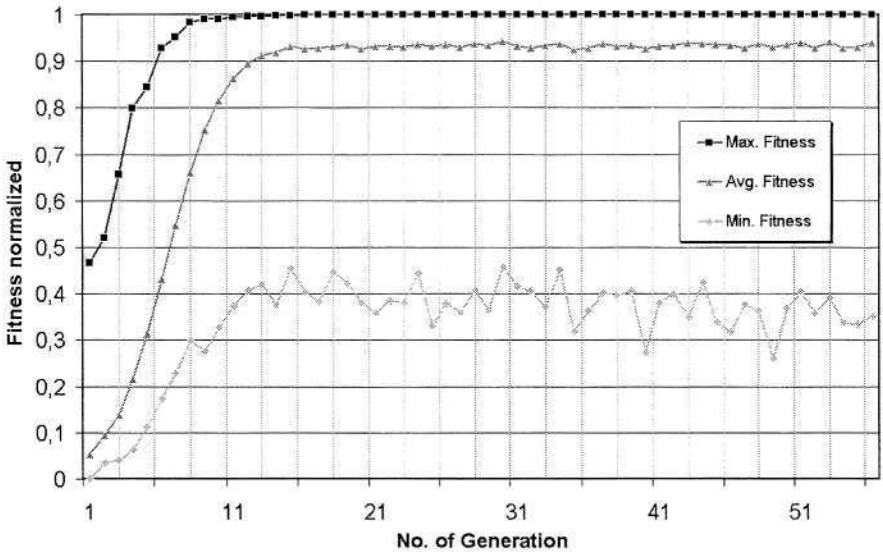


Fig. 5. Average fitnesses from 20 evolution runs with 50 individuals each generation. (Dome Workpiece)

As it can be seen in Fig. 5. the general behavior of the algorithm is similar a collision free position is found in approx 10 generations. After that the fitness is determined by the harmony fitness value.

4.2 Real Workpiece

The pre- parameterized algorithm is now used for a real workpiece. The chosen workpiece and the three axis milling paths are not specially designed for five-axis milling or for this evolutionary path design. The workpiece originates from the European High Speed Machining Award 2000. To preclude a three axis processing a shank based on a cone is used. Surface quality is expected to be better if the process is carried out with the shorter and sturdier conical shank, the present three-axis tool paths will lead to a collision if they are used with this tool, though.

As a test-processing step, a finishing step was chosen. This finishing step consists of approx. 65000 single lines of nc-code. As in the test runs, 50 individuals per population are used. The initial search space for these tests is 200mm x 200mm x 200mm according to the larger dimensions of the workpiece (160mm x 160mm x 40mm). The tool paths can be seen in Fig. 6.

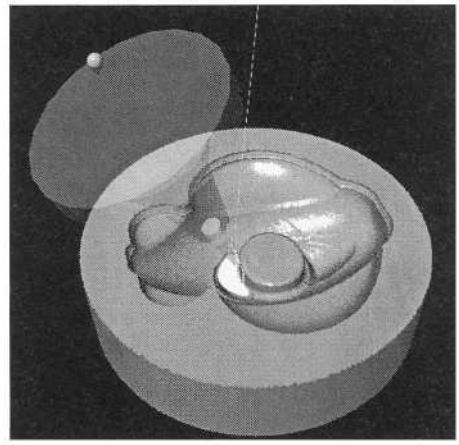


Fig. 6. Workpiece, from the European High Speed Machining Award 2000

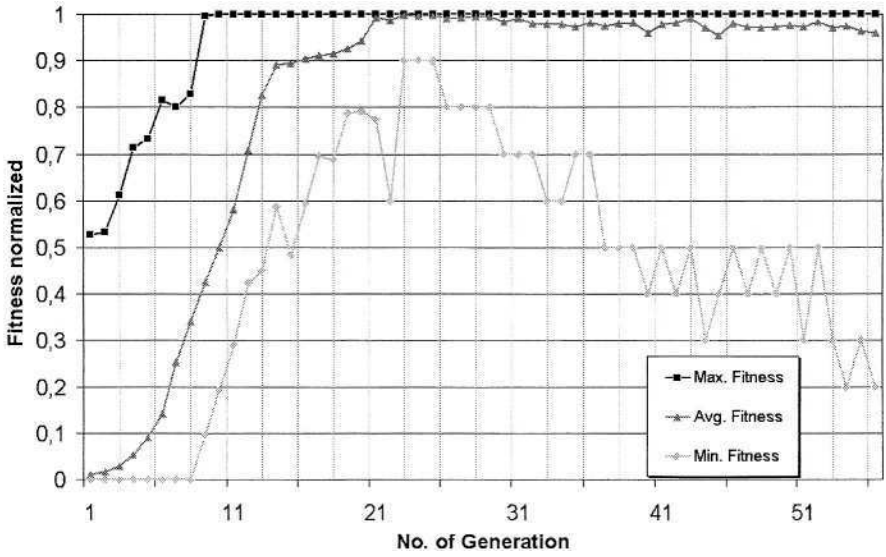


Fig. 7. Average fitnesses from 10 evolution runs with 50 individuals each generation

Fig. 7. shows a fitness plot for 10 subsequent test runs. Each run consist of approx. 50 generations with 50 individuals in each generation. In this plot only the collision (workpiece tool and tool machine space) fitnesses are combined. The path of the first curve which is the average maximal fitness of an individual shows that a collision free position is found by the algorithm on average within the first 10 generations of the evolution. Until generation 20 the other values for average fitness and for minimal

fitness increase accordingly. After generation 20 the average minimal fitness decreases. The reason for this behavior lies in the machine constraints: the collision free tilt point is close to the edge of the region reachable by the machine. The algorithm draws all individuals close to this border and therefore a minimal movement of the tool point leads to tool paths that cannot be realized by the machine kinematics. The fitness of these tool points is set to zero. Therefore, the more points reach the optimal point that is close to this edge the more points can accidentally jump over. This is the reason for the slightly decreasing average fitness.

5 Conclusions and Outlook

This paper presents a new system for the optimization of tool paths and engagement conditions for multiaxis milling as well as the generation of efficient multiaxis tool paths. First studies show that the proposed multi criteria evolutionary algorithm is suitable to deliver an efficient evolution in combination with an efficient simulation system. The simple parameterization of the algorithm allows the application of the system on different problem cases without further user knowledge about the evolutionary parameters. Although the introduced software system is an academic prototype, this qualifies enables this technique as basis for a commercial solution for multiaxis path generation in CAM applications.

Further implementations are planned based on other generation strategies and require a more complex genome. More advanced multiaxis strategies may be developed with this system without the use of three axis tool paths as basis for the evolution.

Acknowledgements. The authors like to thank the Deutsche Forschungsgemeinschaft for financial support as part of the Collaborative Research Center "Computational Intelligence" (SFB 531).

References

1. Müller, H.; Surmann, T; Stautner, M; Albersmann, F.; Weinert, K.: Online Sculpting and Visualization of Multi-Dexel Volumes. In: SM'03, Eighth ACM Symposium on Solid Modeling and Applications, Elber, G.; Shapiro, V. (eds.), Seattle, Washington, USA, ACM 1-58113-706-0/03/0006, ACM Press, NY, USA, June 16-20, S. 258-261
2. Kawashima, Y; Itoh, K.; Ishida, T.; Nonaka, S; Ejiri, K.: A flexible quantitative method for NC machining verification using a space-division based solid model. The Visual Computer7 (1991), pp. 149-157
3. Foley, J. D.; Dam, A. van; Feiner, S. K.; Hughes, J.F.: Computer Graphics: Principles and Practice. Addison-Wesley, New-York 1992
4. Sourin, A.I.; Pasko A.A.: Function Representation for Sweeping by a Moving Solid. IEEE Transactions on Visualization and Computer Graphics 2 (1996) 2, pp. 11-18

5. Glaeser, G.; Gröller, E.: Efficient Volume-Generation During the Simulation of NC-Milling. Technical Report TR-186-2-97-10, April 1997, Institute of Computer Graphics and Algorithms, Vienna University of Technology
6. Hook, T. van: Real Time shaded NC Milling Display. Computer Graphics 20 (1986) 4, pp. 15-20
7. Huang, Y.; Oliver, J.H.: NC Milling error Assessment and Tool Path Correction. In: Computer Graphics Proceedings, Conference Proceedings July 19-24 (1994), pp. 287-294
8. Weinert, K.; Müller, H.; Friedhoff, J.: Efficient discrete simulation of 3-axis milling for sculptured surfaces. Production Engineering 3 (1998) 2, pp. 83-88.
9. Castelino, K.; D'Souza, R.;Wright, P. K.: Tool-path Optimization for Minimizing Airtime during Machinig, Mechanical Engineering Dept. University of California at Berkeley, http://kingkong.me.berkeley.edu/~kenneth/research/pubs/ airtime_minimization _joms .pdf
10. Alander, J., T.: An Indexed Bibliography of Genetic Algorithms and the Traveling Salesman Problem. Internal Report 94-1-TSP, University of Vaasa, Department of Information Technology and Production Economics, Finland
11. Banzhaf, W; Nordin, P; Keller, R. E.; Francone, F. D.: Genetic Programming : An Introduction : On the Automatic Evolution of Computer Programs and Its Applications. Morgan Kaufmann Publishers, Inc. San Francisco, Cal.

Tackling an Inverse Problem from the Petroleum Industry with a Genetic Algorithm for Sampling

Pedro J. Ballester and Jonathan N. Carter

Imperial College London, Department of Earth Science and Engineering,
RSM Building, Exhibition Road, London SW7 2AZ, UK
{p.ballester,j.n.carter}@imperial.ac.uk

Abstract. When direct measurement of model parameters is not possible, these need to be inferred indirectly from calibration data. To solve this inverse problem, an algorithm that preferentially samples all regions of the parameter space that fit data well is needed.
In this paper, we apply a real-parameter Genetic Algorithm (GA) to sample the parameter space for the inverse problem of calibrating a petroleum reservoir model. This results in several important insights into this nonlinear inverse problem.

1 Introduction

Because there is no available analytical expression between model parameters and measured data, numerical models are needed in Petroleum Engineering. The location of petroleum reservoirs (thousands of metres below ground) and their extension (typically several kilometres) make direct measurements of the model parameters mostly impossible. Only few of these parameters can be estimated from measurements at wells drilled into the reservoir. In this context, indirect measurements usually take the form of fluid production historical data and thus the inversion is called history matching. This can be posed as a search and optimisation problem by defining an objective function quantifying the mismatch between the model output and the measured production data.

In this work we consider a cross-sectional model of a layered reservoir (an extensive description of the model can be found in [1]). In order to quantify the match between the model response and the measurements, we define first an objective function for the history matching period, Δ_m, as follows

$$\Delta_m(\mathbf{m}) = \frac{1}{N_s} \sum_{j=1}^{N_s} \sum_{k=1}^{3} \frac{|S_{jk}(\mathbf{m}) - O_{jk}|}{2\sigma_{jk}} \quad . \tag{1}$$

where N_s is the number of time steps and determines the extent of the history matching period, $\mathbf{m} = (h, k_p, k_g)$ is the considered model, $S_{jk}(\mathbf{m})$ is its simulated response for the production series k at time step j, O_{jk} is the corresponding objective ('measured') value and σ_{jk}, an estimation of what would be the associated measurement error. We assume it as a 3% of the measured value, ie. $\sigma_{jk} = 0.03\ O_{jk}$. The response for the truth model is simulated to provide the measurements as $O_{jk} \equiv S_{jk}(h^0, k_p^0, k_g^0)$. The objective function for the prediction period, Δ_f, has an analogous expression.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1299–1300, 2004.

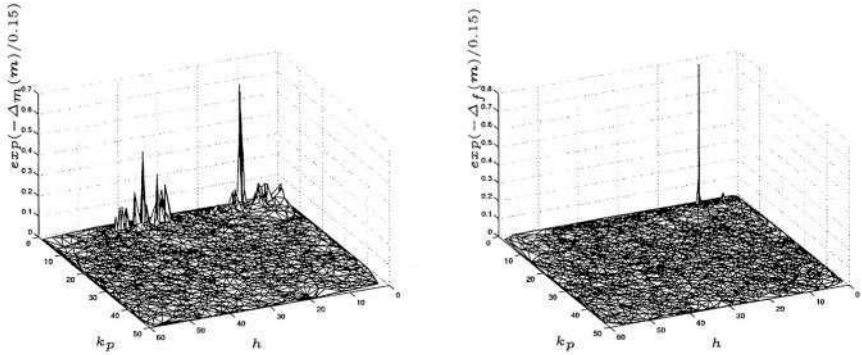


Fig. 1. GA for Sampling using $N_s = 36$ on: a) Δ_m with Δ_m^{best}(10.37,1.425,131.50) = 0.0795 and b) Δ_f with Δ_f^{best}(10.40, 1.309, 131.83) = 0.0373. The truth model was set as $h^0 = 10.4$, $k_p^0 = 1.31$ and $k_g^0 = 131.7$

2 Results and Discussion

The chosen search method is a Steady-state Real-parameter Genetic Algorithm. It combines the following features: parental selection is not fitness biased, a self-adaptative crossover operator, implicit elitism and locally scaled probabilistic replacement. Details of this GA can be found in [2]. Here we introduce an additional feature: children can only be bred within the initialisation region. The GA is used with N=50, $\lambda = 1$, NREP= 10 and $\eta = 0.01$, using a total of 7,050 evaluations. The output of the algorithm will be the ensemble of all the individuals that entered the population during the GA run (typically a third of the total number of evaluations). In figure 1a, the very large spike, with $h \approx 10$, corresponds to the truth model. We can also see notable local optima in the regions with $0 < h < 8$ and $31 < h < 44$. Figure 1b shows the result of carrying out the inversion using the objective function for the prediction period, Δ_f. The only substantial point found corresponds to the truth model. All the other local optima that can be seen in Fig. 1a are unable to match the observations during the prediction period. We conclude that for this model you can only obtain a good prediction from the truth case, and that other good matches from the history matching phase have no predictive value. It is worth noting that the best found models in Fig. 1 are very similar to the truth model.

References

1. Ballester, P.J.: The use of Genetic Algorithms to Improve Reservoir Characterisation. PhD thesis, Department of Earth Science and Engineering, Imperial College London (2004)
2. Ballester, P.J., Carter, J.N.: An effective real-parameter genetic algorithms for multimodal optimization. In Parmee, I.C., ed.: Proceedings of the Adaptive Computing in Design and Manufacture VI. (2004) In Press.

A Genetic Approach for Generating Good Linear Block Error-Correcting Codes

Alan Barbieri, Stefano Cagnoni, and Giulio Colavolpe

Università di Parma
Dipartimento di Ingegneria dell'Informazione
Parco Area delle Scienze 181/A, 43100 Parma - ITALY
http://www.dii.unipr.it/

Abstract. In this paper we describe a method, based on a genetic algorithm, for generating good (in terms of minimum distance) linear block error-correcting codes. Preliminary experimental results indicate that the method can be very effective, especially in terms of fast generation of good sub-optimal codes.

The search space for the problem of finding good codes is too large for many of the standard search algorithms. In such cases, genetic algorithms can be very effective in finding good solutions in a relatively short time. A genetic approach to code generation was followed in [1]. In this case, however, the target was to find the entire codebook. That requirement makes such an approach useful, in practice, only for codes whose parameters are no larger than few units. In [2] a table of minimum-distance bounds for binary linear codes has been presented. These bounds may be used as references in evaluating the effectiveness of new search algorithms.

A binary error-correcting code can be used in digital transmission to reduce the probability of word errors. The coding operation consists in the transformation of a k-bits word to a n-bits one, where $n > k$. This transformation defines a codebook $\mathcal{C}$ of 2^k n-bits codewords, taken from a set of 2^n possible words. An important parameter which determines the asymptotic performance of a block code for low error probabilities is the code *minimum distance*. It is defined as the Hamming distance between the two nearest codewords, i.e.,

$$d_{min} = \min\{d(\mathbf{c}_i, \mathbf{c}_j) | \mathbf{c}_i \neq \mathbf{c}_j \in \mathcal{C}\}$$

where $d(\mathbf{c}_i, \mathbf{c}_j)$ represents the Hamming distance between codewords $\mathbf{c}_i$ and $\mathbf{c}_j$ and is defined as the number of elements by which the two codewords differ.

The algorithm we propose is aimed at finding good codes, i.e., codes with given k and n, and maximum d_{min}. In our implementation, we co-evolve more than one fixed-size population at one time. We use a classical genetic algorithm, with some adaptations to the problem under consideration. In particular, before applying crossover, the columns of every individual are sorted from best to worst, thus ensuring higher probability of a fitness increase. At the end of every iteration, the degree of similarity between individuals in each population is checked. If it is too high, a population crossover is made, i.e. the degenerated population swaps some individuals with another randomly-chosen one. Fitness is evaluated

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1301–1302, 2004.

so that the integer part is the minimum distance, while the fractional part is related with the distance spectrum. Exact evaluation of d_{min} and of the distance spectrum is a computationally demanding problem: in [3] authors prove that it is impossible to find algorithms that approximate d_{min} in polynomial time. For this reason, fitness evaluation is the most time-consuming task of our genetic algorithm.

Some simulation results are reported in Table 1. We have used two different kinds of code: (34,15,9) (using populations of 100 individuals) and (21,11,6) (1000 individuals for each population), where the notation (n, k, d^*_{min}) indicates a $n \times k$ code with upper bound d^*_{min} on the minimum distance (see [2]). In the first case the best codes found have $d_{min} = 8$, while for a (34,15) code $d^*_{min} = 9$. However, it should be also noticed that only very few codes with $d_{min} = 9$ have been reported in literature, and that the algorithm has found several different codes with $d_{min} = 8$ in relatively few iterations. Instead in the second case the algorithm has been able to reach the theoretical bound on the minimum distance ($d_{min} = 6$) within the first 172 iterations, in all experiments but one.

Table 1. Convergence and best code found in each run for the (34,15) and the (21,11) codes: n is the experiment, a the iteration in which the first code with d_{min}=8 and 6 respectively has been found and b the iteration in which the best-fitness code has been found. The last column reports the number of codes with best fitness that have been found in each run.

n	a	b	best fitness	codes found
1	35	730	8.999603	1
2	28	463	8.999573	1
3	5	1309	8.999359	1
4	28	403	8.999664	2
5	34	2039	8.999481	1
6	27	295	8.999573	1
7	42	304	8.999451	2
8	22	1716	8.999237	1
9	25	250	8.999359	2
10	19	143	8.999390	1

n	a	b	best fitness	codes found
1	39	104	6.936035	1
2	62	62	6.935547	1
3	34	34	6.933594	1
4	N.A.	55	5.995583	1
5	172	172	6.935059	1

In conclusion, this approach is able to produce good sub-optimal codes in very few iterations. Future developments of our research will be aimed, in first place, at improving effectiveness of the search strategy by further adapting the genetic operators to the challenges posed by the problem under consideration.

References

1. Dontas, K., Jong, K.D.: Discovery of maximal distance codes using genetic algorithms. In: Proceedings of the 2nd International IEEE Conference on tools for Artificial Intelligence. (1990) 805–811
2. Brouwer, A.E., Verhoeff, T.: An updated table of minimum-distance bounds for binary linear codes. IEEE Transactions on Information Theory **39** (1993) 662–677
3. Dumer, I., Micciancio, D., Sudan, M.: Hardness of approximating the minimum distance of a linear code. IEEE Transactions on Information Theory **49** (2003) 22–37

Genetic Fuzzy Discretization for Classification Problems*

Yoon-Seok Choi and Byung-Ro Moon

School of Computer Science & Engineering, Seoul National University
Shillim-dong, Gwanak-gu, Seoul, 151-742, Korea
{yschoi, moon}@soar.snu.ac.kr

Many real-world classification algorithms can not be applied unless the continuous attributes are discretized and the interval discretization methods are used in many machine learning techniques. It is hard to determine the intervals for the discretization of numerical attributes that has an infinite number of candidates. And interval discretization methods are based on a crisp set, a value in a continuous attribute must belong to only one interval. They are often not proper for describing a value located around the boundaries of intervals. Fuzzy partioning is an attractive method for those cases in classification problems. An important decision in fuzzy partitioning is about the positions of interval boundaries and the degrees of overlapping in the fuzzy sets. We optimize the parameters that specify fuzzy partitioning by genetic algorithms.

We divide the range of a continuous attribute into k intervals and represent each value by a k-bit string where each bit corresponds to one interval. The i^{th} bit of the binary string represents whether the value belongs to the i^{th} interval or not. While a value belongs to only one interval in a simple discretization, it can belong to more than one interval in fuzzy discretization. Thus a value can be represented by a binary mask. For example, in Fig. 1, the value 0.595 belongs to the third and fourth intervals and is represented by a binary mask 00110. It provides more flexibility in machine learning algorithms for pattern classification. We optimize the boundaries of intervals and the degrees of overlapping in fuzzy discretization. We use four parameters for each interval I_i: t_i, t_{i+1}, l_i and u_i. The genetic fuzzy membership function is defined as follows:

$$G_0(a) = \begin{cases} 1, \text{ if } \quad v_{min} \le a < u_0, \\ 0, \text{ otherwise.} \end{cases}$$

$$G_i(a) = \begin{cases} 1, \text{ if } \quad l_i \le a < u_i, \quad i = 1, \ldots, k-2, \\ 0, \text{ otherwise.} \end{cases}$$

$$G_{k-1}(a) = \begin{cases} 1, \text{ if } \quad l_{k-1} \le a \le v_{max}, \\ 0, \text{ otherwise.} \end{cases}$$

where $l_i \le t_i$, $i = 1, \ldots, k-1$, and $t_{i+1} \le u_i$, $i = 0, \ldots, k-2$. In the above, t_i and t_{i+1} indicate the "base" boundary points of interval i. l_i and u_i indicates the left and right boundaries of the interval i, respectively. t_i, l_i and u_i are determined by genetic optimization. Fig.1 shows an example of discretization results.

* This work was supported by Brain Korea 21 Project. The ICT at Seoul National University provided research facilities for this study.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1303–1304, 2004.

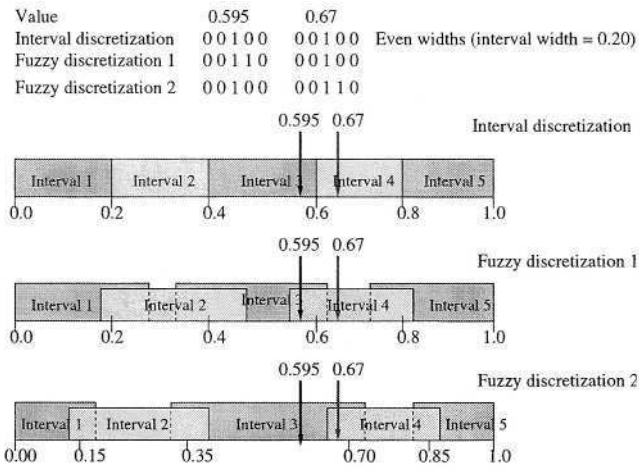


Fig. 1. Three different example discretizations and the corresponding binary masks

We used two well known datasets from the UCI Machine Learning Database Repository[1] to measure the performance of the proposed approach. The domain of each attribute is discretized into five intervals. For robust comparison, we applied the n-fold cross-validation. In order to evaluate the generated data sets, we use artificial neural networks and C4.5 that is a typical top-down method for inducing decision trees. From the simulation results, we can observe that the GAFD improved the classification performance over ID and FD. "ID" indicates the conventional discretization method and "FD" indicates a conventional fuzzy discretization with a pre-specified overlap degree. "GAFD" indicates the suggested genetic fuzzy discretization that evolves the widths of intervals and the degrees of overlap. Considering the group standard deviation($\sigma/\sqrt{n}$) in the case of ANN, we can say that GAFD was better than the others with a confidence level higher than 99% (Table 1).

Table 1. Result on the dataset

Database	WDBC†			WDG‡		
Method	ANN (n=1000)		C4.5	ANN (n=1000)		C4.5
	Avg(%)	$\sigma/\sqrt{n}$	Accuracy(%)	Avg(%)	$\sigma/\sqrt{n}$	Accuracy(%)
ID	94.08912	0.029414	90.32864	81.21342	0.025860	73.66527
FD	95.38696	0.028718	92.26217	81.79459	0.022058	73.56529
GAFD	**96.38463**	0.033088	**93.14554**	**83.11773**	0.024903	**73.98520**

† Wisconsin Diagnostic Breast Cancer.
‡ Waveform Database Generator.

[1] http://www.ics.uci.edu/~mlearn/MLRepository.html

A Genetic Algorithm for the Shortest Common Superstring Problem

Luis C. González, Heidi J. Romero, and Carlos A. Brizuela

Computer Science Department, CICESE Research Center
Km 107 Carr. Tijuana-Ensenada, Ensenada, B.C., México
+52–646–1750500
{gurrola, hjromero, cbrizuel}@cicese.mx

Many real world problems can be modeled as the shortest common superstring problem. This problem has several important applications in areas such as DNA sequencing and data compression.

The shortest common superstring problem (SCS) can be formulated as follows. Given a set of strings $\mathbf{S} = \{s_1, s_2, ..., s_n\}$ the goal is to find the shortest string N^* such that each $s_i \in \mathbf{S}$ is a string of N^*. Finding the SCS of a set of strings is known to be NP-hard [2].

The shortest common superstring problem can be modeled as the asymmetric TSP (ATSP). The optimum tour cost for the ATSP problem represents a lower bound for the optimum of the SCS N^* of $\mathbf{S}$. Unfortunately, the ATSP is also an NP-hard problem. Fortunately, a lower bound for this problem is the well known Held-Karp bound [4]. We will use this bound to study the solution quality produced by our algorithm. We propose an algorithm which is based on a recently proposed algorithm [1] for the sequencing by hybridization (SBH) problem. Each individual is represented by a permutation of indices of substrings in $\mathbf{S}$. Specifically, the adjacency-based coding is used. The fitness of each individual is given by the maximum number k of overlapping characters the individual represents, taking into account all substrings in the chromosome. For each individual, its fitness value is normalized considering the maximum possible overlap. The individuals are selected according to the stochastic remainder method without replacement [3]. The population for the next generation is constructed based on the already selected individuals, which are randomly paired to undergo crossover with $p_c = 1$, always maintaining the best individual found (elitism).

In order to test our algorithm we have decided to deal with the SBH, motivated by the availability of many benchmarks for this problem coming from the real world. All sets $\mathbf{S}$ used in the experiment have been derived from DNA sequences coding human proteins (taken from GenBank, National Institute of Health, USA).

Table 1 presents the approximation ratio between the solutions generated by the GA (SCS GA) and their corresponding Held-Karp (HK) bounds. Column 1 presents the number of instances. Column 2 presents the size of the instances. Column 3 presents average approximation ratios to the HK bound, and column 4 presents their standard deviation. Column 5 presents approximation ratios of the best results to the HK bound and column 6 presents their standard deviation.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1305–1306, 2004.

It would also be interesting to see how similar the solutions are to the original DNA sequences, from where the sets of substrings were derived. For this purpose, we compared the solutions generated by the GA with the original sequences using a pairwise alignment algorithm. Column 7 presents how similar the solutions generated by the GA and the original sequences are; if both sequences are identical, then the similarity score is 100%.

Table 1. Comparison results of the SCS GA and the Held-Karp lower bound over 30 runs

No. of Instances	|S|	SCS GA (%)	S.D.	Best (%)	S.D.	Similarity score (%)
10	100	5.48	2.7	1.20	1.97	98.95%
10	200	3.83	1.34	0.81	1.08	96.48%
10	300	3.38	1.07	1.22	1.28	97.16%
10	400	2.58	0.84	0.82	0.59	96.46%
10	500	2.6	0.86	1.21	0.58	87.85%
10	600	2.33	0.86	1.06	0.63	83.39%
10	700	2.49	0.99	1.50	0.8	70.58%
10	800	2.35	0.81	1.44	0.64	70.72%
10	900	2.02	0.78	1.11	0.49	69.50%
10	1000	2.05	0.67	1.26	0.5	57.18%
10	2000	2.12	0.73	1.28	0.5	31.22%
10	3000	3.22	1.15	2.18	0.9	29.20%

The best solution lengths (except for the case of $|\mathbf{S}| = 3000$) exceeds the HK bound in no more than 1.5%. Furthermore, if we consider that this algorithm was designed for the SCS problem, the similarity percentage score for the first six sets of instances, where $100 \leq |\mathbf{S}| \leq 600$, can be considered as motivating, and this point is of special interest given that for real hybridization experiments $100 \leq |\mathbf{S}| \leq 500$.

Relative errors less than 6% from the optimum tell us about the suitability of this algorithm for real world applications.

References

1. C. Brizuela, L. González, and H. Romero. An Improved Genetic Algorithm for the Sequencing by Hybridization Problem. In *(to appear) Proceedings of the 2nd European Workshop on Evolutionary Bioinformatics, EvoBIO 2004.*
2. M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman, 1979.
3. D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989.
4. M. Held and R. Karp. The Traveling Salesman Problem and Minimum Spanning Trees. *Operations Research,* 18:1138–1162, 1970.

A Genetic Algorithm to Improve Agent-Oriented Natural Language Interpreters

Babak Hodjat[1], Junichi Ito[1], and Makoto Amamiya[2]

[1] Dejima Inc., USA, www.dejima.com
{Babak, junichi.ito}@dejima.com
[2] Kyushu University, Japan
amamiya@is.kyushu-u.ac.jp

Abstract. A genetic algorithm is used to improve the success-rate of an AAOSA-based application. Tests show promising results both in the improvement made in the success-rate of the development and test corpora, and in the nature and number of interpretation rules added to agents.

Keywords: Agent-Oriented Software Engineering, Evolutionary Optimization, GA, Natural Language Interfaces.

1 Introduction

Adaptive Agent Oriented Software Architecture (AAOSA) [1,2] is designed for AOSE Methodology [3,4]. AAOSA is used for Natural Language Interfaces (NLI), where agents represent semantic subdomains of the system (see fig 1). Policies in agents map conditions on input or context of system to claims and actions (see list 1). Policies may need revisions as samples are collected or when the domain of the application is changed or extended.

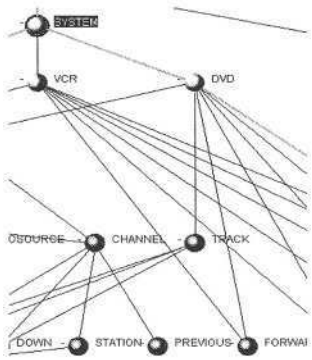


Fig. 1. A cross section of an AAOSA-based NLI for an A/V system.

2 Method, Test, and Conclusions

The GA takes the AAOSA Agent network as the gene to be evolved. Environment is a corpus of sample entries to the NLI system, including desired output actuations.
Mutation adds or removes parts of a policy in an agent. Additions are extracted from failed cases in corpus. Added conditions may be removed through mutation. The existing actions defined in the agent networks are reused in mutated policies.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1307–1309, 2004.

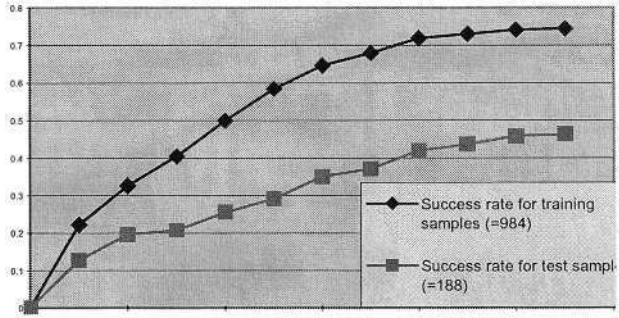


Fig. 2. Plot of success-rate of agent network on training and test data set from 0% to 75% (y-axis) in 5500 generations (x-axis).

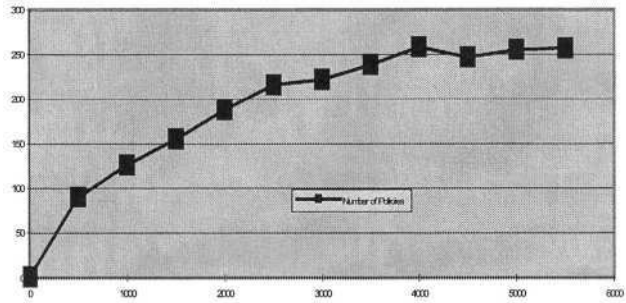


Fig. 3. 257 policies added to agent network after 5500 generations.

Here is a policy from the SOUND agent in the network of figure 1. It has been mutated at least 3 times, once for removing an added policy ('pump'):
(SOUND: ***'ADDED_'/'volume'/'ADDED_' /'del_pump'*** */* ***'sound'*** *{action: {execute '<sound></sound>'}})* Crossover is performed between 2 randomly selected elite pool members. Crossover retains connections and hierarchy of base agent network: *Fitness function = C1 * success-rate – C2 * Number of new conditions, C1 > C2 (C1 and C2 are constants)*
To compute success-rate, the samples are run against the gene being evaluated. *success-rate* is number of corpus samples for which the gene produces a correct expected response, divided by total corpus entries. *Number of new conditions* is the difference between number of conditions in the base agent network and the gene being evaluated. This is to minimize the number of new conditions added.
Test results on a production agent network for CRM/FAQ NLI (67 agents 1172 entry corpus, 188 test corpus, 0% initial success-rate) are shown in figures 2 and 3.
Rate and quality of improvements show promise. In tests, GA always improved success rate, close to the best possible success rate achievable given constraints. AAOSA's high encapsulation makes application of GAs possible and practical.

References

1. Hodjat B, Savoie C J, Amamiya M, An Adaptive Agent Oriented Software Architecture, Proc. of the 5th Pacific Rim International Conference on Artificial Intelligence (PRICAI '98) pp.33-46, 1998.
2. Hodjat B, Amamiya M, Introducing the Adaptive Agent Oriented Software Architecture and its Application in Natural Language User Interfaces, In *Agent Oriented Software Engineering,* P. Ciancarino and M Wooldridge eds, Proc. AOSE 2000 conference, Springer-Verlag, pp. 285-306, 2000.
3. N. R. Jennings and M Wooldridge, "Agent-Oriented Software Engineering" in Handbook of Agent Technology (ed. J. Bradshaw) AAAI/MIT Press, (to appear), 2000.
4. Iglesias, C.A., Garijo, M., & Gonzalez, J.C. A Survey of Agent-Oriented Methodologies. In Intelligent Agents V—Proceedings of the Fifth International Workshop on Agent Theories, Architectures, and Languages (ATAL-98), Lecture Notes in Artificial Intelligence. Springer-Verlag, Heidelberg, 1999.

Optimization of Gaussian Mixture Model Parameters for Speaker Identification

Q.Y. Hong, Sam Kwong, and H.L. Wang

Department of Computer Science, City University of Hong Kong, Hong Kong, China
qyhong,@cs.cityu.edu.hk, {cssamk,wanghl}@cityu.edu.hk

Abstract. Gaussian mixture model (GMM) [1] has been widely used for modeling speakers. In speaker identification, one major problem is how to generate a set of GMMs for identification purposes based upon the training data. Due to the hill-climbing characteristic of the maximum likelihood (ML) method, any arbitrary estimate of the initial model parameters will usually lead to a sub-optimal model in practice. To resolve this problem, this paper proposes a hybrid training method based on the genetic algorithm (GA). It utilizes the global searching capability of the GA and combines the effectiveness of the ML method.

1 The Proposed Algorithm

Spoken utterances convey both linguistic and speaker-specific information. In the GMM-based speaker identification, the distribution of feature vectors extracted from a speaker's utterance is modeled by a weighted sum of mixture components and the speaker model that has the highest likelihood score for the utterance will be selected as the identification result. Before the identification task, the model parameters must be trained to describe the observation sequences of the speaker accurately. The traditional ML method could update the parameters repeatedly but usually only reach a local maximum. The genetic algorithm provides the global searching capability to the optimization problem. Therefore, the GMM training can escape from the initial guess and find the optimal solution if we apply the GA to the training process.

The GA method has been successfully applied for HMM-based speech recognition [2]. In this paper, we extend it to the GMM training and use the ML re-estimation as a special heuristic operator, the iteration time of which is experimentally determined to have a good balance between the searching capability and converging speed. In the proposed GA, the phenotype of a single speaker model is directly represented with the GMM structure in that the diagonal covariance is assumed. In any case, the sum of the mixture weight must satisfy the statistical constraint and will be normalized to 1.0. Moreover, the model parameters in the same mixture are actually correlated so that we form them as a unit in the crossover operator. The basic data type of the elements of the GMM is real number, so we use real number string instead of bit-string as the representation of the individuals and have the following form:

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1310–1311, 2004.

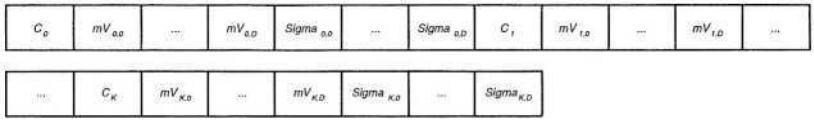


Fig. 1. The representation of the chromosome in the GA training

where C_k, $mV_{k,l}$ and $Sigma_{k,l}$ are the weight, lth mean element and lth diagonal covariance element of the kth mixture component, respectively. The fitness is denned as the average of the log-likelihood of the utterances based on the given model.

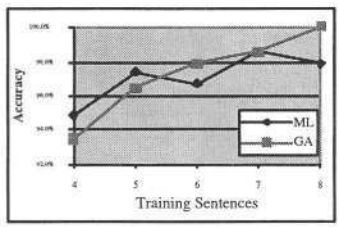
In our GA, five mixtures are randomly selected from the parent for the crossover. Mutation introduces local variations and is applied for each model parameter, which is multiplied by a Gaussian random number generator with mean = 1.0 and variance = 0.001. In each generation, the individual in the offspring is re-estimated five times by the ML operator. Each individual in the population will be further re-estimated eight times every ten GA generations.

2 Experimental Results

From the TI46 corpus, we selected 10 command words of the 8 female speakers to conduct the identification experiment. There were 100 training utterances per speaker and the whole test sections were used as the test data. The left part of Fig. 2 gives the results for different mixture number. It is seen that in all cases, the GMMs trained by our GA had higher values of fitness than the GMMs trained by the ML method. Another experiment was based on TIMIT and 23 female speakers were used. When there were 6 or more training sentences, the performance of our GA was equal to or better than the ML method. Therefore, our training approach was more preferred for sufficient training data.

Speaker	16 mixtures		32 mixtures		64 mixtures	
	ML	GA	ML	GA	ML	GA
F1	-3787.8	-3786.8	-3705.7	-3701.3	-3618.0	-3609.6
F2	-3791.2	-3779.3	-3700.2	-3691.5	-3619.9	-3604.4
F3	-4113.2	-4104.5	-4036.5	-4028.7	-3944.0	-3935.7
F4	-4162.0	-4154.7	-4083.4	-4078.1	-3988.2	-3982.0
F5	-4133.0	-4122.4	-4045.2	-4032.5	-3948.8	-3932.5
F6	-4516.0	-4512.3	-4434.0	-4423.3	-4351.7	-4336.3
F7	-4416.5	-4413.7	-4320.0	-4307.8	-4199.8	-4192.8
F8	-3874.2	-3869.8	-3790.6	-3783.5	-3689.2	-3678.6
Accuracy	89.6%	90.4%	94.2%	95.8%	97.7%	98.2%

8 female speakers in TI46



23 female speakers in TIMIT

Fig. 2. Experimental results of fitness and identification accuracy

References

1. D.A. Reynolds. Speaker identification and verification using Gaussian mixture speaker models. Speech Communication 17 (1995) 91-108.
2. S. Kwong, C.W. Chau, K.F. Man, and K.S. Tang. Optimisation of HMM topology and its model parameters by genetic algorithms. Pattern Recognition 34 (2001) 509-522.

Network Intrusion Detection Using Genetic Clustering

Elizabeth Leon[1], Olfa Nasraoui[1], and Jonatan Gomez[2]

[1] Department of Electrical & Computer Engineering, The University of Memphis
[2] Universidad Nacional de Colombia
{eleon, onasraou, jgomez}@memphis.edu

Abstract. We apply the Unsupervised Niche Clustering (UNC), a genetic niching technique for robust and unsupervised clustering, to the intrusion detection problem. Using the normal samples, UNC generates clusters sumarizing the normal space. These clusters can be characterized by fuzzy membership functions, that are later aggregated to determine a level of normality. Anomalies are identified by their low normality levels.

1 Introduction

Clustering [1] has been applied successfully to the Intrusion Detection Problem (IDP), by generating a set of clusters that can characterize the normal class using the normal samples. The Unsupervised Niche Clustering **(UNC)** is a robust and unsupervised clustering algorithm that uses an evolutionary algorithm to find clusters (using a robust density fitness function), with a niching strategy for maintaining the niches (candidate clusters) [2,3]. In this paper, we combine the UNC with fuzzy sets theory for solving some IDPs [4]. We associate to each cluster evolved by the UNC (cluster center c and scale σ) a membership function that follows a Gaussian shape, $\mu(x) = e^{\left(-\frac{d(x,c)^2}{2\sigma^2}\right)}$. Such function will define the normalcy level of a data sample. Then, the normal class is defined by the fuzzy-union set *(max-OR)* of all the clusters generated *(C)*. Thus, a data sample x is considered normal with a $\mu_{normal}(x) = max\{\mu_i(x)|\forall i = 1, 2, ..C\}$ degree.

2 Experimentation

Tests were conducted on a reduced version of the KddCup'99 data set (21 features) after applying a Principal Component Analysis (PCA) to the non-zero numerical features. We used 5000 normal samples as training data set, while 40% of the full data set was used for testing. Table 1 shows the performance reached by our approach, while Figure 1 shows the ROC curves generated.

3 Conclusions

In this paper, the UNC algorithm [3] was applied to the intrusion detection problem (KddCup'99 data set). A fuzzy characterization of the model generated

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1312–1313, 2004.

by UNC was proposed. Our results show that a fuzzy analysis increases the performance reached by our approach while PCA reduces the complexity of the data set and further improves the performance of our approach [4].

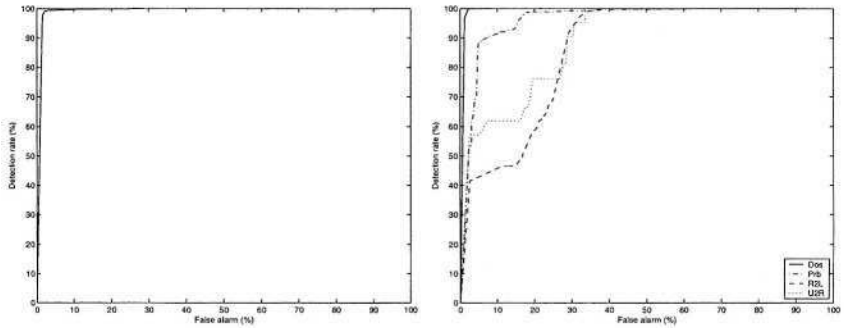


Fig. 1. ROC curve generated by our approach. Left: all attacks, Right: per attack.

Table 1. Performance of the proposed approach on the KddCup'99 data set.

	FULL	DOS	PRB	R2L	U2R
Detection Rate (%)	99.20	95.9	93.9	98.6	90.9
False Alarm Rate (%)	2.20	1.0	12.2	28.6	20.6

Acknowledgments. This work is supported by National Science Foundation CAREER Award IIS-0133948 to O. Nasraoui.

References

1. R. Duda and P. Hart, *Pattern Classification and Scene Analysis.* NY: Wiley Interscience, 1973.
2. O. Nasraoui, E. Leon, and R. Krishnapuram, "Unsupervised niche clustering: Discovering an unknown number of clusters in noisy data sets," in *Evolutionary Computing in Data Mining,* Invited chapter, A. Ghosh and L. C. Jain, Eds, Springer Verlag, 2004.
3. O. Nasraoui and R. Krishnapuram, "A novel approach to unsupervised robust clustering using genetic niching," in *Proceedings of the Ninth IEEE International Conference on Fuzzy Systems,* pp. 170–175, 2000.
4. E. Leon, O. Nasraoui, and J. Gomez, "Anomaly detection based on unsupervised niche clustering with application to intrusion detection," in *Proceedings of the 2004 Congress on Evolutionary Computation,* 2004.

Enhanced Innovation: A Fusion of Chance Discovery and Evolutionary Computation to Foster Creative Processes and Decision Making

Xavier Llorà[1], Kei Ohnishi[1], Ying-ping Chen[1], David E. Goldberg[1], and Michael E. Welge[2]

[1] Illinois Genetic Algorithms Laboratory (IlliGAL), National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign
{xllora,kei,ypchen,deg}@illigal.ge.uiuc.edu
[2] Automated Learning Group, National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign
welge@ncsa.uiuc.edu

Abstract. Human-based genetic algorithms are powerful tools for organizational modeling. If we enhance them using chance discovery techniques, we obtain an innovative approach for computer-supported collaborative work. Moreover, such a user-centered approach fuses human and computer partners in a natural way. This paper presents a first test, as well as analyzes the obtained results, of real human and computer collaboration powered by the fusion of human-based genetics algorithms and chance discovery.

1 Motivation

DISCUS *(Distributed Innovation and Scalable Collaboration in Uncertain Settings)* [1] is an endeavor to create innovation support mechanisms. Such innovation support mechanisms rely on the metaphor of genetic algorithms as models of human innovation. The metaphor establishes both a qualitative and quantitative pathway for the design of such systems, taking advantage of mechanism and analytical design theory developed for more than twenty years on competent GAs [2]. Human-based genetic algorithms [3] take this so-called innovation intuition and run with it, using common GA concepts and theory as a means to design pervasive and open-ended innovation support systems. From the more practical perspective of creativity consultants and practitioners, human-based genetic algorithms may also be regarded as the formal next step beyond face-to-face brainstorming [4] and electronic brainstorming [5]. DISCUS enhances the human-based genetic algorithms with two powerful tools (1) data- and text-mining, and (2) chance discovery [1]. Data- and text-mining tools provide the computational embodiments for frequent pattern extraction during distributed on-line collaboration. Chance discovery—when used together with genetic algorithms—provides a natural mechanism to identify salient fortuitous events [6,7], emphasizing rare events during the on-line collaboration.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1314–1315, 2004.

2 Lessons Learned

The goal of the work summarized here was to use a first DISCUS pilot prototype—synthesis of elements to promote distributed innovation using web-based IT infrastructure—for discussing research topics about the linkage learning genetic algorithm [8]. The first test of DISCUS on a real-world problem involved 11 researchers in two countries, 8 in the USA and 3 in Japan, collaborating to brainstorm solutions for the improvement of LLGA. The collaborative team (1) improved their understanding of the linkage learning genetic algorithm state-of-the-art research, and (2) suggested solutions judged to be of high quality by linkage learning genetic algorithm experts. Strongly highlighted by discussion members was the simplicity to interact between geographically distant locations and researchers, as well as to reason in non-linear ways, sparking their creativity.

Current research and implementations, after this first DISCUS experience, focus on (1) improving the integration of the different available technologies, (2) creating a tutorial mode for learning purposes, (3) providing discussion structures for various innovation-supported scenarios, and (4) advancing in the theoretical understanding of human-based genetic algorithms and interactive genetic algorithms with and without the interaction of chance discovery and data- and text-mining tools using the Illinois decomposition methodology.

Acknowledgments. This work was sponsored by the AFOSR (F49620-03-1-0129), and by TRECC-NCSA-ONR (N00014-01-1-0175). The views and conclusions contained herein are only those of the authors.

References

1. Goldberg, D.E., Welge, M., Llorà, X.: DISCUS: Distributed Innovation and Scalable Collaboration In Uncertain Settings. IlliGAL Report No. 2003017, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Lab, Urbana, IL (2003)
2. Goldberg, D.E.: The Design of Innovation: Lessons from and for Competent Genetic Algorithms. Kluwer Academic Publishers, Norwell, MA (2002)
3. Kosorukoff, A., Goldberg, D.E.: Evolutionary computation as a form of organization. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002), Morgan Kaufmann (2002) 965–972
4. Osborn, A.F.: Applied imagination. Scribners, New York (1953)
5. Holt, K.: Brainstorming-from classics to electronics. Engineering Design **7** (1996) 77–82
6. Goldberg, D.E., Sastry, K., Ohsawa, Y.: Discovering deep building blocks for competent genetic algorithms using chance discovery. In: Ohsawa, Y. & McBurney, P. (eds.), Chance discovery, Springer-Verlag (2003) 276–301
7. Ohsawa, Y.: Chance discoveries for making decisions in complex real world. New Generation Computing **20** (2002) 143–163
8. Harik, G.R., Goldberg, D.E.: Learning linkage. Foundations of Genetic Algorithms **4** (1996) 247–262

Development of a Genetic Algorithm for Optimization of Nanoalloys

Lesley D. Lloyd[1], Roy L. Johnston[1], and Said Salhi[2]

[1] School of Chemistry, University of Birmingham,
Edgbaston, Birmingham B15 2TT, UK
lesley@tc.bham.ac.uk
[2] School of Mathematics and Statistics, University of Birmingham,
Edgbaston, Birmingham B15 2TT, UK

Abstract. A genetic algorithm has been developed in order to find the global minimum of platinum-palladium nanoalloy clusters. The effect of biasing the initial population and predating specific clusters has been investigated.

1 Introduction

Clusters are aggregates ranging from a few to many millions of atoms [1]. The desire to fabricate materials with well defined, controllable properties and structures, on the nanometre scale, coupled with the flexibility afforded by intermetallic materials, has engendered considerable interest in bimetallic "nanoalloy" clusters. Our research generally involves using a genetic algorithm (GA) to find the arrangement of atoms corresponding to the global minimum (GM) on the potential energy hypersurface and our standard cluster optimization GA program has been described in a recent review [2].

Here we describe the improvement of our GA for studying nanoalloy clusters, such as those of palladium (Pd) and platinum (Pt) [3,4], using the Gupta many-body potential [5] to describe the Pd–Pd, Pd–Pt and Pt–Pt interactions. In our previous work, we found that certain GM are considerably more difficult to find than others – for example $Pt_{12}Pd_{12}$ [3,4], which was chosen as the test example for developing an improved algorithm. To this end, the standard GA program has been modified to include biasing of the initial population and the use of a "predator" operator to remove certain local minima from the population.

2 Results

In our original GA program, the initial population (gen_0) was chosen at random. In this study, we have used our chemical insight to bias the initial population: by replacing some members of the initial population with selected cluster geometries; and by inserting one or two geometrically favoured clusters into the initial random population. Firstly, particularly unfavorable clusters – the "reverse GM"

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1316–1317, 2004.

where all Pt and Pd atoms are swapped relative to the GM structure and a cluster where the Pt and Pd atoms are segregated into separate halves of the cluster – were included in gen_0. Although contributing to the diversity of the population, as was expected, this "unfavourable biasing" leads to the GA being less successful at finding the GM than using a completely random gen_0. Secondly, a low energy, "favorable" cluster, was added to 19 randomly generated clusters in gen_0. This particular cluster, which has a potential energy of -105.9919 eV, is the fourth lowest isomer overall, hence it is the third lowest local minimum, LM3. The LM3 structure was selected to "seed" the initial population because it is found (in the generation immediately preceding that in which the GM is found) in 54% of successful GA runs. As the GA is elitist, the LM3 structure is maintained in the population from one genaration to the next. Including LM3 in gen_0, increased the rate of finding the global minimum from 22 to 43%.

We have previously introduced a "predator" operator for removing unwanted individuals or traits from the population [6]. An energy predator can be used to search for low energy minima other than the global minimum, or to enhance the efficiency of the GA by removing specific low energy non-global minima that the GA may be incorrectly converging towards. Since there is a high incidence of convergence of the GA on the sub-optimal structures LM1 and LM2, these structures can be regarded as "traps", which prevent the GA finding the GM. In order to test this hypothesis, predation of these clusters was carried out. With a random gen_0 and predation of LM2, the GM was found 31 times and with predation of LM1 28 times – an increase of 9% and 7%, respectively. With predation of both LM1 and LM2, the GM was found 43 times – an increase of 21%, however when both LM1 and LM2 were predated and LM3 was seeded into gen_0, the GM was found 65 times – an incease of 43%.

In conclusion, the results reported here, together with those from previous studies [2], show that the GA is a powerful technique for finding the GM for nanoalloys, as well as elemental clusters. Biasing the population and predating local minima has produced a more robust algorithm. This strategy can be used to study larger nanoalloy clusters.

References

1. Johnston, R.L. Atomic and Molecular Clusters. (Taylor and Francis, London, New York, 2002)
2. Johnston, R.L. Evolving Better Nanoparticles: Genetic Algorithms for Optimising Cluster Geometries. Dalton Transactions (2003) 4193-4207.
3. Massen C., Mortimer-Jones T.V., Johnston R.L., Journal of The Chemistry Society, Dalton Transactions (2002) 4375
4. Lloyd, L.D., Johnston, R.L.,Salhi, S.: Theoretical Investigation of Isomer Stability in Platinum-Palladium Nanoalloy Clusters. Journal of Material Science, in press.
5. Cleri, F., Rosato, V.: Tight-Binding Potentials for Transition-Metals and Alloys. Physical Review B **48** (1993) 22–33.
6. Manby, F.R., Johnston, R.L., Roberts, C.: Predatory Genetic Algorithms. MATCH (Communications in Mathematical and Computational Chemistry) **38** (1998) 111–122.

Empirical Performance Evaluation of a Parameter-Free GA for JSSP

Shouichi Matsui, Isamu Watanabe, and Ken-ichi Tokoro

Central Research Institute of Electric Power Industry (CRIEPI)
2-11-1 Iwado-kita, Komae-shi, Tokyo 201-8511, JAPAN
{matsui,isamu,tokoro}@criepi.denken.or.jp

1 Introduction

The job-shop scheduling problem (JSSP) is a well known difficult NP-hard problem. Genetic Algorithms (GAs) for solving the JSSP have been proposed, and they perform well compared with other approaches [1]. However, the tuning of genetic parameters has to be performed by trial and error. To address this problem, Sawai et al. have proposed the Parameter-free GA (PfGA), for which no control parameters for genetic operation need to be set in advance [3].

We proposed an extension of the PfGA, a real-coded PfGA, for JSSP [2], and reported that the GA performed well without tedious parameter-tuning. This paper reports the performance of the GA to a wider range of problem instances. The simulation results show that the GA performs well for many problem instances, and the performance can be improved greatly by increasing the number of subpopulations in the parallel distributed version.

2 Computational Results

The GA is tested by the benchmark problems from ORLib [4]. We tested a wider range of instances, namely 10 tough problems, ORB01–ORB10, SWV01–SWV20, and TA01–TA30, but only the results for ORB01–ORB10 and SWV01–SWV20 are shown in Table 1 and Table 2. The GA was run for each problem instance using 50 different random seeds. The maximum number of fitness evaluation was set to 1,000,000 for all cases.

- The GA can always find the optimal makespan when $N \geq 1$ in ORB01, ORB03, ORB04, and ORB07–ORB10, when $N \geq 2$ in ORB10, when $N \geq 32$ in ORB09, and when $N = 64$ in ORB08.
- As we increase the number of subpopulations, the average makespan is reduced, but the best makespan does not always shorten.
- The relative error of SWV instances to the best upper bound is larger than that of ORB cases. The size of the problems is a reason of the difference. The size of ORB problems is 10×10 and it is 20×10, 20×15, or 50×20 in SWV problems. The length of the chromosomes used in the GA is proportional to the size of problems, $2 \times n\,(\text{jobs}) \times m\,(\text{machines})$, therefore the length *(L)* is 200 in ORB problems, and 400, 600, or 2000 in SWV problems.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1318–1319, 2004.

Table 1. Relative error to the optimal makespan (%) (ORB problems)

		$N=1$		$N=2$		$N=4$		$N=8$		$N=16$		$N=32$		$N=64$	
Prob.	Opt[1]	best	μ	best	μ	best	μ	best	μ	best	μ	best	μ	best	μ
ORB01	1059	**0.0**	2.6	**0.0**	2.1	**0.0**	1.6	**0.0**	1.0	**0.0**	0.7	**0.0**	0.3	**0.0**	0.1
ORB02	888	0.1	0.5	0.1	0.5	**0.0**	0.3	**0.0**	0.1	**0.0**	0.1	**0.0**	0.1	**0.0**	0.1
ORB03	1005	**0.0**	2.3	**0.0**	2.2	**0.0**	1.7	**0.0**	1.1	**0.0**	1.0	**0.0**	0.6	**0.0**	0.2
ORB04	1005	**0.0**	1.2	0.6	1.4	0.6	1.1	**0.0**	0.8	**0.0**	0.6	**0.0**	0.5	**0.0**	0.3
ORB05	887	0.2	0.7	0.2	0.5	0.2	0.4	**0.0**	0.2	**0.0**	0.2	**0.0**	0.2	**0.0**	0.2
ORB06	1010	0.2	1.9	0.2	1.6	0.2	1.4	**0.0**	1.1	**0.0**	0.9	**0.0**	0.7	**0.0**	0.4
ORB07	397	**0.0**	1.1	**0.0**	1.0	**0.0**	0.8	**0.0**	0.6	**0.0**	0.2	**0.0**	0.0	**0.0**	**0.0**
ORB08	899	**0.0**	2.4	**0.0**	2.4	**0.0**	1.9	**0.0**	1.5	**0.0**	0.8	**0.0**	0.4	**0.0**	0.3
ORB09	934	**0.0**	0.6	**0.0**	0.8	**0.0**	0.3	**0.0**	0.3	**0.0**	0.1	**0.0**	**0.0**	**0.0**	**0.0**
ORB10	944	**0.0**	0.1	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**

Table 2. Relative error to the best upper bound (%) (hard SWV problems)

		$N=1$		$N=2$		$N=4$		$N=8$		$N=16$		$N=32$		$N=64$	
Prob.	bub[1]	best	μ	best	μ	best	μ	best	μ	best	μ	best	μ	best	μ
SWV01	1407	5.8	9.8	5.5	8.5	5.1	7.5	4.5	7.1	3.5	5.9	2.5	5.1	3.7	5.2
SWV02	1475	5.0	7.7	4.3	7.1	4.2	6.4	3.5	5.6	1.6	5.2	1.4	4.6	1.9	4.3
SWV03	1398	5.9	8.9	6.1	8.8	4.4	7.2	3.7	6.5	2.5	5.5	2.7	5.1	2.5	4.8
SWV04	1483	5.1	8.0	3.6	7.3	3.0	6.5	2.9	5.8	1.8	4.9	2.4	4.2	1.8	3.5
SWV05	1424	7.6	10.7	7.0	10.0	6.3	8.9	4.1	8.2	3.6	7.0	3.2	6.3	3.2	5.8
SWV06	1678	8.3	12.0	8.5	11.4	6.7	10.6	6.9	9.5	6.2	8.8	6.0	7.8	4.4	7.1
SWV07	1620	7.2	10.4	6.9	10.3	5.2	9.6	5.0	8.7	5.6	7.5	4.2	6.9	4.1	6.2
SWV08	1763	10.0	13.8	9.6	12.8	5.8	11.5	6.7	10.7	5.8	9.7	6.7	8.9	5.0	8.0
SWV09	1663	8.8	12.6	7.2	12.1	7.8	11.1	6.0	10.0	6.0	9.0	4.7	8.0	4.8	7.3
SWV10	1767	7.6	9.8	6.1	8.8	4.4	8.2	5.3	7.5	4.6	6.7	3.6	5.8	2.4	5.4
SWV11	2991	11.2	16.0	10.9	15.6	9.8	14.8	10.6	13.5	8.3	12.2	8.4	11.1	7.4	10.3
SWV12	3003	14.2	16.9	12.5	16.0	10.7	15.0	11.1	14.6	10.9	13.1	9.8	12.2	9.0	11.5
SWV13	3104	11.5	14.9	10.1	14.1	8.8	12.6	9.0	12.1	7.9	11.2	8.7	10.3	6.6	9.1
SWV14	2968	10.3	14.1	9.8	13.7	7.4	11.7	8.4	11.2	6.5	9.8	6.1	9.1	5.0	7.4
SWV15	2904	14.6	18.0	13.8	17.7	12.6	16.3	11.6	15.4	10.5	14.3	9.8	12.8	9.8	12.1

References

1. Jain, A.S., and Meeran, S.: Deterministic job-shop scheduling: past, present and future, *European Journal of Operational Research,* vol.113, pp.390–434, 1999.
2. Matsui, S., Watanabe, I., and Tokoro, K.: Real-coded parameter-free genetic algorithm for job-shop scheduling problems, *Proc. Seventh Parallel Problem Solving from Nature – PPSN VII,* pp.800–810, 2002.
3. Sawai, H., Kizu, S.: Parameter-free genetic algorithm inspired by "disparity theory of evolution", *Proc. Fifth Parallel Problem Solving from Nature – PPSN V,* pp.702–711, 1998.
4. Vaessens, R.J.M.: Operations Research Library of Problems, Management School, Imperial College London, ftp://mscmga.ms.ic.ac.uk/pub/jobshop1.txt, 1996.

A Caching Genetic Algorithm for Spectral Breakpoint Matching

Jonathan Mohr[1] and Xiaobo Li[2]

[1] Augustana University College, Camrose, Alberta, Canada T4V 2R3
mohrj@augustana.ca, http://www.augustana.ca/~mohrj/
[2] University of Alberta, Edmonton, Alberta, Canada T6G 2M7
li@cs.ualberta.ca, http://www.cs.ualberta.ca/~li/

Abstract. Two methods were evaluated for performing spectral breakpoint matching: a multi-level pruned exhaustive search and a genetic algorithm. The GA found matches about as good as those found by pruned search, but savings in time were realized only if the objective function cached the results of previous evaluations.

1 Introduction

Multiple wavetable interpolation synthesis [1] is a form of music analysis/synthesis in which a recorded sound is reduced to a set of breakpoints by piecewise linear approximation of the spectral envelopes of its harmonics. The spectrum at each breakpoint is then matched by determining weightings for a small number of wavetables selected from a *wavetable bank,* and the sound is resynthesized using multiple wavetable additive synthesis by interpolating between the weightings for each wavetable at consecutive breakpoints.

2 Spectral Breakpoint Matching

Given a particular wavetable bank, a set of breakpoint data representing a particular tone, and the number of oscillators *(N)* to be used in resynthesis, our breakpoint-matching algorithm [2] selects at most *N* wavetables from the bank of basis spectra that, in weighted combination, best match the spectrum at each breakpoint according to some error measure.

In the first stage of the algorithm, an initial match is found for each breakpoint. In a subsequent optimization stage, a weighted wavetable is assigned to each available oscillator at each breakpoint such that the overall error is minimized, taking into account the need to fade a wavetable in or out when it begins or ceases to be used [3].

Two methods were evaluated for finding initial matches in the first stage: a multi-level pruned search and a genetic algorithm (GA). It was found that a "3+1" search—an exhaustive search for the best 3-wavetable matches, augmented with a fourth wavetable by a second-level search—executes about an

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1320–1321, 2004.

order of magnitude faster than an exhaustive search for a match of size 4, yet yields about the same or better error rates, on average, after optimization [4].

When a genetic algorithm was used as a first-level search in combination with an exhaustive second-level search, the GA found matches about as good as those found by exhaustive or pruned search; however, savings in time were only realized relative to the larger pruned searches, and then only if the objective function cached the results of previous calls to the evaluation functions.

3 A Caching Genetic Algorithm

Caching was implemented by introducing two mappings as static data members of the objective function so that the contents of the mappings would be preserved across the evaluations of all individuals in all generations. The first maps from sets of wavetables to the results of LUP decompositions, the second, from breakpoint number and wavetable set to the corresponding least-squares solution. When iterating across the multimap, the first map is checked for each wavetable set in the current individual, and LUP decompositions are performed only for those wavetable sets not already in the map; similarly, when iterating across the breakpoints at which a given wavetable set is used in the current individual, the second map is searched for a pre-existing least-squares solution and error level.

One of the configurations that was tested used a population size of 50 and early termination (25 generations to convergence). Without caching, this GA performed about 25% more LUP decompositions and least-squares evaluations than the pruned search to find the initial 3-wavetable matches. With caching, the GA performed only 3% as many calculations as the GA without caching, finishing in 11% of the time.

4 Conclusion

The use of mappings as static data members of the objective function to cache previously calculated results for reuse in evaluating the fitness of all individuals across generations was essential in order to make a GA competitive in time with pruned exhaustive search.

References

1. Horner, A.: Computation and memory tradeoffs with multiple wavetable interpolation. Journal of the Audio Engineering Society **44** (1996) 481–496
2. Mohr, J.: Music Analysis/Synthesis by Optimized Multiple Wavetable Interpolation. PhD thesis, University of Alberta (2002)
3. Serra, M.H., Rubine, D., Dannenberg, R.: Analysis and synthesis of tones by spectral interpolation. Journal of the Audio Engineering Society **38** (1990) 111–128
4. Mohr, J., Li, X.: Computational challenges in multiple wavetable interpolation synthesis. In Sloot, P.M., et al., eds.: Computational Science—ICCS 2003. Number 2657 in Lecture Notes in Computer Science, Springer (2003) 447–456

Multi-agent Simulation of Airline Travel Markets

Rashad L. Moore[1], Ashley Williams[1], and John Sheppard[2]

[1]MITRE, Corporation, McLean, Virginia
{rlmoore, ashley}@mitre.org
[2]ARINC Engineering Services, LLC, Annapolis, Maryland
jsheppar@arinc.com

Abstract. This paper explores the use of a learning classifier system variant, XCS, in learning effective airline decision rules within the context of a multi-agent team simulation. From this study a general approach to modeling an airline market has been developed based on a multi-agent, team-based concept. Additionally, several preliminary trials of the simulation have been executed.

1 Introduction

Traditionally, the Local Airport Authorities (LAA) charge airlines a flat rate to land and depart from their airports regardless of a flight's arrival or departure time. Due to passenger preferences this policy results in heavy travel during the two "rush hour" periods - morning and early evening. In an attempt to encourage airlines to schedule their flights evenly throughout the day, thus easing congestion, we are interested in experimenting with variable pricing policies. This study investigates what the response of the individual airlines would be to the proposed change in pricing policy. The aim of this project is to accurately model the decision making process of the airlines using a multi-agent model, in which each airline is modeled by a cooperating set of teams (i.e., a "team-of-teams") that learn to work in concert to compete effectively against other similarly structured airline teams.

2 Multi-agent Model Structure

The agent model (Figure 1) consists of several teams comprised of learning agents that we call, Flight Agents, each representing a single flight (departure or arrival). Flight Agents are logically grouped into teams representing the specific routes that the airline services (Route Teams). Flight Agents must cooperate with the members of its particular Route Team. Additionally, the set of Route Teams as a whole must learn to cooperate with each other to optimize overall airline profitability. Furthermore, these Airline Teams must also learn to compete against other similarly structured Airline Teams for passenger market share. Each Flight Agent contains its own schedule that consists of the Fare, Time, and Capacity of the flight. Collectively, the Flight Agents represent the Airline's flight schedule for that airport.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1322–1323, 2004.

Each flight of an airline's flight schedule is evaluated in a simulation environment by the Passenger Demand Allocator Model (PDAM). The PDAM assigns a utility value to each of the flights and allocates passengers to these flights based on their relative utility values.

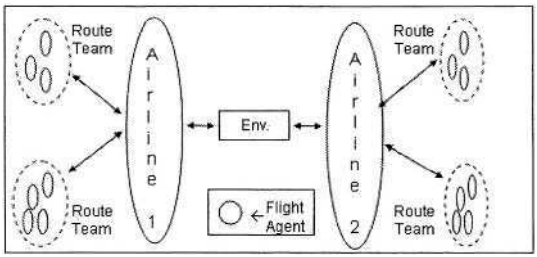


Fig. 1. Multi-agent structure for airline market.

3 Learning Algorithm

For this initial study each Flight Agent used a Learning Classifier System (LCS) [1] as its learning algorithm. Specifically, the LCS algorithm used was XCS [2], a variant of Holland's original LCS. The rules' conditions were comprised of 15 attributes divided among five categories: profitability, passenger load factor, market share, price, and arrival/departure time. The allowed actions for each flight agent included adjusting its own fare and time slot, selecting a smaller (or larger) plane, and deciding whether it should fly (thus allowing airlines to reduce the number of flights in a particular market).

4 Preliminary Results

Our preliminary results were inconclusive, however, promising enough to warrant further refining our airline market model. This study used very simplistic passenger demand allocation and flight utility models. Once acceptable performance is realized on the simplified environment more complex models will be integrated into the environment.

References

1. Holland, J.H. (1995). Escaping Brittleness: The Possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems. *G. F. Luguer, editor, Computation & Intelligence: Collected Readings,* Cambridge, Mass: MIT Press, pages 275 – 304.
2. Wilson, S. W., Butz, M. V. (2001). An Algorithmic Description of XCS. *Lecture Notes in Computer Science,* Vol. 1996, pp. 253+.

Improved Niching and Encoding Strategies for Clustering Noisy Data Sets

Olfa Nasraoui and Elizabeth Leon

Department of Electrical and Computer Engineering, The University of Memphis
Memphis, TN 38152
{onasraou,eleon}@memphis.edu

Abstract. Clustering is crucial to many applications in pattern recognition, data mining, and machine learning. Evolutionary techniques have been used with success in clustering, but most suffer from several shortcomings. We formulate requirements for efficient encoding, resistance to noise, and ability to discover the number of clusters automatically.

1 Introduction

The Need for a Robust Fitness Measure: Most existing evolutionary clustering techniques, such as [1], and [2], rely on a fitness that is based on a Sum of Squared Errors that is sensitive to noise because it increases indefinitely with distance. A *robust* fitness function can resist noise, for example by weighting the points' contributions by a robust weight function that decreases the influence of outliers.
The Need for a Scalable Chromosome Encoding: The chromosome in most existing evolutionary clustering techniques either encodes a possible partition of the entire data set, or encodes all the cluster prototypes. The former encoding leads to an explosion of the search space size as the data set gets larger. The latter assumes a known number of clusters and leads to a search space size that explodes exponentially with the number of clusters. A scalable encoding that is independent of the number of clusters and the size of the data, encodes a single cluster prototype in each chromosome.
The Need for Niching and Automatic Niche Size Estimation: An optimal single cluster encoding strategy will cause the fitness to have a different mode for each cluster. Therefore, niching methods are required. As in nature, niches in our context correspond to different subspaces of the environment (clusters) that can support different types of life (data samples).

2 The Unsupervised Niche Clustering and Comparison to Existing Evolutionary Clustering Techniques

The Unsupervised Niche Clustering (UNC) [3] is a recent approach to evolutionary clustering. UNC uses a chromosome representation encoding a single cluster prototype, and optimizes a density based fitness function that reaches a maximum at every good cluster center, hence requiring a niching strategy. A hybrid scale updating strategy is used to estimate the niche sizes reliably, and thus improve the niching. Because UNC

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1324–1325, 2004.

uses robust weights in its cluster fitness definition, it is less sensitive to the presence of noise. Furthermore, the combination of the single-cluster chromosome encoding with niching offers a simple and efficient approach to automatically determine the optimal number of clusters.

Table 1 compares some evolutionary clustering techniques, including UNC.

Table 1. Comparison of UNC with Other Evolutionary Clustering Algorithms for data of size N, population of size N_P, and C clusters

Approach →	*UNC* [3]	*GGA* [1]	*Lee* [2]	*G-C-LMedS* [4]	*k-d-Median* [5]
Search Method	GA	GA	ES	GA	GA
Robustness to noise	yes	no	no	yes	yes
Automatic Scale Estimation	yes	no	no	no	no
Complexity per Generation	$O(NN_P)$	$O(CNN_P)$	$O(CNN_P)$	$O(CNN_P)$	$O(N_PCN\log(N))$
Hybrid	yes	no	no	no	yes
Does not require No. of Clusters	yes	no	yes	no	no
Handles ellipsoidal clusters	yes	no	no	no	no
Density/Partition	Density	Partition	Partition	Partition	Partition

3 Conclusion

Most existing clustering techniques necessitate the derivation of the optimal prototypes by differentiation to guarantee convergence to a *local* optimum, which can be impossible for most subjective and non-metric dissimilarity measures. For this reason, Evolutionary clustering methods are preferrable. Unfortunately most evolutionary clustering techniques are sensitive to noise, and assume a known numer of clusters. We summarized requirements to ensure efficient encoding, resistance to noise, and ability to discover the number of clusters automatically.

Acknowledgment. This work is supported by a National Science Foundation CAREER Award IIS-0133948 to O. Nasraoui.

References

1. L. O. Hall, I. O. Ozyurt, and J. C. Bezdek, "Clustering with a genetically optimized approach," *IEEE Trans. Evolutionary Computations,* vol. 3, no. 2, pp. 103–112, July 1999.
2. C.-Y. Lee and E. K. Antonsson, "Dynamic partitional clustering unsing evolution strategies," in *3rd Asia Pacific Conf. on simulated evolution and learning,* Nagoya, Japan, 2000.
3. O. Nasraoui and R. Krishnapuram, "A novel approach to unsupervised robust clustering using genetic niching," in *Ninth IEEE International Conference on Fuzzy Systems,* San Antonio, TX, May 2000, pp. 170–175.
4. O. Nasraoui and R. Krishnapuram, "Clustering using a genetic fuzzy least median of squares algorithm," in *North American Fuzzy Information Processing Society Conference,* Syracuse NY, Sep. 1997.
5. V. Estivill-Castro and J. Yang, "Fast and robust general purpose clustering algorithms," in *Pacific Rim International Conference on Artificial Intelligence,* 2000, pp. 208–218.

A Multi-objective Approach to Configuring Embedded System Architectures

James Northern and Michael Shanblatt

Department of Electrical and Computer Engineering
2120 Engineering Building
East Lansing, MI 48824-1226
jnorthern@xula.edu, mas@msu.edu

Abstract. Portable embedded systems are being driven by consumer demands to be thermally efficient, perform faster, and have longer battery life. To design such a system, various hardware units (*e.g.,* level one (L1) and level two (L2) caches, functional units, registers) are selected based on a set of specifications for a particular application. Currently, chip architects are using software tools to manually explore different configurations, so that tradeoffs for power consumption, performance, and chip size may be understood. The primary contribution of this paper is the development of a novel power-performance design tool based around a core GA search and optimization technique. The tool targets the implementation of portable embedded systems.

1 Summary

This paper presents a framework for an evolutionary computational approach to configuring an "ideal" embedded processor based on power consumption and performance. In addition, a database of simulation results that gives a more comprehensive evaluation of tradeoffs between power, performance, and inter-dependence between parameter configurations (*i.e.,* L1 to L2 cache size, memory bandwidth, instruction window size, datapath width) is presented. Appropriate search techniques to reduce exploration space and decrease time-to-market are also discussed.

Power and performance trade-offs are evaluated through the use of Pareto-optimal analysis, for benchmark designs. The experiments used estimated values with a targeted workload (*test-math.c* and *126.gcc*) for power consumption and cycles per instruction (CPI) per configuration. The fitness function was varied based on a weighted scale, where power or performance is selected, with a 100% weighted value, and as 50% power and 50% performance to gather a mixture of both objectives to guide the search process. Five initial randomly seeded populations were submitted to the GA for diverse exploration of the search space. The population size for each run was 50, with termination set at a maximum of 50 generations. A single-point crossover rate of 30%, and mutation rate of 1.4% was used for reproduction operators. The selection scheme for new chromosomes was stochastic universal sampling using linear scaling of fitness values.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1326–1327, 2004.

A comparison of published performance and maximum power for three processors is presented. These three processors are from MIPS, Wattch, and a study done at the University of Pennsylvania [1,14]. The first configuration is the MIPS processor taken from the R10000 specifications [1]. The second configuration is a baseline model for a high-end processor taken from the Wattch simulator toolset [1]. The third configuration is taken from a study done at the University of Pennsylvania (PENN), which demonstrated different tradeoffs between configurations for improved performance. The first two configurations used out-of-order processing, while the last configuration used in-order processing. The in-order processing improves power efficiency, but increases CPI performance. Since performance is the leading objective for these examples, the original configuration is taken from the best solution in the first generation of solutions for performance optimization. The results are expressed as a percentage of the original designs' power consumption and CPI performance. The percentage illustrates the improvement in power consumption and performance for conventional designs and those derived through the GA optimization tool.

However, for each GA run the other objective suffered. With performance being the main objective in the case of the math benchmark, the calculated power consumed showed a decline of 125% in comparison to the original best configuration. The industry example, MIPS R10K simulated processor showed an improvement of 14.19% in power consumption, but a 26.69% decline in CPI performance in comparison to the original configuration. In comparison with the other specified processor configurations, the R10K trades power consumption for higher performance. The Wattch example showed an improvement of 30.09% in power consumption, but a 14.67% decline in CPI performance. The Wattch example considers both power and performance for an overall better processor. Again, the configuration used in a study from the University of Pennsylvania showed lower performance and better power savings in comparison to the other examples.

In the case of the GNU C compiler benchmark, the architecture was optimized for power and the maximum amount of savings was 34.71% with a 10% increase of performance. Using the multi-objective search, we were able to select a configuration with improved power consumption of 49.17%, sacrificing performance with a gain of 1%. The total in power-performance savings was ~50% compared to single-objective optimization total of ~45%.

References

[1] D. Brooks, V. Tiwari, M. Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," Proc. of the 27th International Symposium on Computer Architecture, June 2000, Vancouver, BC, pp. 83-94.

[2] Roth, A., online at http://www.cis.upenn.edu/~amir/cis501-01, CIS 501: Introduction to Computer Architecture, University of Pennsylvania, 2001.

Achieving Shorter Search Times in Voice Conversion Using Interactive Evolution

Yuji Sato

Faculty of Computer and Information Sciences, Hosei University
3-7-2 Kajino-cho, Koganei-shi, Tokyo 184-8584, Japan
yuji@k.hosei.ac.jp

Abstract. We have already proposed using evolutionary computation to adjust the voice quality conversion parameters, and we have reported that this approach produces results that are not only closer to the desired target than the results of parameter adjustment based on designer experience or trial and error, but which also have relatively little sound quality degradation. In this paper we propose improved techniques for the generation of initial entities and genetic manipulation in order to reducing the workload associated with human evaluation in interactive evolution. We perform voice quality conversion experiments both on natural speech recorded with a microphone and on synthetic speech generated from text data. As a result, we confirm that the proposed improvements make it possible to perform voice quality conversion more efficiently than when using the technique proposed earlier.

1 Background and Basic Idea for Reducing the Workload

New markets are appearing using voice quality conversion technology. These include multimedia-content editing, computer games, and man-personal machine interfaces. For example, in multimedia-content editing, adding narration to business content such as presentation material and digital catalogs or to personal content such as photo albums and self-produced video can enhance content. It is not necessarily easy, however, for the general user to provide narration in a clear and intelligible voice, and the need for voice quality conversion can be felt here. Against the above background, we have proposed the application of evolutionary computation to parameter adjustment for the sake of voice quality conversion, and it has also been shown that the use of evolutionary computation for parameter adjustments can be effective at improving the clarity not only of natural speech that has been subjected to voice conversion but also of synthetic speech generated automatically from text data.

In the system we proposed earlier [1], we represented the three variables α, β and γ as real numbers, and we defined a chromosome as an array of the form $[\alpha, \beta, \gamma]$. Then, we performed the crossover operation by randomly selecting one variable from among the three array elements and swapping the values of this variable between two parent entities. Next, we use mutation as represented by Eq. (1) to raise the probability that target mutants are in the vicinity of parents

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1328–1329, 2004.

and to improve local searching. In the equation, C_i represents a modification coefficient for generation i, I is a unit matrix, k is a constant, and N is a normal distribution function with a mean vector of 0 and a covariance of kI and is common to all elements.

$$C_{i+1} = C_i + N(0, kI) \tag{1}$$

Here, the first problem to be addressed in practical implementations of the above system is that the interactive evolution scheme places a heavy workload on the evaluators. At first, the evaluator workload depends on the number of evaluations that have to be performed. We think that if empirical conversion coefficients for different conversion objectives are used as the starting values, then it should be possible to reduce the total number of entities even if there are no empirical conversion coefficients available from previous manual searches.

Next, it is thought that the search performance can be improved by making changes to the crossover and spontaneous mutation operations so as to reduce the number of generations needed to arrive at a practical quasi-optimal solution. In new operation, one of the two entities generated by the crossover operation shown the above is randomly subjected to crossovers in which the average value of each coefficient in the two parent entities [2] are obtained.

For spontaneous mutations, the standard deviation of the mutation distribution was set small as shown in Eq. (2) for entities where crossovers were performed just by swapping coefficients as in the conventional approach.

$$C_{i+1} = C_i + N(0, 0.000025I) \tag{2}$$

Conversely, a larger standard deviation was set for entities where crossovers were performed by taking the average of two coefficients as shown in Eq. (3).

$$C_{i+1} = C_i + N(0, 0.01I) \tag{3}$$

That is, the emphasis is placed on local search performance for entities where crossovers are performed in the same way as in earlier systems, and the emphasis is placed on increasing diversity and searching new spaces for entities where crossovers are performed by obtaining the average of two coefficients.

Using an evolutionary computation conversion method reported earlier [1] and the conversion technique proposed here, we compared the number of evaluations (the number of generations multiplied by the number of entities). In terms of the number of evaluations, convergence is achieved with an even smaller number of evaluations (from 100 times to 40 times) when the measures proposed here are applied.

References

1. Sato, Y.: Voice Conversion Using Interactive Evolution of Prosodic Control. In: Proc. of the 2002 Genetic and Evolutionary Computation Conference (GECCO-2002), Morgan Kaufmann Publishers, San Francisco, CA (2002) 1204–1211
2. Back, T., Fogel, D.B., and Michalewicz, Z. (eds.): Evolutionary Computation 1: Basic Algorithms and Operators. Institute of Physics Publishing, Bristol, UK (2000)

Predicting Healthcare Costs Using Classifiers

C.R. Stephens[1,2], H. Waelbroeck[1,3], S. Talley[1], R. Cruz[1], and A.S. Ash[4,5]

[1] Adaptive Technologies Inc., 6424 West Chisum Trail, Glendale, AZ 85310
[2] Instituto de Ciencias Nucleares, UNAM, A. Postal 70-543 México D.F. 04510
[3] eXa Inc., 51 East 42nd Street, Suite 602, New York, NY 10017
[4] Boston University School of Medicine
[5] DxCG Inc., Boston MA

1 Introduction

In the battle to control escalating health care costs, predictive models are increasingly employed to better allocate health care resources and to identify the "best" cases for preventive case management. In this investigation we predicted the top 0.5% most costly cases for year $N + 1$, given a population in year N, with data for the period 1997-2001 taken from the MEDSTAT Marketscan Research Database for a cohort of privately insured individuals diagnosed with diabetes. We considered two performance metrics: i) classification accuracy, i.e. the proportion of correctly classified persons in the top 0.5% and ii) the total number of dollars associated with the predicted top 0.5% of most costly cases.

2 Methodology

The fundamental objects of interest are $P(\text{top } 0.5\%|\mathbf{X}_i)$ - the conditional probabilities to be in the top 0.5% cost category given a certain attribute vector $\mathbf{X}_i$ with components X_{ij}, where j runs over the values of the attribute i and $*$, where $*$ denotes a sum over all attribute values. We used 184 medical attributes associated with Diagnostic Cost Groups (DCGs) [2], which give a complete classification of medical conditions, and a set of quarterly cost data consisting of a further 30 variables. A GA was used to search for "fit" classifiers, fitness being measured by $\epsilon = N_{X_i}(P(\text{top } 0.5\%|\mathbf{X}_i) - P(\text{top } 0.5\%))/(N_{X_i}P(\text{top } 0.5\%)(1 - P(\text{top } 0.5\%)))^{1/2}$, where N_{X_i} is the number of individuals in the data with attribute vector $\mathbf{X}_i$. For the GA we determined the following optimal parameter values: population = 100, no. of generations = 50, mutation rate = 0.1 and crossover rate = 1. The fittest 100 classifiers over a set of runs were filtered out. This filtered list was then sorted according to a score function, S_1. Finally, a score, S_2, was assigned to an individual by an assignment algorithm between the classifiers and the individual. The highest scoring top 0.5% according to S_2 for year N was the prediction for year $N + 1$ and was compared with taking the highest ranked cases based on each of two widely-used benchmarks in health care modeling: 1) year N cost [1]; and 2) "out-of-the box" year-N DCG prospective risk scores [2].

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1330–1331, 2004.

3 Results

Best results were obtained using for S_1 - Winner-rerank-reevaluation. In this case the first, fittest, classifier is fixed. Individuals corresponding to this classifier are removed from the other classifiers in the list, the fitness recalculated and the list reranked. Passing to the subsequent classifiers this reevaluation procedure is iterated until one reaches the final classifier in the list. This procedure helps prevent poor classifiers from "hitchhiking" on the back of fitter ones. Finally, the final list is reranked using $P(\text{top } 0.5\%|\mathbf{X}_i)$ directly rather than ϵ. For S_2 a simple "winner-takes-all" assignment strategy was used, where an individual was assigned a score that was the final score of the winning classifier after the reevaluation and reranking procedure.

N		Benchmark 2	Benchmark 1	Score function
1997	# correct	29	31	52.5
	% correct	20	21.3	36.2
	$	11.7	11.4	15.9
1998	# correct	35	34	53.5
	% correct	18	17.5	27.6
	$	14.4	12.7	17.0
2000	# correct	82	93	132.1
	% correct	18.2	20.7	29.4
	$	35.6	35.3	46.6

We see above both in- and out-of-sample results. In-sample data were used to determine a set of optimal classifiers from predicting 1998 costs with N = 1997 data. These classifiers were then used for predicting the out-of-sample years 1999, using N = 1998 data, and 2001, using N = 2000 data. The results are averages over 10 runs. Both performance measures are shown - number/percentage of correctly identified individuals in the top 0.5% of year N + 1 costs and the dollar amount ($ - millions) of costs associated with the predicted group.

The GA-discovered classifiers gave average out-of-sample improvements of 47% and 59% in predictive accuracy for individuals in the top 0.5% of next year costs over benchmarks 1 and 2 respectively - these being the most common in the industry. There were also significant improvements in the total dollar amount. The classifier system is also ideal for identifying important drivers of costs as that is precisely what the genetic search through the classifier space is determining.

References

1. W.F. Bluhm and S. Koppel, *Individual Health Insurance Premiums,* In *Individual Health Insurance,* ed. F.T. O'Grady, 59-61, Society of Actuaries, Schaumberg IL, (1988).
2. A. Ash, R.P. Ellis, G.C. Pope, J.Z. Ayanian, D.W. Bates, H. Burstin, L.I. lezzoni, E. McKay and W. Yu, *Using Diagnoses to Describe Populations and Predict Costs, Health Care Financing Review* **10** (4), 17-29 (2000).

Generating Compact Rough Cluster Descriptions Using an Evolutionary Algorithm

Kevin Voges and Nigel Pope

School of Management, University of Canterbury, Christchurch, New Zealand
Kevin.Voges@canterbury.ac.nz
School of Marketing, Griffith University, Nathan Campus, Brisbane, Australia
N.Pope@griffith.edu.au

1 Cluster Analysis

Cluster analysis is a technique used to group objects into clusters such that similar objects are grouped together in the same cluster. Early methods were derived from multivariate statistics. Some newer methods are based on rough sets, introduced by Pawlak [3], [4]. An extension of rough sets to rough clusters was introduced in [5]. The lower approximation (LA) of a rough cluster contains objects that only belong to that cluster, and the upper approximation (UA) contains objects that may belong to more than one cluster. An EA can be used to find a set of lower approximations of rough clusters that provide the most comprehensive coverage of the data set with the minimum number of clusters.

2 Rough Clustering Algorithm

The building block of the data structure is the *template* [2]. Let $S = (U, A)$ be an information system, where U is the set of data objects, and A is the set of attributes of the data objects. Any clause of the form $D = (a \in Va)$ is called a *descriptor,* and the value set Va is the *range* of D. A *template* (T) is a conjunction of unique descriptors defined over attributes from $B \subseteq A$. That is, $T = \Lambda_{a \in B}(a \in Va)$ is a *template* of S. The data structure acted on by the EA is a cluster solution, C, which is defined as any conjunction of k unique templates. This data structure was encoded as a simple two-dimensional array with a variable length equal to the number of unique templates in the cluster solution and a fixed width equal to the number of attributes being considered.

A number of considerations determine the fitness measure. Firstly, the algorithm maximizes the data set coverage C, defined as the fraction of the universe of objects that matches the set of templates in C. Secondly, the algorithm minimizes k, the number of templates in C. Finally the accuracy a, of each template needs to be maximized [4]. This is the sum of the cardinal value of the LA divided by the cardinal value of the UA defined by each template in C. The *fitness* value, f, of each cluster solution is defined as the coverage multiplied by the accuracy divided by the number of templates in C.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1332–1333, 2004.

A multi-point recombination operator was used to generate valid rough cluster solutions. Templates are randomly selected from each parent, and added to the offspring after checking that they are unique to the cluster solution. Two mutation operators were used. One randomly sampled a unique template from the list of valid templates and added it to the cluster solution, and the other randomly removed a template from the cluster solution. Repair operators were not required.

3 Application

The technique was used to analyze data from a study of beer preferences in young adults, who were asked to evaluate possible attributes to be considered when making a purchasing decision. Five attributes were used: image, packaging, price, alcohol content, and place sold.[1] A rough cluster analysis was conducted, partitioning the study participants into distinct clusters based on which attributes were considered important. The best cluster solution obtained achieved coverage of 94.8% of the data set using 13 templates. The accuracy of each template ranged from 0.50 to 1.00. This EA-based technique was able to find a rough cluster solution that covered a large percentage of the data set with a small number of templates. The technique also overcame some limitations of previous techniques, such as those that require the number of clusters be specified in advance [1], or those that generate too many clusters to be easily interpretable [5].

References

1 Lingras, P.: Rough Set Clustering for Web Mining. In: *Proceedings of 2002 IEEE International Conference on Fuzzy Systems.* (2002)

2 Nguyen, S. H.: Regularity Analysis and its Applications in Data Mining. In: Polkowski, L., Tsumoto, S., Lin, T. Y. (eds.): *Rough Set Methods and Applications: New Developments in Knowledge Discovery in Information Systems.* Physica-Verlag, Heidelberg New York (2000) 289 - 378

3 Pawlak, Z.: Rough Sets. *International Journal of Information and Computer Sciences.* 11:5 (1982) 341 - 356

4 Pawlak, Z.: *Rough Sets: Theoretical Aspects of Reasoning About Data.* Kluwer, Boston (1991)

5 Voges, K. E., Pope, N. K. Ll., Brown, M. R.: Cluster Analysis of Marketing Data Examining On-line Shopping Orientation: A Comparison of *K*-means and Rough Clustering Approaches. In: Abbass, H.A. Sarker, R.A., Newton, C.S. (eds.): *Heuristics and Optimization for Knowledge Discovery.* Idea Group Publishing, Hershey, PA (2002) 207-224

[1] Our thanks to Michael Veitch and Kevin Nicholson for providing this data

An Evolutionary Meta Hierarchical Scheduler for the Linux Operating System

Horst F. Wedde, Muddassar Farooq, and Mario Lischka

Informatik III, University of Dortmund,
44221, Dortmund, Germany
{wedde, farooq, lischka}@ls3.cs.uni-dortmund.de

Abstract. The need for supporting CSCW applications with *heterogeneous and varying user requirements* calls for adaptive and reconfigurable schedulers accommodating a mixture of real-time, proportional share, fixed priority and other policies, thus overcoming frustrating processor bottlenecks. In this paper we try to overcome this anomaly by proposing an *evolutionary strategy* for a *Meta Hierarchical Scheduler (MHS)* in which a *user is actively involved* in the design cycle of a scheduler. Our framework analyzes *user requirements* by formulating an *Abstract Scheduler Model (ASM)* for an optimum scheduler with the help of an evolutionary algorithm that satisfies the needs. Finally a C source code for this MHS is generated for the Linux kernel. Our experimental results demonstrate that our MHS enhances through the evolutionary approach, the user satisfaction level by a factor of two as compared to the satisfaction level achieved by the standard Linux scheduler.

1 The Evolutionary Algorithm in SAADI

The emergence of the Internet market toward the end of the last century has revolutionized the traditional use of PC. Nowadays a user in a CSCW environment attends a multimedia business meeting, take notes for minutes of the meetings and reads/sends important documents at the same time. In CSCW applications the processor usage pattern becomes frequently unpredictable because of changing user needs. Conflicting but coexisting scheduling requirements make it an uphill and difficult task to create a generic scheduler that could easily, efficiently if not optimally, adapt to the *user needs* in collaborating communities.
In SAADI framework we incorporate the *user needs* in the design cycle of a scheduler [2]. By using the constructs of SAADI-SDL, a scheduler description language, a user could specify the processing requirements of different applications and of their relative importance. Currently, the schedulable application set consists of single/multi-threaded batch, periodic (multimedia and network applications) and interactive applications. Our ASM module parses SAADI-SDL file and analyzes the requirements to generate scheduler individuals for the first generation of our evolutionary algorithm with the help of a context-free grammar. Figure 1 shows the complete design workflow in our SAADI framework. A scheduler individual consists of a hierarchical organization of simple schedulers as proposed by Regehr [1].

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1334–1335, 2004.

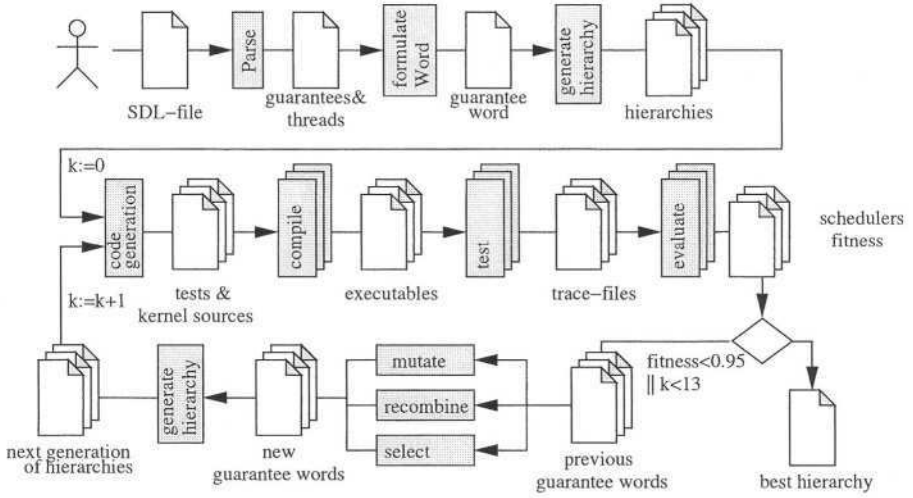


Fig. 1. Design Workflow in SAADI

2 Evaluation of Results and Future Work

We designed and developed a testing infrastructure to validate our approach. Here, we report the results for one SDL file in which the test user specified 7 interactive, 4 periodic and 4 batch threads. Our SAADI framework found an individual whose fitness value was 0.81. We also tested the standard Linux 2.5.58 scheduler under the same above-mentioned load . The fitness value of standard Linux scheduler was 0.33 on the average. This shows the superiority of SAADI over the standard Linux scheduler because the best performing individual of SAADI has 0.81 fitness value which is more than two times better than the standard Linux scheduler. Hence, this first step proved the conceptual validity of our model.

Currently SAADI provides an off-line interaction to a user. The user specifies his requirements using SDL, and then SAADI invokes a MHS generation process and the evolutionary algorithm to find an optimum MHS. In near future we intend to transform the framework into a real-time on-line module inside a kernel and provide easy system calls and a graphical user interface (GUI) for an administrative user to interact with the system.

References

1. John Regehr and John A. Stankovic. HLS: A framework for composing soft real-time schedulers. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS 2001),* pages 3–14, London, UK, December 2001.
2. Horst F. Wedde, Muddassar Farooq, and Mario Lischka et. al. SAADI–An Integrated Approach of Adaptable Schedulers. Technical report-pg424, School of Computer Science, University of Dortmund, 2003.

An Evolutionary Algorithm for Parameters Identification in Parabolic Systems*

Zhijian Wu[1], Zhilong Tang[1], Jun Zou[2], Lishan Kang[1], and Mingbiao Li[3]

[1]The State Key Lab. of Software Eng., Wuhan Uni., Wuhan, China
zjwu@public.wh.hb.cn
[2]Dept. of Math., The Chinese Univ. of Hong Kong, Hong Kang, China
[3]School of Computer Sci. and Eng., Wenzhou Normal College, Zhejiang, China

Abstract. In this paper we construct a novel evolutionary algorithm. It yields good performance on a collection of parabolic parameter identification problems. The algorithm has a good tolerability for the noise in the observed data. Even when the noise level is up to 10% we can also get such a good result.

1 Description of Problem

Let's consider the following parabolic problem:

$$\begin{cases} L(q)u = \dfrac{\partial u}{\partial t} - \dfrac{\partial u}{\partial x}(q(x)\dfrac{\partial u}{\partial x}) = f(x,t), & (x,t) \in (0,1)\times(0,T) \\ u(x,0) = u_0(x), & x \in (0,1) \\ u(0,t) = u(1,t) = 0, & t \in (0,T) \end{cases}$$

Parameters identification is the process to find the potential solution $q^*(x)$ that makes $u_{q^*}(x,t)$ match the observed data of u(x, t) as optimally as possible.

The interval [0,1] is divided equally into n parts, the step size h= 1/n, and mesh point $x_i = ih$ (i=0, 1,2,...,n). Suppose we have the observed values $u_i^{(o)}$ at points (x_i, T), $\vec{u}_{ob} = (u_1^{(o)}, u_2^{(o)}, \ldots, u_{n-1}^{(o)})$. We consider the case in which $q(x)$ is continuous and smooth. We adopt Hat Functions $\varphi_1(x), \varphi_2(x), \ldots, \varphi_n(x)$ as the basis of $q(x)$, then $q(x)$ can be expressed as $q_h(x) = \sum_{i=0}^{n} q_i \varphi_i(x)$. .Denote $\vec{q} = (q_0, q_1, \ldots, q_n)$. The parameters identification problem can be transferred to the optimization problem: $\min_{q_u} fitness(\vec{q}) = \min_{q}(h \| \vec{u}_q - \vec{u}_{ob} \|_2^2 + \frac{\beta}{h}\sum_{i=1}^{n}(q_i - q_{i-1})^2)$.

* This work was supported by Open Fund of State Key Laboratory of Software Engineering, Wuhan University, National Natural Science Foundation of China (Nos. 60133010, 60073043, 70071042)

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1336–1337, 2004.

2 Description of Algorithm

Step 1. N individuals $\vec{q}^{(1)}, \vec{q}^{(2)}, \ldots, \vec{q}^{(N)}$ are randomly and uniformly produced in the search space to form initial population $P(0)$, evaluate $P(0)$. Set t=0;

Step 2. While t> *Max-Generation* go to Step 5.

Step 3. Select M individuals $\vec{q}^{(1)}, \vec{q}^{(2)}, \ldots, \vec{q}^{(M)}$ randomly from $P(t)$ to form a sub space $V = \left\{ \vec{q} / \vec{q} \in D^{n+1}, \vec{q} = \sum_{i=1}^{M} a_i \vec{q}^{(i)} \right\}$, where a_i satisfies the condition $\sum_{i=1}^{M} a_i = 1$, $a_i \in [-0.5, 1.5]$, produce a new individual $\vec{q}^{(new)}$ randomly in V, for i=1 to n-1 do $q_i^{(new)} = (q_{i-1}^{(new)} + q_i^{(new)} + q_{i+1}^{(new)})/3$, if $\vec{q}^{(new)}$ is better than $\vec{q}^{(worst)}$, then substitute $\vec{q}^{(new)}$ for $\vec{q}^{(worst)}$, where $\vec{q}^{(worst)}$ is the worst individual in $P(t)$;

Step 4. Select an individual $\vec{q}^{*}$ randomly from the population and an element q_i^{*} randomly from $\vec{q}^{*}$, conduct $q_i^{*} = \alpha_l + (\alpha_u - \alpha_l) \bullet rand()$, where *rand*() is a random number in [0,1], get a new individual $\vec{q}^{(mutatiom)}$. If $\vec{q}^{(mutatiom)}$ is better than $\vec{q}^{(worst)}$, then substitute $\vec{q}^{(mutatiom)}$ for $\vec{q}^{(worst)}$; t=t+1;

Step 5. Output the best individual.

3 Numerical Experiments

In the experiments, we get observed values of $u(x)$ by adding some noise to the function $u(x)$, $\vec{u}_{ob} = (1 + \delta \bullet rand(x)) \bullet \vec{u}(x)$, where *rand*($x$) is an uniformly distributed random function in [-1,1], and δ is the noise level parameter. Parameters set: N=200, M=10, *Max-Gen*=10000, $\beta = 10^{-6}$, $\alpha_l = 0.001$, $\alpha_u = 10.0$

Test Problem: $u(x) = \sin(2\pi x), q(x) = 3 + 2x^2 - 2\sin(2\pi x)$.

We conducted three experiments for noise level parameter δ=1%,5% and 10%. The result of the experiments is given as follows.

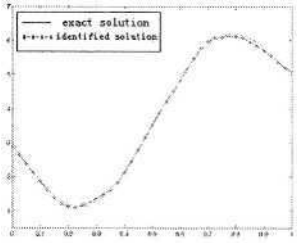


Fig. 1. δ =1%

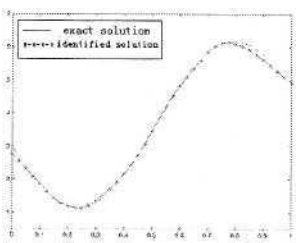


Fig. 2. δ =5%

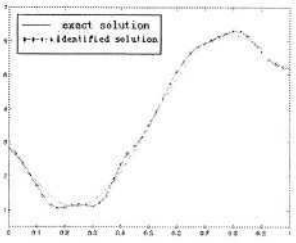


Fig. 3. δ =10%

How to Overcome the Equivalent Mutant Problem and Achieve Tailored Selective Mutation Using Co-evolution

Konstantinos Adamopoulos, Mark Harman, and Robert M. Hierons

Department of Information Systems and Computing,
Brunel University,
Uxbridge, Middlesex, UB8 3PH, UK
konstantinos.adamopoulos@brunel.ac.uk

Abstract. The use of Genetic Algorithms in evolution of mutants and test cases offers new possibilities in addressing some of the main problems of mutation testing. Most specifically the problem of equivalent mutant detection, and the problem of the large number of mutants produced. In this paper we describe the above problems in detail and introduce a new methodology based on co-evolutionary search techniques using Genetic Algorithms in order to address them effectively. Co-evolution allows the parallel evolution of mutants and test cases. We discuss the advantages of this approach over other existing mutation testing techniques, showing details of some initial experimental results carried out.

1 Introduction

Software testing is a vital yet expensive part of the software development process. Studies suggest that it often consumes in the order of fifty percent of the total development budget [1,2]. Thus, approaches that automate or semi-automate sections of software testing may significantly improve the efficiency and quality of the software development process.

1.1 Overview of Mutation Testing

Mutation testing is a software testing technique originally proposed by Hamlet [3]. Mutation testing is based upon seeding the implementation (original program) with a fault *(mutating it),* by applying a *mutation operator,* and determining whether testing identifies this fault. A mutated program is called a *mutant* and if a test case distinguishes between the mutant and the original program it is said to *kill* the mutant. Given a set of test cases, if no test case can distinguish between the mutant and the original program then the mutant is still *live.*

Mutation testing may be used to judge the effectiveness of a test set: the test set should kill all the mutants. Similarly, test generation may be based on mutation testing: tests are generated to kill the mutants. Interestingly, many test

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1338–1349, 2004.

criteria may be represented using mutation testing by simply choosing appropriate mutation operators. Test sets are measured according to *mutation score,* or *adequacy score,* which is defined as the total number of killed mutants over the number of non-equivalent mutants, where a mutant is said to be *equivalent* if there does not exist a test case which can distinguish the output of the mutant from the output of the original program. Mutation score takes real values between 0.0 and 1.0, where 1.0 is the best score possible, meaning that this particular test set can kill all the non-equivalent mutants. Such a test set is said to be 100% mutation adequate.

Due to the fact that mutation testing is of high computational cost, even in the case of small and rather simple programs, several techniques were developed to reduce considerably the computational cost of effectiveness. Among these techniques are selective mutation [4], mutant sampling [5], weak mutation [6,7], schema-based mutation [8], separate compilation [9], and search. In this paper we examine some weak points of the 'do fewer' techniques, such as selective mutation and mutant sampling, and we introduce a new, search based, methodology based on Genetic Algorithms (GAs).

Selective mutation identifies the critical mutation operators that provide almost the same testing coverage as non-selective mutation. A number of mutants are rejected, because low performance mutation operators are rejected. As a result, the number of mutants is considerably decreased thereby reducing computational cost.

In a similar manner, mutant sampling is another approach to reduce the number of mutants under consideration. The main concept of this approach is the random selection of a subset of mutants; either by using samples of some *a priori* fixed size, or until sufficient evidence has been collected indicating that a statistically appropriate sample size has been reached [4,5].

The main assumption of the present work is that the 'do fewer' approaches (selective mutation and mutant sampling) have some weak points: Selective mutation does not take into consideration the fact that there are different sets of critical mutation operators for different classes of programs. On the other hand, mutant sampling is an approach that works satisfactorily with small randomly selected subsets of mutants but the testing coverage achieved is reduced [10].

While mutation testing is a powerful and general technique, a number of associated problems have limited its practical impact.

1. One of these problems is that the standard set of mutation operators may lead to a vast number of mutants, not always appropriate ones. Typically there is a large number of mutants for even small software units. Because of this, mutation testing is considered to be a computationally expensive method; all mutants must be tested against all test cases, leading to an increasing demand for computational resources. However, it has been noted that this problem can be reduced by an appropriate choice of mutation operators, using selective mutation [4].
2. Another important problem with mutation testing is that a mutant, while syntactically different from the initial program, may have the same behaviour

as the program. Clearly such *equivalent* mutants can never be killed and thus it is important, but often difficult, to identify them. Some approaches to detecting equivalent mutants and reducing the number of equivalent mutants produced have been introduced [11,12,13,14,15]. Manual equivalent mutant detection is tedious and new methodologies to automate this process should be introduced.

The contributions of this paper are to:

- Address the problem of selecting the effective mutants and test cases by searching for them.
- Reduce the large number of mutants produced and the associated computational cost, using natural selection, rather than artificial selection.
- Show how the fitness function can be designed to avoid generation of equivalent mutants.
- Explore how co-evolution can be used to automatically generate and evolve both mutants and test cases.

The first author to consider the application of genetic algorithms to mutation testing was Bottaci [16,17]. Many other authors have also suggested search for test data generation for other forms of testing [18,19,20,21,22,23,24,25,26,27,28]. However, this is the first paper to introduce the idea of co-evolution for mutation testing. A recent survey of work on evolutionary test data generation is provided by McMinn [29].

The problem of selection is a special case of the Feature Subset Selection Problem [30], where from an existing set of features there must be selected a subset of features that can achieve the same quality of properties. In our case to select the subsets of mutants and the corresponding subsets of test cases that provide the highest testing coverage possible. That is, as close as possible to 100% mutation adequacy.

In the present work we will present a three step methodology. The first step is to utilize a GA for the evolution of the mutants of a given program. The second step, similarly, is to utilize a GA for the evolution of the test cases of a given program. And finally, the third step which refers to the use of GAs for the co-evolution of both a population of mutants and a set of test cases for a given program.

The rest of the paper is organized as follows: Section 2 describes the proposed methodology. Section 3 presents some results of the experiments carried out. Section 4 outlines some of the outstanding issues for future consideration. Finally section 5 presents the conclusions.

2 Methods

In this section the three aspects of our methodology will be presented, namely the evolution of subsets of mutants against a fixed set of test cases, the evolution of subsets of test cases against a fixed set of mutants, and the co-evolution of both populations by evolving in parallel subsets of mutants and subsets of test cases.

2.1 Evolution of Subsets of Mutants Against a Fixed Set of Test Cases

In order to evolve subsets of mutants that are non-equivalent and at the same time difficult to kill first we generate a pool of mutants of the original program under test. From this pool our candidate mutants will be drawn. This is done by using a simulation of a mutation testing tool such as Mothra [31].

Each member of the constructed pool of mutants has been validated against a fixed set of test cases. This validation procedure associates each mutant with a corresponding score that is related to the performance of the mutant against a given fixed set of test cases. The higher the score, the more difficult it is for a mutant to be killed. In this way the score is a measure of the ability of the mutant to avoid being killed by the test cases. This validation procedure has been simulated. The simulation generates random real numbers between 0.0 and 1.0; each number represents a score associated with a mutant.

The main idea is to construct subsets of these mutants, each subset to be considered as an individual of the genetic algorithm. Therefore the initial population of the genetic algorithm consists of subsets of mutants that are randomly selected from the pool of mutants. The simulation does this by grouping a number of scores together; each group of scores correspond to a subset of mutants.

Evolution of this population with a GA will lead to subsets of mutants that are combined to give the maximum total score. This is done by selecting the fitness function of an individual to be the sum of the scores of the mutants divided by the length of the individual. That is, if S_i is the score of a mutant i and the individual consists of L mutants the fitness Mf of this individual is given by:

$$Mf = \begin{cases} \frac{\sum_{i=1}^{L} S_i}{L} & \text{if } \forall i. S_i \neq 1. \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

It is obvious that if there exists i such that $S_i = 1$, then there is a mutant killed by no test cases. This may be equivalent, that's why in this case the fitness function of this particular set of mutants is penalized by assigning to it the value 0. Also, the normalization of the fitness values, in order to get real values between 0.0 and 1.0, makes the fitness of individuals with different length comparable.

The fitness function, as described here, is relatively easy to optimise (even without a search technique). However, the evolution of subsets of mutants against a fixed set of test cases is only a preliminary milestone in the road towards the ultimate goal of achieving full co-evolution of both test set and mutant set. See Section 2.3.

We use uniform crossover. The mutation operator exchanges a single mutant of an individual with another mutant randomly selected from the pool of mutants. The selection operator is the roulette-wheel selection that ensures that the probability of selection of an individual is proportional to its fitness. Elitism was also introduced in the GA, that means a user defined number of highly fit individuals are automatically passed unchanged to the next generation.

By applying the GA we achieve to work with the whole set of mutation operators; there is no need to decrease the number of mutation operators, as in selective mutation. At the same time the GA achieves tailored selective mutation by genetically selecting sets of mutants with higher fitness values. That is, a higher chance of avoiding being killed. Additionally possible equivalent mutants are detected and eliminated.

2.2 Evolution of Subsets of Test Cases Against a Fixed Set of Mutants

We can apply the same methodology to the evolution of subsets of test cases as we applied to evolution of mutants. First, we randomly generate a pool of test cases from which our candidate test sets will be drawn. Each test case is a sequence of random appropriately typed values that can be used as input for the program under test.

Each member of the constructed pool of test cases has been validated against a fixed set of mutants. This validation procedure associates each test case with a corresponding mutation score that is related to the performance of the test case against a given fixed set of mutants. The higher the mutation score the more effective is a test case in killing mutants, so, in this way the score is a measure of the ability of the test case to kill mutants. This validation procedure has been simulated. The simulation generates random real numbers between 0.0 and 1.0; each number represents a mutation score associated with a test case.

The main idea is to construct subsets of these test cases, each subset to be considered as an individual of the genetic algorithm. Therefore the initial population of the genetic algorithm consists of subsets of test cases that are randomly selected from the pool of test cases. The simulation does this by grouping a number of mutation scores together; each group of mutation scores correspond to a subset of test cases.

Evolution of this population with the GA, will lead to subsets of test cases that are combined to give the maximum total mutation score. This is done by selecting the fitness function of an individual to be the sum of the mutation scores of the test cases divided by the length of the individual. That is, if MS_i is the mutation score of a test case i and the individual consists of L test cases the fitness Tf of this individual is given by:

$$Tf = \frac{\sum_{i=1}^{L} MS_i}{L} \tag{2}$$

The normalization of the fitness values, in order to get real values between 0.0 and 1.0, makes the fitness of individuals with different length comparable. As for the mutants, the fitness function is relatively easy to optimise. However, the presented here technique is a useful step towards the ultimate goal of achieving full co-evolution of both test set and mutant set.

Similarly with the GA for the mutants above, we use uniform crossover. This well-known crossover operator is applied to the individuals of the population.

Each test case is a gene of the individual and each individual is a set of test cases. The mutation operator refers to exchange of a test case of an individual with another test case randomly selected from the pool of test cases. The selection operator is the roulette-wheel selection that ensures that the probability of selection of an individual is proportional to its fitness.

Elitism is defined in the same way as for the previous GA in Section 2.1; a user defined number of high performance sets of test cases pass automatically to the next generation.

By applying this GA we work with a large set of test cases, and at the same time genetically select sets of test cases with higher fitness values. That is, higher ability in killing mutants and achieving high mutation scores. This method can automatically generate high performance test cases for an original program under test.

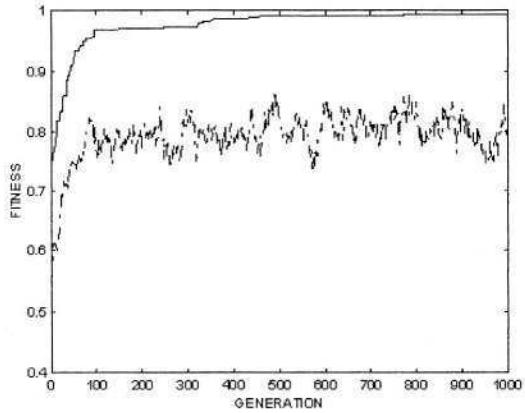


Fig. 1. GA performance for a standard set of parameter values used as a reference.

2.3 Co-evolution of Sets of Mutants and Sets of Test Cases

The concept of co-evolution in evolutionary computing is inspired by nature, where organisms that are ecologically intimate, for example, predators and prey, or hosts and parasites, influence each other's evolution. Co-evolution entails a change in the genetic composition of one species (or group) in response to a genetic change in another. This approach has been chosen to be part of our methodology for two main reasons:

1. **We achieve selective mutation.** Previous work has shown selection of mutation operators to be a productive way to reduce the (often infeasibly

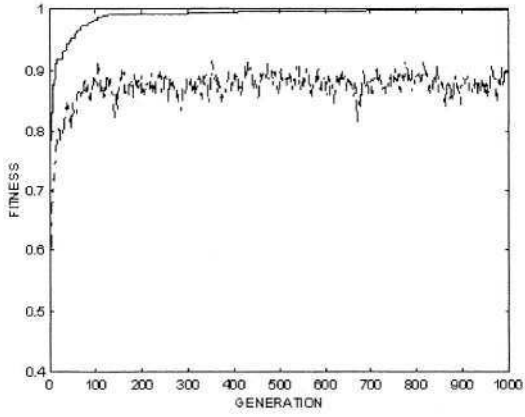


Fig. 2. Improved GA performance for higher elitism value, increased from 2 to 10,

large) number of mutants. However previous work requires that the mutation operators are selected *a priori.* Using evolution we are able to select individual mutants, tailored to the program under test, based upon their individual fitness.

2. **Our fitness function guarantees not to generate equivalent mutants.** The fitness function for mutants requires that at least one test case kills the mutant. Those mutants which are killed by no test cases, receive a very low value of fitness. This means that mutants are guaranteed to be non-equivalent. It may be that some *stubborn* (hard to kill) mutants which are not equivalent are also given low fitness scores. It is hoped that the robustness of the evolutionary search strategy will ensure that such stubborn mutants are rediscovered. That is, as the co-evolving test set improves its fitness, a point will be reached where the test set can kill the stubborn mutant. The authors believe that the ability to guarantee that all mutants are non-equivalent is an important goal. It justifies the 'aggressive' approach adopted here, because the presence of equivalent mutants makes mutation testing infeasible.

These two properties of our approach aim to address the two fundamental barriers to wider uptake of mutation testing, specifically:

1. Too many mutants are generated.
2. Too much effort is spent on identifying equivalent mutants.

In this paper the concept of co-evolution is based on the idea that two populations, that is the one consisting of individuals of mutants, and the one consisting of individuals of test cases, are evolved in parallel with a specifically designed GA. On each generation of the GA, the fitness of the individuals of these two

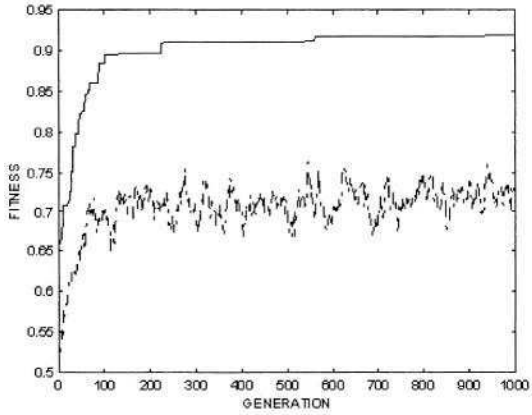


Fig. 3. Lower GA performance for higher individual's length, increased from 10 to 30.

populations are recalculated considering the alteration of the individuals of these two populations.

In other words the score of each mutant is re-evaluated with respect to the present population of the test cases, and on the other hand the score of each test case is re-evaluated with respect to the present population of mutants. Co-evolution of these two populations is expected to lead to individuals consisting of mutants that are very difficult to be killed and individuals consisting of test cases that are capable of killing high quality mutants.

3 Results and Discussions

This section presents simulated results for the GA that evolves sets of mutants. A pool of N mutants was generated and evaluated. Figure 1 shows the performance of the GA for N=10,000, constant population size POP of 100, individual's length L=10 (10 mutants in each set), crossover probability Pc=1.0, mutation probability Pm=0.05, and elitism E=2. The GA ran for 1000 generations and the results presented are averaged over 10 runs for each GA.

The solid line shows the evolution of the fitness of the fittest individual of each generation, whereas the dotted line represents the evolution of the mean fitness of each generation.

By increasing the value of elitism from 2 to 10, leaving the values of the rest of the parameters the same, we obtain the results shown in Figure 2.

We observe that in this case the algorithm converges faster, and achieves a higher value of best fitness. This result introduces the idea of how elitism, or similar operators, can be used in the future as a kind of memory to improve the performance of the algorithm.

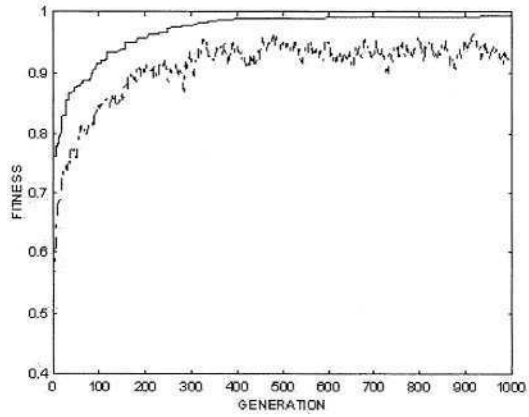


Fig. 4. Higher performance for the mean fitness of each generation when the mutation probability has decreased from 0.05 to 0.01.

Figure 3 shows the performance of the GA when the length of each individual has increased to 30 genes. The values of the rest of the parameters are the same as for Figure 1. In this case the algorithm converges more slowly and does not perform as well. It appears that the number of genes forming an individual is a critical parameter for the performance of the GA.

Figure 4 illustrates the performance of the GA when the mutation probability has decreased from 5% to 1%, while the values of the other parameters are the same as in Figure 1. The algorithm achieves an increase in the effectiveness of the average fitness of each generation.

4 Future Work

In future work, more simulated results from all the GAs will be presented. These results will be compared with those obtained from a real mutation testing tool, which is currently under development.

In addition more results will be presented concerning the performance of the GAs and the influence of GA parameters (population size, crossover probability, choice between one-point crossover and uniform crossover, mutation probability, length, selection scheme, elitism) on that performance.

Moreover elitism will be expanded to offer a kind of memory of the best individuals. The influence of this memory on GA performance will be further analysed.

Additionally a comparative analysis with selective mutation and mutation sampling will be presented.

Finally the technique used for elimination of equivalent mutants will be analysed in order to examine if this technique results in killing also stubborn mutants.

5 Conclusion

In this paper we addressed two main problems of mutation testing. First, the problem of equivalent mutant detection; we showed how to design a fitness function in such a way that possible equivalent mutants can be detected. Second, the problem of the large number of mutants produced; we showed how to select effective mutants and test cases of an original program. We approached these problems as search problems using probabilistic, meta-heuristic algorithms such as Genetic Algorithms and co-evolution.

Our main motivation was to overcome the twin difficulties of the infeasibly large number of generatable new mutants and the problem of weeding out equivalent mutants. The proposed method does not reject mutation operators like the method of selective mutation, nor does it decrease the number of mutants by selecting a random sample of the mutants. Instead the proposed method generates a pool of mutants that can be generated from an original program, after the application of all the mutation operators. As a second step the method utilizes a GA that evolves subsets of mutants and therefore genetically rejects irrelevant and low performance mutants.

In this way we overcome one of the principal disadvantages of selective mutation; false generalisation for the rejection of mutation operators.

In the same manner genetic evolution on sets of test cases is proposed as a method for increasing their testing ability according to their adequacy score.

Finally these two methods are incorporated in an algorithm for the co-evolution of both mutants and test cases in parallel. These two populations compete with each other and therefore better results are expected.

References

1. Myers, G.J.: The Art of Software Testing. Wiley - Interscience, New York (1979)
2. Pressman, R.: Software Engineering: A Practitioner's Approach. 3rd edn. McGraw-Hill Book Company Europe, Maidenhead, Berkshire, England, UK. (1992) European adaptation (1994). Adapted by Darrel Ince. ISBN 0-07-707936-1.
3. Hamlet, R.G.: Testing programs with the aid of a compiler. IEEE Transactions on Software Engineering **3** (1977) 279–290
4. Bottaci, L., Mresa, E.S.: Efficiency of mutation operators and selective mutation strategies: An empirical study. Software Testing, Verification and Reliability **9** (1999) 205–232
5. Budd, T.A.: Mutation analysis: Ideas, examples, problems and prospects. In: Proceedings of the Summer School on Computer Program Testing, Sogesta (1981) 129–148
6. Howden, W.E.: Weak mutation testing and completeness of test sets. IEEE Transactions on Software Engineering **8** (1982) 371–379
7. Woodward, M.R., Halewood, K.: From weak to strong, dead or alive? an analysis of some mutation testing issues. In: Proceedings of the Second Workshop on Software Testing, Verification, and Analysis, Banff, Canada (1988)
8. Untch, R.H., Offutt, A.J., Harrold, M.J.: Mutation analysis using mutant schemata. In Ostrand, T., Weyuker, E., eds.: Proceedings of the 1993 International Symposium on Software Testing and Analysis (ISSTA). (1993) 139–148

9. Byoungju, C., Mathur, A.P.: High-performance mutation testing. The Journal of Systems and Software **20** (1993) 135–152
10. Offutt, A.J., Untch, R.: Mutation 2000: Uniting the orthogonal. In Wong, W.E., ed.: Mutation Testing for the New Century (proceedings of Mutation 2000), San Jose, California, USA, Kluwer (2001) 45–55
11. Baldwin, D., Sayward, F.: Heuristics for determining equivalence of program mutations. Research Report 276, Department of Computer Science, Yale University (1979)
12. Hierons, R.M., Harman, M., Danicic, S.: Using program slicing to assist in the detection of equivalent mutants. Software Testing, Verification and Reliability **9** (1999) 233–262
13. Offutt, A.J., Craft, W.M.: Using compiler optimization techniques to detect equivalent mutants. Software Testing, Verification and Reliability **4** (1994) 131–154
14. Offutt, A.J., Pan, J.: Detecting equivalent mutants and the feasible path problem. In: Annual Conference on Computer Assurance (COMPASS 96), IEEE Computer Society Press, Gaithersburg, MD (1996) 224–236
15. Offutt, A.J., Pan, J.: Automatically detecting equivalent mutants and infeasible paths. Software Testing, Verification, and Reliability **7** (1997) 165–192
16. Bottaci, L.: Instrumenting programs with flag variables for test data search by genetic algorithms. In Langdon, W.B., Cantú-Paz, E., Mathias, K., Roy, R., Davis, D., Poli, R., Balakrishnan, K., Honavar, V., Rudolph, G., Wegener, J., Bull, L., Potter, M.A., Schultz, A.C., Miller, J.F., Burke, E., Jonoska, N., eds.: GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, New York, Morgan Kaufmann Publishers (2002) 1337–1342
17. Bottaci, L.: Predicate expression cost functions to guide evolutionary search for test data. In Cantú-Paz, E., Foster, J.A., Deb, K., Davis, D., Roy, R., O'Reilly, U.M., Beyer, H.G., Standish, R., Kendall, G., Wilson, S., Harman, M., Wegener, J., Dasgupta, D., Potter, M.A., Schultz, A.C., Dowsland, K., Jonoska, N., Miller, J., eds.: Genetic and Evolutionary Computation – GECCO-2003. Volume 2724 of LNCS., Chicago, Springer-Verlag (2003) 2455–2464
18. Ferguson, R., Korel, B.: The chaining approach for software test data generation. ACM Transactions on Software Engineering and Methodology **5** (1996) 63–86
19. Jones, B., Sthamer, H.H., Eyres, D.: Automatic structural testing using genetic algorithms. The Software Engineering Journal **11** (1996) 299–306
20. Jones, B.F., Eyres, D.E., Sthamer, H.H.: A strategy for using genetic algorithms to automate branch and fault-based testing. The Computer Journal **41** (1998) 98–107
21. Michael, C., McGraw, G., Schatz, M.: Generating software test data by evolution. IEEE Transactions on Software Engineering (2001) 1085–1110
22. Mueller, F., Wegener, J.: A comparison of static analysis and evolutionary testing for the verification of timing constraints. In: 4th IEEE Real-Time Technology and Applications Symposium (RTAS '98), Washington - Brussels - Tokyo, IEEE (1998) 144–154
23. Pargas, R.P., Harrold, M.J., Peck, R.R.: Test-data generation using genetic algorithms. The Journal of Software Testing, Verification and Reliability **9** (1999) 263–282
24. Pohlheim, H., Wegener, J.: Testing the temporal behavior of real-time software modules using extended evolutionary algorithms. In Banzhaf, W., Daida, J., Eiben, A.E., Garzon, M.H., Honavar, V., Jakiela, M., Smith, R.E., eds.: Proceedings of the Genetic and Evolutionary Computation Conference. Volume 2., Orlando, Florida, USA, Morgan Kaufmann (1999) 1795

25. Schultz, A., Grefenstette, J., Jong, K.: Test and evaluation by genetic algorithms. IEEE Expert **8** (1993) 9–14
26. Tracey, N., Clark, J., Mander, K.: The way forward for unifying dynamic test-case generation: The optimisation-based approach. In: International Workshop on Dependable Computing and Its Applications (DCIA), IFIP (1998) 169–180
27. Wegener, J., Grimm, K., Grochtmann, M., Sthamer, H., Jones, B.F.: Systematic testing of real-time systems. In: 4th International Conference on Software Testing Analysis and Review (EuroSTAR 96). (1996)
28. Wegener, J., Sthamer, H., Jones, B.F., Eyres, D.E.: Testing real-time systems using genetic algorithms. Software Quality **6** (1997) 127–135
29. McMinn, P.: A survey of evolutionary testing. (Software Testing, Verification and Reliability) To appear.
30. Kirsopp, C., Shepperd, M., Hart, J.: Search heuristics, case-based reasoning and software project effort prediction. In Langdon, W.B., Cantú-Paz, E., Mathias, K., Roy, R., Davis, D., Poli, R., Balakrishnan, K., Honavar, V., Rudolph, G., Wegener, J., Bull, L., Potter, M.A., Schultz, A.C., Miller, J.F., Burke, E., Jonoska, N., eds.: GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, New York, Morgan Kaufmann Publishers (2002) 1367–1374
31. DeMillo, R.A., Offutt, A.J.: Experimental results from an automatic test generator, acm Transactions of Software Engineering and Methodology **2** (1993) 109–127

Evaluating Evolutionary Testability with Software-Measurements

Frank Lammermann, André Baresel, and Joachim Wegener

DaimlerChrysler AG, Alt-Moabit 96a, 10559 Berlin, Germany
{Frank.Lammermann, Andre.Baresel,
Joachim.Wegener}@DaimlerChrysler.com

Abstract. Test case design is the most important test activity with respect to test quality. For this reason, a large number of testing methods have been developed to assist the tester with the definition of appropriate, error-sensitive test data. Besides black-box tests, white-box tests are the most prevalent. In both cases, complete automation of test case design is difficult. Automation of black-box test is only meaningfully possible if a formal specification exists, and, due to the limits of symbolic execution, tools supporting white-box tests are limited to program code instrumentation and coverage measurement. Evolutionary testing is a promising approach for automating structure-oriented test case design completely. In many experiments, high coverage degrees were reached using evolutionary testing. In this paper we shall investigate the suitability of structure-based complexity measures to assess whether or not evolutionary testing is appropriate for the structure-oriented test of given test objects.

1 Introduction

A large number of today's products are based on the deployment of embedded systems. Examples can be found in nearly all industrial areas, such as in aerospace technology, railway and motor vehicle technology, process and automation technology, communication technology, process and power engineering, as well as in defense electronics. Nearly 90 % of all electronic components produced today are used in embedded systems.

In software development for embedded systems, analytical quality assurance techniques must also be employed intensively besides constructive methods for the specification, design and implementation of the systems. In practice, the most important analytical quality assurance measure is dynamic testing. Testing is the only procedure which allows for the examination of dynamical system behavior in the real application environment. Test case design is the most important test activity, since the type, scope, and quality of the test are determined by selecting feasible test cases. Evolutionary testing is a promising approach for the automation of structure oriented test case design [1, 2, 3, 4, 5]. As evolutionary tests are based on the use of heuristic search methods, it is difficult to assess whether the use of evolutionary tests for a concrete test object is promising or not – that is, whether or not the evolutionary test

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1350–1362, 2004.

is able to determine test cases which achieve a high code coverage. We term a test object's suitability for evolutionary testing its *evolutionary testability.*

In this paper we investigate the suitability of structure-based software measures for predicting the evolutionary testability of a test object. If it is possible to establish a relationship between software measures and evolutionary testability, then the evolutionary test could be tailored to the needs of the individual test objects in order to achieve the highest degree of efficiency possible.

This paper is arranged as follows: In the second section, the basics for the automation of test case design for structure-oriented test procedures with evolutionary tests are provided. In the third section, the software measures investigated are briefly presented. Section 4 provides an overview of the results. The causes of the results and the weaknesses of the analyzed software measures are examined in more detail. Section 5 provides an evaluation of these results. The paper closes with a short summary in the sixth section.

2 Test Case Design

In the context of test case design, those test cases are defined with which the testing of the system is to be executed. Existing test case design methods can essentially be differentiated into black-box tests and white-box tests. In the case of black-box tests, test cases are determined from the specification of the program under test, whereas, in the case of white-box tests, they are derived from the internal structure. In both cases, complete automation of the test case design is difficult. Automation of the black-box test is only meaningfully possible if a formal specification exists, and, due to the limits of symbolic execution, tools supporting structure-oriented tests are limited to program code instrumentation and coverage measurement. The aim of applying evolutionary testing to structure-oriented test case design is the generation of a quantity of test data, leading to the best possible coverage of the structural test criterion under consideration. In the case of this test case generation method, it is only possible to determine the input values. Corresponding expected values must be defined by the tester him or herself.

Evolutionary testing is characterized by the use of meta-heuristic search techniques for test case generation. The test aim considered is transformed into an optimization problem. The input domain of the test object forms the search space in which one searches for test data that fulfil the respective test aim. Due to the non-linearity of software (if-statements, loops etc.) the conversion of test problems into optimization tasks mostly results in complex, discontinuous, and non-linear search spaces. In our work, evolutionary algorithms are used to generate test data because their robustness and suitability for the solution of different test tasks has already been proven in previous work, e.g. [1, 3, 5, 6, 7].

2.1 Application of Evolutionary Testing to White-Box Testing

In order to automate test case design for white-box testing with the aid of evolutionary algorithms, the test is divided into test aims. Each test aim represents a program structure that requires execution to achieve full coverage, e.g. for simple condition testing each program condition represents two test aims: evaluating the condition as True and as False. The fitness function is minimized during optimization. If an individual obtains a fitness value of 0, a test datum is found which fulfils all branching conditions in the way required to reach the current test aim. The evolutionary test proceeds to the next test aim. For a more detailed description of the fitness functions refer to [5]. Further definitions of fitness functions for evolutionary structure tests are contained in [1, 3].

3 Structure-Based Software Measures

In the case of the structure-oriented tests considered here, a program has a high level of *evolutionary testability* if and only if two requirements are met during the evolutionary test for the chosen test criterion. In the first instance, a high level of coverage must be achieved and in the second, a low number of test data necessary. These are exactly the requirements in which we are interested for the possible application of evolutionary tests. If it were possible to reliably predict the evolutionary testability of a program or that of its individual test aims, one would be able to decide, before the test, which test termination criteria should be chosen for the individual test aims and whether or not an evolutionary structure test would be of any use for the program. Furthermore, the scope of the search and the selection of the evolutionary operators used in dependence on the qualities of the test object could be changed. Therefore, we shall look at the complexity measures *Number of Test Aims, Executable Lines of Code, Halstead's Vocabulary, Halstead's Length* [8], *Cyclomatic Complexity* [9], *Myers Interval* [10], and *Nesting Level Complexity* [11], all of which allow statements as to the structural properties of programs to be made.

Executable Lines of Code. When determining the Executable Lines of Code (ELOC) the data-flow and control-flow properties of the software examined are not taken into consideration. It only contains lines that contain executable statements.

Halstead's Length and Vocabulary. The software measure Halstead's Length (HALL) is based on counting the number of operands and operators as well as their number of uses in the examined software. When calculating Halstead's Vocabulary (HALV), in contrast, the number of different operators and operands is also taken into account.

Cyclomatic Complexity. Cyclomatic Complexity (CYC) is defined as the number of edges of the programs control-flow graph minus the number of its nodes plus two times the number of its linked components.

Myers Interval. Myers Interval (MI) is an extension of Cyclomatic Complexity, which takes the complexity of the branching conditions more accurate into account. The value of this metric is the sum of the number of logical operators AND and OR in the conditional expressions of the program investigated.

Nesting Level Complexity. Nesting Level Complexity (NLC) assesses the complexity of a software with regard to the nesting level of its conditions and statements.

Number of Test Aims. The aim of the evolutionary structure test is to find a set of test data with which every test aim of the test object is reached at least once. The software measure Number of Test Aims (NTA) can thus be used as a means of estimating the test effort, which would be expected to increase with the number of test aims.

4 Experiments

Source text and structure-based software measures do not seem to be able to sufficiently express the evolutionary testability of a test object. In [12] Buhr showed, using 40 test objects, that the seven above-mentioned software measures were not adequately suited to this purpose. Buhr was unable to establish a sufficient connection between these software measures and the level of coverage achieved, to justify the use of software measures to reach conclusions regarding evolutionary test behavior. The cause is stated as being the insignificant connection between the structural properties evaluated by the software measures and the properties which it is necessary to describe in order to judge evolutionary testability. To investigate the difficulties which conventional software measures have in evaluating evolutionary testability we would like to take a closer look at the experiments.

4.1 Test Preparation

A large number of different test objects was chosen, which spanned a broad spectrum of different program complexities and originated from different application areas: mathematical calculations (both control-related tasks and the execution of string and character operations) and components from automotive and motor electronics. They possessed an appropriate value spectrum for each software measure selected: ELOC varied from 7 to 320, HALV reached values between 25 and 949, HALL between 93 and 9138. CYC in one area differed from between 3 to 50, MI reached 0 to 47, NLC varied from 1 to 17 and NTA from 4 to 132.

The minimal multiple-condition coverage test was chosen as a structure test criterion in the experiments because in this test the coverage level reflects the percentage of non-accomplishable test cases exactly. The branch coverage test can lead to the problem that the number of test goals not achieved does not comply with the number of non-executable commands and thus some conditions may be optimized several times. As the parameter settings of the evolutionary algorithms influence the coverage

level achieved to a large extent, the evolutionary parameters were kept constant during all the experiments. This ensures that the results can be compared. For all the experiments, the population size was set at 300 individuals: 6 subpopulations, each made up of 50 individuals. Each subpopulation deployed different evolutionary algorithms, which, in turn, pursued different search strategies and competed with each other. Fitness assignment took place proportionally, selection was carried out by means of Stochastic Universal Sampling [13] and the type of recombination used was Mühlenbein und Schlierkamp-Voosen's discrete recombination [14]. On average, for each individual, a variable was mutated and executed with the aid of the mutation of real variables. During reinsertion the generation gap was 90%, i.e. the next generation consisted of 10% parent individuals and 90% offspring. Five test runs per test object were carried out for a total of 13 selected test objects using these settings.

When choosing coverage level and the generations, only those test goals from the control flow graph were taken into account which were executable and which the evolutionary structure test, due to the definition of its fitness function, could reach with a sufficiently long execution time.

Table 1. Minimal multiple-condition coverage test of the 13 test objects investigated. The number of evolutionarily possible test goals, the average coverage level reached during this process and the average number of generations required are provided as results

Test object No.	Test object name	evol. possible test goals	Coverage reached in %	No. of average generations
1	blockerkennung	27	85,9	533
2	bnldev	24	98,3	248
3	einklemmschutz	10	90,0	308
4	firstJan	4	100,0	28
5	function_hhs	56	100,0	322
6	gcd1	5	100,0	79
7	gcd2	12	100,0	18
8	hail	16	100,0	70
9	kindersicherung	52	88,9	3.175
10	leap	4	100,0	47
11	mzuef	126	100,0	23
12	powi	22	100,0	35
13	tuermodul	97	80,0	8.612

In order to be able to evaluate the quality of software measures with regard to their ability to judge evolutionary testability, one can compare them with real measurements of evolutionary testability (table 1). It is possible to do this by sorting test objects according to their evolutionary testability measured. The further in front a test object is in the sorting, the higher its evolutionary testability will be. For this reason, weighting is carried out firstly according to the coverage reached and then according to the number of generations required (see sect. 3). This kind of sorting will be called *evolutionary sorting* in the following. Thus it becomes possible to evaluate various other measures with regard to their suitability to measuring evolutionary testability, depending on how far their sorting differs from the evolutionary sorting.

4.2 Average Divergence in the Case of Sortings

So as to be able to assess the prediction quality of a software measure regarding evolutionary testability we need to have a look at the divergence of the test objects sorted according to the measure from test objects sorted in an evolutionary way. The divergence describes the sum of distances of all positions of the same test objects in two different sortings. The less the test objects which are sorted on the basis of a certain software measure diverge from the evolutionary sorting method, the better the influence on the quality of the predicted evolutionary testability will be.

The *average divergence* of a random sorting from an evolutionary sorting or any other sorting could help us in evaluating the quality of the software measures. If a sorting of a specific software measure diverges from the evolutionary sorting in such a way that it is close to the average divergence to be expected, this means that it does not contain any suitable prediction quality. In order to determine the average divergence $\mathrm{div}_{\varnothing}(n)$ we will now have a closer look at two sortings of n elements:

Let $S_n = \{x_1, x_2, \ldots, x_n\}$ and $S_n' = \{x_1', x_2', \ldots, x_n'\}$ be any two sortings of an ordered set A, whose n elements are clearly distinguishable with regard to one variable relation. Now, the average overall divergence of S_n from S_n' is calculated from the sum of the average divergences of all the elements.

$$\mathrm{div}_{\varnothing}(x_1) = \frac{\sum_{i=0}^{n-1} i}{n} \tag{1}$$

applies to the first element x_1 of the sorting S_n since each of the positions receives the same probability $\frac{1}{n}$. The same goes for the last element x_n, i.e. $\mathrm{div}_{\varnothing}(x_1) = \mathrm{div}_{\varnothing}(x_n)$. To the second and the next to last element of the sorting S_n the following line applies:

$$\mathrm{div}_{\varnothing}(x_2) = \mathrm{div}_{\varnothing}(x_{n-1}) = \frac{1+\sum_{i=0}^{n-2} i}{n} = \frac{\sum_{j=0}^{1} j+\sum_{i=0}^{n-2} i}{n}, \tag{2}$$

and for the third and third to last element:

$$\mathrm{div}_{\varnothing}(x_3) = \mathrm{div}_{\varnothing}(x_{n-2}) = \frac{1+2+\sum_{i=0}^{n-3} i}{n} = \frac{\sum_{j=0}^{2} j+\sum_{i=0}^{n-3} i}{n}. \tag{3}$$

This continues up to the middle elements, to which the following lines apply:

$$\mathrm{div}_{\varnothing}(x_{\frac{n}{2}}) = \mathrm{div}_{\varnothing}(x_{\frac{n}{2}+1}) = \frac{\sum_{j=0}^{\frac{n}{2}-1} j+\sum_{i=0}^{\frac{n}{2}} i}{n} \quad \text{where n is even.} \tag{4}$$

$$\mathrm{div}_{\varnothing}(x_{\lceil \frac{n}{2} \rceil}) = \frac{\sum_{j=0}^{\lfloor \frac{n}{2} \rfloor} j+\sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} i}{n} \quad \text{where n is odd.} \tag{5}$$

After adding up all the average divergences of the individual elements $x_1, x_2, \ldots, x_n$, we get for n even the following overall divergence

$$\mathrm{div}_{\varnothing}(n) = \sum_{i=1}^{n} \mathrm{dev}_{\varnothing}(x_i) = 2 \cdot \left(\frac{\sum_{i=0}^{n-1} i}{n} + \frac{\sum_{j=0}^{1} j + \sum_{i=0}^{n-2} i}{n} + \frac{\sum_{j=0}^{2} j + \sum_{i=0}^{n-3} i}{n} + \cdots + \frac{\sum_{j=0}^{\frac{n}{2}-1} j + \sum_{i=0}^{\frac{n}{2}} i}{n} \right) =$$

$$2 \cdot \frac{\sum_{j=0}^{\frac{n}{2}-1} \sum_{i=0}^{j} i + \sum_{j=\frac{n}{2}}^{n-1} \sum_{i=0}^{j} i}{n} = \frac{2 \cdot \sum_{j=0}^{n-1} \sum_{i=0}^{j} i}{n} \tag{6}$$

and for n odd

$$\mathrm{div}_{\varnothing}(n) = \sum_{i=1}^{n} \mathrm{div}_{\varnothing}(x_i) = 2 \cdot \left(\frac{\sum_{i=0}^{n-1} i}{n} + \frac{\sum_{j=0}^{1} j + \sum_{i=0}^{n-2} i}{n} + \frac{\sum_{j=0}^{2} j + \sum_{i=0}^{n-3} i}{n} + \cdots + \frac{\sum_{j=0}^{\lfloor \frac{n}{2} \rfloor - 1} j + \sum_{i=0}^{\lceil \frac{n}{2} \rceil} i}{n} \right) +$$

$$\frac{\sum_{j=0}^{\lfloor \frac{n}{2} \rfloor} j + \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} i}{n} = 2 \cdot \left(\frac{\sum_{j=0}^{\lfloor \frac{n}{2} \rfloor - 1} \sum_{i=0}^{j} i + \sum_{j=\lceil \frac{n}{2} \rceil}^{n-1} \sum_{i=0}^{j} i}{n} \right) + \frac{\sum_{j=0}^{\lfloor \frac{n}{2} \rfloor} j + \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} i}{n} = \frac{2 \cdot \sum_{j=0}^{n-1} \sum_{i=0}^{j} i}{n} . \tag{7}$$

Consequently, the following applies in general to the average divergence of an n-element sorting:

$$\mathrm{div}_{\varnothing}(n) = \frac{2 \cdot \sum_{j=0}^{n-1} \sum_{i=0}^{j} i}{n} . \tag{8}$$

From this, it follows that for the 13 remaining test objects $\mathrm{div}_{\varnothing}(13) = 56$. This means that 56 is the value which we get as an average for the divergence from the evolutionary sort sequence, if we arbitrarily sort the test objects without any particular order.

4.3 First Results

If we apply the seven software measures mentioned above to the 13 test objects they are not able to provide us with reliable predictions concerning evolutionary testability – as already noticed in [12]. In table 2 and figure 1 the evolutionary sorting is depicted, while in table 3 and figure 2 the most successful sorting according to NLC is shown. Also, for the evaluation of evolutionary testability the average degree of coverage of the minimal multiple condition coverage and the number of generations needed on average to achieve the evolutionary possible test aims was stated for every test object. The best possible sorting on the basis of the software measure NLC possesses a test object (no. 11), whose position within a sorting of only 13 test objects nonetheless diverges from the evolutionary sorting by six positions. Furthermore, there are two test objects (no. 5 and no. 6), which each diverge by five positions.

Table 2. Evolutionary sorting of the test objects

Test object no.	Test object name	Coverage in %	Number of generations
7	gcd2	100,0	18
11	mzuef	100,0	23
4	firstJan	100,0	28
12	powi	100,0	35
10	leap	100,0	47
8	hail	100,0	70
6	gcd1	100,0	79
5	function_hhs	100,0	322
2	bnldev	98,3	248
3	einklemmschutz	90,0	308
9	kindersicherung	88,9	3.175
1	blockerkennung	85,9	533
13	tuermodul	80,0	8.612

Table 3. Sorting of test objects on the basis of NLC

Test object no.	Test object name	Coverage in %	Number of generations	NLC
7	gcd2	100,0	18	23
6	gcd1	100,0	79	23
4	firstJan	100,0	28	46
10	leap	100,0	47	46
12	powi	100,0	35	70
8	hail	100,0	70	70
3	einklemmschutz	90,0	308	70
11	mzuef	100,0	23	93
2	bnldev	98,3	248	93
9	kindersicherung	88,9	3.175	93
1	blockerkennung	85,9	533	139
13	tuermodul	80,0	8.612	139
5	function_hhs	100,0	322	395

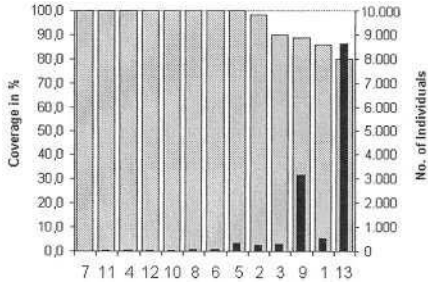


Fig. 1. Evolutionary sorting of the test objects

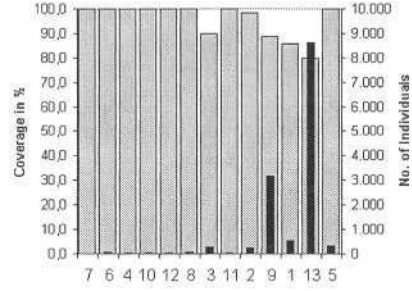


Fig. 2. Sorting of test objects on the basis of NLC

Table 4 shows us a closer quantification of the individual sortings sorted on the basis of software measures. Here we can find the divergences of the test objects which were sorted according to the seven chosen software measures from the evolutionary sorting sequence. It is clear that the best possible sorting of the test objects, based on the complexity measure NLC and provides us with a value of 24, is not very successful when compared with an average divergence value of 56. Additionally, the worst possible sortings which were sorted according to the software measures HALF and HALL show a value of 50 which is almost the result of random sorting.

Table 4. Divergences of software measures compared to evolutionary sorting

Software measure	ELOC	HALF	HALL	CYC	MI	NLC	NTA
Divergence	46	50	50	44	32	24	48

4.4 Software Measure Combinations

A further approach consists of combining different software measures in order to obtain reliable statements as to their evolutionary testability. It is possible that the weaknesses of individual software measures may be compensated for by the combination of several software measures.

The software measures are normalized, thus allowing us to compare their different values. It then has to be investigated which combination and weighting of the individual software measures represents the best possible measure for describing evolutionary testability. During this investigation the seven software measures together with the divergence resulting from the evolutionary sorting span an eight-dimensional search space, whose minimum has to be optimized.

The evolutionary algorithm lends itself to this because it is particularly well suited to optimizing a multi-dimensional search space using a corresponding fitness function [15]. Fitness assignment occurs in proportion to the divergence of the sorting sequence, resulting from the weightings of the software measure combinations, to the evolutionary sorting. In this way, the evolutionary algorithm will favor those software measure combinations which best describe evolutionary testability.

Table 5. Best divergences and weightings of the software measure combinations

Test no.	Divergence	ELOC	HALV	HALL	CYC	MI	NLC	NTA
1	30	0.0010	0.0033	0.0480	0.0011	0.9944	0.4167	0.0730
2	30	0.0505	0.0001	0.0123	0.0000	0.9092	0.3624	0.0295
3	30	0.0017	0.0768	0.0036	0.0075	0.8927	0.4312	0.0094
4	30	0.0000	0.0371	0.0000	0.0305	0.9372	0.4646	0.0350
5	30	0.0702	0.0121	0.0206	0.0012	0.9881	0.4167	0.0156
6	30	0.0449	0.0004	0.0274	0.0000	0.9092	0.3624	0.0295
7	30	0.0020	0.0507	0.0036	0.0108	0.9690	0.4164	0.0453
8	30	0.0000	0.0191	0.0082	0.0000	0.9632	0.4646	0.0350
9	30	0.0064	0.0120	0.0416	0.0011	0.9872	0.5204	0.0307
10	30	0.0016	0.0000	0.0036	0.0075	0.8873	0.4157	0.0355

During the optimization, values in the interval [0, 1] were used for the weighting of the individual software measures. The sum of the respective software measure weightings produced a new measure, from which a new sorting sequence also resulted. It turned out that, with the exception of MI and NLC all the values of the software measures always lay between 0 and 0.08 (table 5). In contrast, the values for MI ranged between 0.88 and 1 and those for NLC between 0.36 and 0.53. This leads us to surmise that NLC and MI primarily contribute to optimal sorting. However, if one considered these two software measures exclusively for an optimization run, it would not be possible to attain below 30 unless the proportion of MI became vanishingly low in comparison to NLC (< approx. 0,2%), which alone managed to achieve a divergence of 24. In no run was it possible to achieve a lower value than this.

If we compare table 4 and table 5, the result is astonishing: The best value of the software measure combinations obtains a divergence of 30 and is presented in table 5 with the respective weightings of the software measures. In not one of the ten test

runs was any of the combinations able to achieve a divergence as low as the best software measure NLC alone, which obtained a divergence of 24. Combinations of software measures obviously do not lead to improvements compared to individual software measures. It seems that combinations, however they are weighted, weaken the predictions of individual software measures with regard to evolutionary testability.

4.5 Eliminating Disturbing Influences

Other influences resulting from the test objects investigated also play a decisive role in determining evolutionary testability. Buhr [12] believes that it is not the structure but rather the data flow properties of a test object which primarily influence its evolutionary testability. This assumption is based on the localization of different disturbing influences – disturbing influences which could occur during the execution of the evolutionary structure test. Examples are incrementors, i.e. local static variables which act as timers and only make it possible to achieve a condition by repeating program calls very often; floating point optimizations whose randomly created optimization steps might possibly be too large to reach a certain constant value correctly; flags [16] or side-effects [17]. These so strongly disturb the evolutionary testability that characteristics described by source code or structure-based software measures no longer have any sufficient effect.

If we eliminate those test objects which have disturbing influences on the evolutionary structure test concerning test goal reachability, this would leave us with only seven of the 13 test objects described. With these seven test objects some of the software measures analyzed provide quite reliable predictions about evolutionary testability. As a measure for the quality of a reliable prediction the divergence from the evolutionary sorted test objects was again chosen. In table 6 the divergences of all software measures for this test series are depicted, in table 7 and figure 3 the evolutionary sorting is shown, and in table 8 and figure 4 one can find the most successful sorting according to CYC.

With seven test objects the average divergence $\mathbf{div}_{\varnothing}$ achieves a value of 16. Nevertheless, the sorting based on CYC surpasses this by a factor of 4. Even if we combine the software measures with the help of the evolutionary algorithm, the result of 6 reached by test objects without disturbing influences on the evolutionary structure test, is not better than the most successful sorting of individual software measures.

Table 6. Divergences of software measures compared to evolutionary sorting of test objects without problem cases

Software measure	ELOC	HALF	HALL	CYC	MI	NLC	NTA
Divergence	6	10	10	4	6	6	6

Table 7. Evolutionary sorting of test objects without problem cases

Test object no.	Test object name	Coverage in %	Number of generations
4	firstJan	100,0	28
10	leap	100,0	47
8	hail	100,0	70
6	gcd1	100,0	79
3	einklemmschutz	90,0	308
9	kindersicherung	88,9	3.175

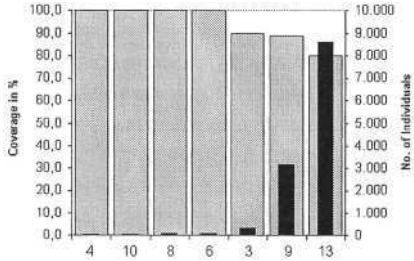


Fig. 3. Evolutionary sorting of test objects without problem cases

Table 8. Sorting of test objects without problem cases based on CYC

Test object no.	Test object name	Coverage in %	Number of generations	CYC
4	firstJan	100,0	28	22
10	leap	100,0	47	22
6	gcd1	100,0	79	22
3	einklemmschutz	90,0	308	44
8	hail	100,0	70	59
9	kindersicherung	88,9	3.175	169

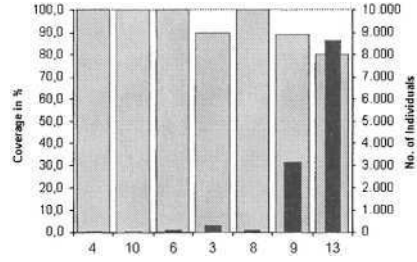


Fig. 4. Sorting of test objects without problem cases based on CYC

5 Evaluation

On the basis of the experiments it could be shown that by using source code or structure-based software measures it was only possible to make mediocre predictions as to evolutionary testability, which complies with Buhr's findings. Characteristics of the test objects exist, which are not sufficiently depicted by these software measures, even if the test objects are free of disturbing influences regarding the reachability of the test aim. It is questionable how strongly the differing weightings of the test objects within the sortings would affect the results. To make better statements, extensive test series which use an application area which is as broad as possible need to be executed.

Moreover, slight source code modifications such as, for example, greater limitations in the test objects' conditions hardly influence the software measures or do not do so at all. They can, however, have a powerful influence on evolutionary testability. This is illustrated by the following example:

```
long w, x, y, z;
if (w == 0 && x == 0 && y == 0 && z == 0) { // testaim
```

In the experiment, this artificially created test object obtained an average coverage of 40% for five test runs. This low coverage is a consequence of the subconditions' substantial limitations. The evolutionary structure test is already overstrained starting from two subconditions to be fulfilled together, producing 200 test data generations. The ratio of positive to negative test data is 1 to $1.84 * 10^{19}$ for two subconditions and around 1 to $3.40 * 10^{38}$ for all four conditions. Only for one or no subconditions to be fulfilled, could a successful test datum be found. The test object thus possesses low evolutionary testability, whereas the low software measures would lead one to expect high evolutionary testability (except for MI and NTA the evaluation of the software measures is always the lowest in this example). In the case of this test object, the source code and structure-based software measures would fail to evaluate evolutionary testability.

6 Conclusion

In order to increase the quality of tests and to reduce the development costs for software-based systems, test methods are called for which support a complete test and which are, to a large extent, automatable. The evolutionary test lends itself to this, because it supports fully automated test case generation during structure tests.

So as to estimate the evolutionary testability of test objects we checked for a possible connection between different software measures and the execution of the evolutionary test. We looked at the following source code and structure-based software measures: *Number of Test Aims, Executable Lines of Code, Halstead's Vocabulary, Halstead's Length, Cyclomatic Complexity, Myers Interval, Nesting Level Complexity,* and *Number of Test Goals.* The values measured were compared to the evolutionary testability of test objects of varying complexity. After the elimination of disturbing influences within test objects which would hinder the execution of the evolutionary test, the structure-based software measure *Cyclomatic Complexity* was able to generate the best forecasts for the evolutionary testability to be expected.. *The sorting of the test objects based on this software measure came quite close to the sorting based on evolutionary testability and surpassed an average sorting by a factor of 4.*

It was possible to identify cases in which the software measure *Cyclomatic Complexity would fail, i.e. there are test objects whose evolutionary testability is rated incorrectly by the Cyclomatic Complexity measure (or any other of the source code or structure-based software measures analyzed). The reason for this can be found in the dependency of evolutionary testability on certain structure characteristics of the software to be tested, which none of the software measures analyzed (nor any other known software measure) is capable of expressing.* Thus, future work should concentrate on the development of a software measure which is able to mirror evolutionary testability as exactly as possible.

Acknowledgements. The work described has been performed within the SysTest project. The SysTest project is funded by the European Community under the 5th Framework Programme (GROWTH), project reference G1RD-CT-2002-00683.

References

1. Sthamer, H.: The Automatic Generation of Software Test Data Using Genetic Algorithms, PhD Thesis, University of Glamorgan, Pontyprid, Wales, Great Britain (1996)
2. Tracey, N., Clark, J., Mander, K., McDermid, J.: An Automated Framework for Structural Test-Data Generation, Proceedings of the 13th IEEE Conference on Automated Software Engineering, Hawaii, USA (1998)
3. Pargas, R., Harrold, M., Peck, R.: Test-Data Generation Using Genetic Algorithms, Software Testing, Verification & Reliability, vol. 9, no. 4 (1999) 263–282
4. Michael, C., McGraw, G., Schatz, M.: Generating Software Test Data by Evolution, IEEE Transactions on Software Engineering, vol. 27, no. 12 (2001) 1085–1110
5. Wegener, J., Baresel, A., Sthamer, H.: Evolutionary Test Environment for Automatic Structural Testing, Information and Software Technology, vol. 43 (2001) 841–854
6. Wegener, J., Grochtmann M.: Verifying Timing Constraints of Real-Time Systems by means of Evolutionary Testing, Real-Time Systems, vol. 15, no. 3 (1998) 275–298
7. Jones, B., Eyres, D., Sthamer, H.: A Strategy for using Genetic Algorithms to Automate Branch and Fault-based Testing, The Computer Journal, vol. 41, no. 2 (1998) 98–107
8. Halstead, M.: Elements of Software Science, Prentice Hall, New York, USA (1977)
9. McCabe, T.: A Complexity Measure, IEEE Transactions on Software Engineering, vol. 2, no. 12 (1976) 208–220
10. Myers, G.: An Extension to the Cyclomatic Measure, ACM SIGPLAN Notices, vol. 12, no. 10, 61–64
11. QA C Source Code Analyser – Command Line, Programming Research Ltd. (1974)
12. Buhr, K.: Einsatz von Komplexitätsmaßen zur Beurteilung Evolutionärer Testbarkeit (Complexity Measures for the Assessment of Evolutionary Testability). Diploma Thesis, Technical University Clausthal (2001)
13. Baker, J.: Reducing Bias and Inefficiency in the Selection Algorithm. Proceedings of the 2nd International Conference on Genetic Algorithms (ICGA '87), Cambridge, Massachusetts, USA (1987) 14–21
14. Mühlenbein, H., Schlierkamp-Voosen, D.: Predictive Models for the Breeder Genetic Algorithm: I. Continuous Parameter Optimization, Evolutionary Computation, vol. 1, no. 1 (1993) 25–49
15. Pohlheim, H.: Evolutionäre Algorithmen. Verfahren, Operatoren und Hinweise für die Praxis, Springer-Verlag, Berlin (2000)
16. Harman, M., Hu, L., Hierons, R., Baresel, A., Sthamer, H.: Improving Evolutionary Testing by Flag Removal, Proceedings of the Genetic and Evolutionary Computation Conference, New York, USA (2002)
17. Harman, M., Hu, L., Hierons, R., Munro, M., Zhang, X., Dolado, J., Otero, M., Wegener, J.: A Post-Placement Side-Effect Removal Algorithm, Proceedings of the IEEE International Conference on Software Maintenance, Montreal, Canada (2002)

Hybridizing Evolutionary Testing with the Chaining Approach

Phil McMinn and Mike Holcombe

Department of Computer Science, The University of Sheffield,
Regent Court, 211 Portobello Street,
Sheffield, S1 4DP, UK
{p.mcminn, m.holcombe}@dcs.shef.ac.uk

Abstract. Fitness functions derived for certain white-box test goals can cause problems for Evolutionary Testing (ET), due to a lack of sufficient guidance to the required test data. Often this is because the search does not take into account data dependencies within the program, and the fact that some special intermediate statement (or statements) needs to have been executed in order for the target structure to be feasible. This paper proposes a solution which combines ET with the Chaining Approach. The Chaining Approach is a simple method which probes the data dependencies inherent to the test goal. By incorporating this facility into ET, the search can be directed into potentially promising, unexplored areas of the test object's input domain. Encouraging results were obtained with the hybrid approach for seven programs known to originally cause problems for ET.

1 Introduction

Evolutionary Testing (ET) (see Ref. [1]) uses evolutionary algorithms to search for software test data. For white-box testing coverage criteria, the execution of each uncovered structure - for example a program statement or branch - is taken individually as the goal of the search. However, the coverage of certain structures in certain types of programs can cause problems for ET, due to a lack of sufficient guidance to the required test data. Often this is because the technique does not take into account data dependencies between statements in the program. In order for the target structure to be feasible, special intermediate statements may need to have been executed. For instance, the outcome of a target branching condition may be dependent on a variable having a special value, which is only set in a special circumstance - for example a special flag or enumeration value denoting an unusual condition; a unique return value from a function call indicating an error has occurred; or a counter variable only incremented under certain situations. If the intermediate assignment is particularly unusual, it will not be executed by chance. Without specific knowledge of such dependencies, the evolutionary search tends to stagnate and fail. The problem of flag variables in particular has received much interest from researchers [2,3,4]

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1363–1374, 2004.

- however there has been little attention with regaxds to the broader problem as described.

This paper proposes a solution which combines ET with the Chaining Approach. The Chaining Approach [5,6] is a structural test data generation technique based on local search. If the local search fails to find test data which directly executes the target, data flow analysis is used to identify intermediate statements, or *events,* which can decide whether the target will be reached or not. By instead focusing on test data that executes a *chain* of events, the search can be directed into potentially unexplored, yet promising areas of the test object's input domain. In the hybrid approach, evolutionary algorithms are applied for the test data search. However, a chaining mechanism is employed whenever a "problematic" test goal is encountered. Encouraging results were obtained with initial experiments carried out on seven test objects known to originally cause problems for ET.

2 Evolutionary Testing (ET)

ET (see Ref. [1]) uses evolutionary algorithms to search the input domain of a test object for desired test data. Individuals of the search are simply input vectors to the test object. The fitness function depends on the type of test data required.

White-box testing coverage criteria demand that all program structures of a certain type - for example statements or branches - are exercised. ET can automate the derivation of test data for this purpose by searching for input vectors which will execute each specific structure. In more developed approaches [7,8], the fitness function is made up of two components. The first component is referred to as the *approximation level,* or, (perhaps more appropriately) the *approach level.* This metric assesses how close an individual was to reaching the target on the basis of its execution path through the program's control structure. Central to this is the notion of a *critical branch* (also referred to in the literature as a *decisive branch*). A critical branch is simply a branch which leads to the target being missed. Once such a branch has been taken, failure to reach the target has essentially been "decided". For improved handling of structures nested within loops [8], a branch that misses the target within a loop iteration is also counted as critical. For the example of Figure 1, where the goal of the search is the execution of node 6, the set of critical branches includes the true branch from node 1 and the false branch from node 4. The false branch from node 5 is also treated as critical, since it misses node 6 within a loop body. The approach level for an individual is calculated by subtracting one from the number of critical branches lying between the node from which the individual diverged away from the target, and the target itself. For the execution of node 6, individuals taking the false branch at node 5 receive an approach level of zero, individuals diverging away down the true branch at node 4 receive an approach level of one, and so on.

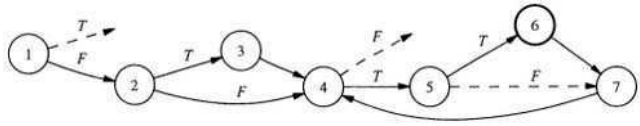


Fig. 1. An example control flow graph for calculating approach levels, with respect to the target - node 6. Critical branches are indicated with dashed arrows

At the point at which control flow diverged away from the target down a critical branch, a *branch distance calculation* is computed. This is the second component of the fitness function. This value indicates how close the alternative branch was to being taken. For example if the false branch is taken from node 5, and the branching condition at this node is `(x == y)`, the distance for the true branch is calculated using the formula `abs(x - y)` [9]. The overall (minimizing) fitness function is zero if the target structure is exercised, otherwise, a fitness value is computed for the individual on the basis of the approach level and the branch distance calculation d mapped into the range $0 \leq d < 1$:

$$approach\_level + d \tag{1}$$

3 The Problem

ET has been shown to be an effective method for structural test data generation [7]. However, the approach performs poorly for programs with certain characteristics. One problem can arise as a result of data dependencies within the program, where the target structure requires the prior execution of certain intermediate statements in order for it to be feasible. Since such statements do not affect whether the target structure will be reached in terms of control flow through the program, they are effectively ignored for the purposes of computing fitness information. If such statements are only executed under "special circumstances rather than by chance, the target structure might not be covered.

Take the example of Figure 2. The `inverse` function finds the multiplicative inverse of an input `x`. To avoid a division by zero error, zero is simply returned when the divisor is zero. If the goal of the search is to execute node b, it is necessary to execute the true branch from node a - requiring a zero return value from the `inverse` function. This is in turn passed the input value `x` as an argument. However, the zero input value required to execute the target is unlikely to be found by chance, simply because it represents a very small portion of the overall input domain. Furthermore, the fitness function actually leads the search away from this value for all other input values - since as the value of `x` increases, the result of the `inverse` function decreases. This deception can be seen in a plot of the fitness function landscape (Figure 4a).

```
    double function_under_test(double x)        double inverse(double d)
    {                                           {
(a)     if (inverse(x) == 0)                (c)     if (d == 0)
(b)         // target                       (d)         return 0;
    }                                               else
                                            (e)         return 1 / d;
                                                }
```

Fig. 2. C example resulting in a deceptive fitness function into order to execute node e (control flow graph node identifiers appear in brackets to the left of program statements)

The fitness landscape can also be flat for large areas of the input domain, giving the search little guidance at all. It is well known that this problem can be caused by the existence of flag and enumeration variables in branch predicates [2,3,4]. A flag is simply a variable that is true or false. When such a variable is used in a branch predicate, plateaux form in the fitness landscape corresponding to the true and false values of the flag. This can be seen with the example of Figure 3. The target of the search is to execute the statement corresponding to node f, requiring the true branch from node e to be taken. The branching condition at node e involves the variable `flag`, which is only true when all the variables of an inputted array are zero. However no guidance is provided to this input vector, since for all other vectors the branch distance calculation result, using the false flag value, is the same. The resulting plateau can be seen in a plot of the fitness landscape (Figure 5a). It can easily be seen from the program that the search simply needs to execute the path up to e where the assignment at node d is avoided on each iteration of the loop. However, ET is not aware of this assignment, and consequently does not recognize what needs to be done to prevent it - that is the optimization of all array values to zero.

A similar problem can occur with enumeration variables. Again, two plateaux form in the fitness landscape when such variables are involved in branch predicates - one for the "correct" value, and one for all other values. Since enumerations are not ordinal types, no value from the enumeration can be deemed to be "closer" to the desired value.

A number of solutions have been proposed for the particular problem of flags. Harman et al. [3] present an approach to the problem of flags which transforms the flag out of the program. A different type of transformation procedure needs to be applied depending on the characteristics of the test object. No work has yet been published applying these transformations to programs with loops. Bottaci [2] deals with a special case. Baresel et al. [4] present an alternative approach that analyses the program, and selects a suitable fitness function from a library depending on features found within the program (e.g. a flag involved in a loop). It is stated that the method struggles to avoid undesirable assignments to flags in loops. It is claimed the method extends to enumerations, but no experimental results are presented. Whilst these solutions apply different "rules" each time

```
    void flag_avoid_loop_assignment(int a[10])
    {
(a)     int flag = 1;
        int i;
(b)     for (i = 0; i < 10; i++)
        {
(c)         if (a[i] != 0)
(d)             flag = 0;
        }
(e)     if (flag)
(f)         // target
    }
```

Fig. 3. C example requiring the avoidance of a flag assignment contained in a loop in order to execute node f (control flow graph node identifiers appear in brackets to the left of program statements)

a different program characteristic is encountered, incorporation of the Chaining Approach represents a general and elegant solution to flags and the broader problem as described.

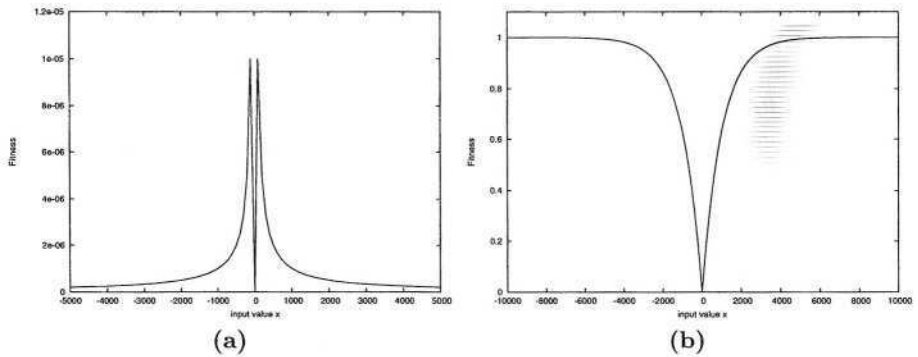
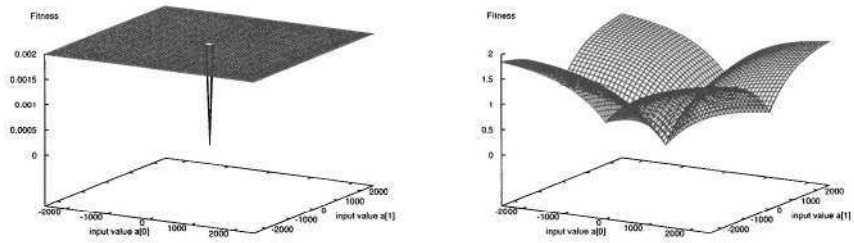


Fig. 4. Fitness landscapes for the deceptive example (Figure 2). (a) shows the landscape for the fitness function used by ET (equivalent to that of the initial event sequence with the hybrid approach). (b) shows the landscape of the fitness function for the successful event sequence with the hybrid approach

4 The Chaining Approach

The Chaining Approach [5,6] is an alternative test data generation technique, based on a local search method known as the "alternating variable method". If a critical branch is taken, the search method attempts to change the flow of



(a) (b)

Fig. 5. Fitness landscapes for the flag example, plotted for two array elements, assuming all other elements are zero, (a) shows the landscape for the fitness function used by ET (equivalent to that of the initial event sequence with the hybrid approach). (b) shows the landscape of the fitness function for the successful event sequence with the hybrid approach

control by searching for input values using the branch distance calculation of the alternative branch as a cost function. In practice this is performed by successively finding local minima for each input variable until the alternative branch is taken, or no further improvement can be made. At this point the search declares failure. Any move which results in an critical branch being taken earlier in the execution path is disallowed.

Being based on local search, the alternating variable method frequently encounters problematic test goals with non-trivial search landscapes. Typically a "problem" branching node is encountered, for which the search can not change control flow so that a critical branch is not taken. In an attempt to circumvent this problem, the Chaining Approach employs its backup plan - the construction of "event" chains - or *event sequences.* Event sequences force the consideration of certain statements leading up to the target structure, which may influence the outcome at the problem branching node. Such statements are identified by the use of data flow analysis. Using this mechanism, the search can be directed into potentially unexplored but promising areas of the input domain.

The crux of the chaining mechanism is best explained with an example. For the flag program of Figure 3 the execution of branching node e as true is problematic. Prior events are identified by finding the last definitions of variables used at the problem node. For node e, this is the variable `flag`, last defined at nodes a and d. Two event sequences are constructed, one demanding the execution of a before e, and the other requiring instead d before e. Node a of course, is the event that causes node e to be executed as true. However, this is only the case if the value of `flag` is preserved from this point up to node e - i.e. that node d is avoided on each iteration of the loop. Associated with each event is a *constraint set* of variables that should not be modified until the next event. The variable `flag` is inserted into this constraint set because its modification

destroys the effect of the assignment at node a. If node d is accidentally taken, the search uses the branch distance calculation of the false branch from this node in order to change the flow of control. In this way, the focus of the search changes from one which attempts to make `flag` true, into one which specifically searches for an array where each element is zero - which in turn results in `flag` being true.

Formally, an event sequence is described as a sequence of events $< e_1, e_2, \cdots e_k >$ where each event is a tuple $e_i = (n_i, C_i)$ where n_i is a program node and C_i is the set of variables referred to as the constraint set [5]. The initial event sequence for the flag example is simply $< (f, \phi) >$, with the event sequences constructed at the next "level" being denoted as follows:

$$\begin{array}{ll} (1) & < (a, \{flag\}), (e, \phi), (f, \phi) > \\ (2) & < (d, \{flag\}), (e, \phi), (f, \phi) > \end{array}$$

Of course, the addition of node d is unhelpful, and if the latter event sequence is tried, the search for test data will fail. Such an event sequence is described as *infeasible.*

The generated event sequences are organized in a tree. At the root of the tree is the initial event sequence. The first level contains the event sequences generated as a result of the first problem node. In more complicated examples, further problem branching nodes could be encountered on route to executing some new event inserted into the sequence. In such instances the Chaining Approach backtracks further, and looks for last definition statements for variables used at these new problem nodes. The additional event sequences are added to the tree, which is explored to some specified depth limit.

5 A Hybrid Approach

In the hybrid approach, evolutionary algorithms are used for the test data search, with a chaining mechanism employed for "problematic" test goals. The search for test data for the initial event sequence is equivalent to the search conducted by the original ET approach. However, if the search fails, the best overall individual is taken, and its problem branching nodes are identified. These are the branching nodes at which critical branches were taken. Using data flow information with regards to the first problem branching node, new event sequences are built, with separate evolutionary searches conducted in an attempt to try find test data for each new event sequence.

For an event sequence S of length l, the fitness for an input vector $\mathbf{x}$ is computed using the formula:

$$\sum_{i=1}^{l} fitness(e_i) \tag{2}$$

where e_i is the ith event in the event sequence, and $fitness(e)$ is calculated for an event $e = (n, C)$ as follows:

1. If control flow diverged down a critical branch with respect to n, add the result of Equation 1 for node n to the fitness
2. For each definition node executed for each variable $v \in C$ violating the definition clear path required until the next event in the sequence (if one exists); add d; the branch distance for the alternative branch at the last branching node, mapped into the range $0 \leq d < 1$

Figure 5 depicts the fitness landscape for the initial event sequence for the flag example (equivalent to the search conducted by the original ET approach), alongside that of the event sequence $< (a, \{flag\}), (e, \phi), (f, \phi) >$. As can be seen, the latter landscape offers more guidance to the search, and is more amenable to the discovery of the required test data. As an example, the fitness function is computed as follows. With values of the array being (1000,1000,...) (where all elements are zero unless stated), node e is executed twice, and the branch distance of the false branch from node c on the first two iterations of the loop are used in accumulating the fitness, according to rule 2. For array values of (500,1000,...), the situation is the same, but the branch distance of the false branch from node c in the first iteration is smaller, and therefore the overall fitness is less. For array values of (0,1000,...) only the false branch distance in the second iteration is needed. When all array values are zero, the event sequence is successfully executed and the overall fitness is zero.

For the deceptive example of Figure 2, the fitness landscape for the initial event sequence can be seen alongside that of $< (d, \{inverse\}), (a, \phi), (b, \phi) >$. As can be seen, the latter landscape is more conducive to the discovery of the required test data. With an input value of 500, nodes b and e are unexecuted. The approach levels and branch distances for these nodes are used to compute the fitness, according to rule 1. With an input value of 100, these nodes are still unexecuted, but the branch distances are smaller at the point at which control flow diverged away, being closer to zero, and therefore the overall fitness *is* smaller. At zero, all nodes are successfully executed.

In our work we have extended the data flow analysis utilized by the original Chaining Approach in order to find further program nodes that can influence the outcome at the problem node. The original Chaining Approach considers only last definitions of variables used at the current problem node. In our work, if this does not lead to the execution of the problem node in the desired way, last definitions of variables used at these last definitions are also considered. In practice this is performed by maintaining an *influencing set* of variables which can still affect the outcome at the current problem branching condition. (Unfortunately space restrictions prevent a full discussion of this algorithm here).

6 Results

Experiments were performed using different programs. Each contained a specific statement that could not be easily covered by ET, due to a specific "low probability" statement sequence required to be followed before the target is reached.

Table 1. Average fitness evaluations and running times for the experiments. The average number of fitness evaluations is the average sum of evaluations used by each evolutionary search for each event sequence for a test object.

	Average No. of Fitness Evaluations		Average Search Time (s)	
	200 generations	50 generations no improvement	200 generations	50 generations no improvement
Counter	547,064	213,546	157.9	67.1
Deceptive	63,684	23,907	10.2	4.8
Enumeration	314,551	105,523	48.6	18.0
Flag Assignment	121,772	56,687	21.3	11.2
Flag Loop Assignment	109,274	49,147	29.5	14.0
Flag Avoid Loop Assignment	81,592	45,562	23.7	14.7
Multiple Flag	228,892	83,867	39.3	16.2

Each problem statement was used as the target of the search using the hybrid approach. Although each program is of a relatively simple nature, the technical difficulties that will be encountered by ET are the same whether they are embedded in programs large or small. The actual size of a program is not directly responsible for increasing the complexity of the problem.

ET parameters applied by other authors in the field [3,10] were used for the evolutionary searches - namely 300 individuals per generation, comprising of 6 subpopulations with 50 individuals each; competition and migration across subpopulations; utilization of linear ranking; and a selective pressure of 1.7. Each experiment was repeated 50 times. The chaining tree was explored to a maximum depth of 5.

In the first run of experiments, searches were terminated after 200 generations. However it was found that for unsuccessful or infeasible event sequences, the search tended to stagnate well before the final generation was reached. Conversely, it was noted that for successful event sequences, test data could generally be found with an improvement on the previous best fitness value occurring within the last 50 generations. In an attempt to improve efficiency by saving on unnecessary fitness evaluations, a second run of the experiments was performed where each evolutionary search was terminated after 50 generations of no improvement. The results for the experiments are recorded in Table 1. The timing information reflects overall search times using a 1.3GHz PC with 512Mb of RAM. As can be seen, the latter termination criterion yielded the best performance, in some cases cutting search times by a third. This was achieved without comprising the reliability of the result. The following briefly describes each program and problem target statement, with further discussion of the results obtained.

Counter. This program is similar to the flag program of Figure 3 ("Flag Avoid Loop Assignment"), except a counter variable is used which keeps a total of the inputted ten array elements that are equal to zero. The target statement is executed if at least half of the array elements are zero. The range of the integers

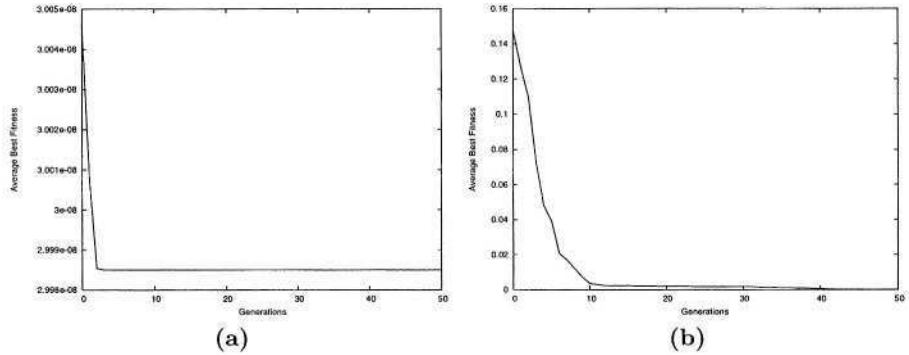


Fig. 6. Average best fitness plots for the deceptive fitness function program. (a) shows the search using the initial event sequence, with early stagnation. (b) shows the progress of the search for the successful event sequence

of the array was -15,000 to 15,000 giving a search space size of approximately 6×10^{44}. In all trials, eleven event sequences required consideration in order to find the required test data.

Deceptive. This is the program of Figure 2, with the target statement being that of node *b*. This experiment was carried out using an input range for x of -50,000 to 50,000 with a precision of 0.001, giving a search space size of approximately 10^8. The initial search stagnates early (Figure 6a). Event sequences are generated as outlined in Section 4. In all trials, the search succeeded with the event sequence that attempts to execute node *b* first (Figure 6b).

Enumeration. This decides whether three inputted color intensity values (integers in the range 0 to 255) represents one of the colors in an enumeration. A problem node occurs in the function under test when the color must be black. The search space size of approximately 1.6×10^7. The hybrid approach generated test data in all runs of the experiment. In five runs, the search fortuitously found test data for the initial event sequence. For all other trials, seven event sequences had to be considered before the one yielding test data could be found.

Flag Assignment. This function takes two double values. In searching for test data for one of the program statements, a flag has to be true. The flag is initially set to false, and is only set true when the first input value is zero. With an input range of -50,000 to 50,000 for each input variable, with a precision of 0.001, the search space size is approximately 10^{16}. The hybrid approach successfully generated test data in all trials, requiring the consideration of three event sequences.

Flag Loop Assignment. This program is similar to the program of Figure 3, except the flag which must be true for the target statement to be executed is set within the loop body when one or more of the ten inputted array values are zero. The range of the integers of the array was -15,000 to 15,000 giving a search space size of approximately 6×10^{44}. The hybrid approach generated test data in

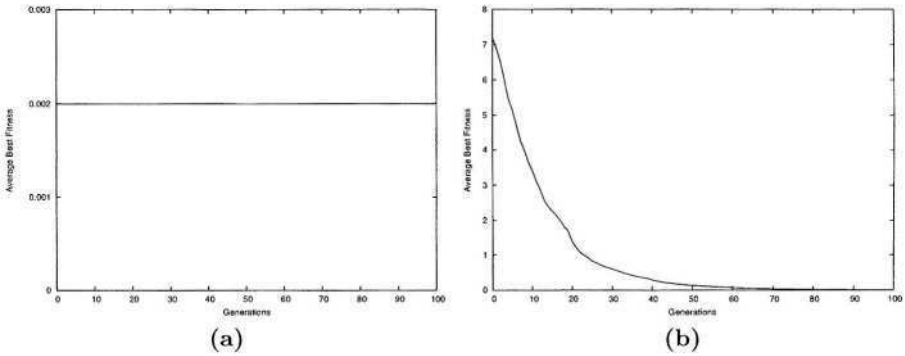


Fig. 7. Average best fitness plots for the flag avoid loop assignment program. (a) shows the search using the initial event sequence, which fails to improve on the best fitness of the initial generation. (b) shows the progress of the search for the successful event sequence

all trials. Generally three event sequences were evaluated, although in six runs, the search fortuitously found test data for the initial event sequence.

Flag Avoid Loop Assignment. This is the program of Figure 3, with the target statement being that of node f. Again, the range of the integers of the array was -15,000 to 15,000 giving a search space size of approximately 6×10^{44}. The initial search fails to improve on the best fitness value of the first generation (Figure 7a). Event sequences are generated as outlined in Section 4. The test data search for the event sequence that attempts to avoid node d was successful 94% of the time (Figure 7b).

Multiple Flag. This program involves two flags, both of which must be true in order to execute the target statement. The program takes two double input variables. Both flags are true if the first and second inputs are zero. However if the second double value is 1, the second flag is reset to false. Both double input ranges were -50,000 to 50,000 with a precision of 0.001 giving a search space size of approximately 10^{16}. The hybrid approach generated test data in all trials, evaluating five event sequences.

7 Conclusions

ET can often fail to test data for certain white-box test goals for types of programs, due to the lack of guidance provided by the search. This can occur when the target structure has data dependencies on previous statements which are only exercised under special circumstances. This paper argues that ET can be improved by incorporating ideas from the Chaining Approach, which forces the consideration of "events" leading up to the target structure. By instead searching for test data that executes an *event sequence,* the search can be directed into potentially untried, yet promising areas of the program's input domain. Experiments were carried out on seven programs. These programs - involving flags,

enumerations, counters and special function call return values - cause problems for the original ET approach. With the hybrid approach, test data could be successfully generated. Much of the literature in this area has solely concentrated on flag problems. The hybrid approach is general enough to tackle a wider range of programs which cause problems that are not just the result of flags.

Acknowledgements. This work is funded by DaimlerChrylser AG. The deceptive program (Figure 2) is an adaptation of an example presented by Mark Harman at the SBSE workshop in Windsor, UK, September 2002.

References

1. P. McMinn. Search-based software testing: A survey. *Software Testing, Verification and Reliability,* To appear, 2004/5.
2. L. Bottaci. Instrumenting programs with flag variables for test data search by genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002),* pages 1337 – 1342, New York, USA, 2002. Morgan Kaufmann.
3. M. Harman, L. Hu, R. Hierons, A. Baresel, and H. Sthamer. Improving evolutionary testing by flag removal. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002),* pages 1359–1366, New York, USA, 2002. Morgan Kaufmann.
4. A. Baresel and H. Sthamer. Evolutionary testing of flag conditions. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2003), Lecture Notes in Computer Science 2724,* pages 2442 – 2454, Chicago, USA, 2003. Springer-Verlag.
5. R. Ferguson and B. Korel. The chaining approach for software test data generation. *ACM Transactions on Software Engineering and Methodology,* 5(1):63–86, 1996.
6. B. Korel. Automated test generation for programs with procedures. In *International Symposium on Software Testing and Analysis (ISSTA 1996),* pages 209–215, San Diego, California, USA, 1996.
7. J. Wegener, A. Baresel, and H. Sthamer. Evolutionary test environment for automatic structural testing. *Information and Software Technology,* 43(14):841–854, 2001.
8. A. Baresel, H. Sthamer, and M. Schmidt. Fitness function design to improve evolutionary structural testing. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002),* pages 1329–1336, New York, USA, 2002. Morgan Kaufmann.
9. N. Tracey, J. Clark, and K. Mander. The way forward for unifying dynamic test-case generation: The optimisation-based approach. In *International Workshop on Dependable Computing and Its Applications,* pages 169–180, 1998.
10. A. Baresel, H. Pohlheim, and S. Sadeghipour. Structural and functional sequence test of dynamic and state-based software with evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2003), Lecture Notes in Computer Science 2724,* pages 2428 – 2441, Chicago, USA, 2003. Springer-Verlag.

Using Interconnection Style Rules to Infer Software Architecture Relations

Brian S. Mitchell, Spiros Mancoridis, and Martin Traverso

Department of Computer Science
Drexel University, Philadelphia PA 19104, USA,
{bmitchell, spiros, umtraver}@drexel.edu,
http://www.mcs.drexel.edu/~{bmitchel, smancori}

Abstract. Software design techniques emphasize the use of abstractions to help developers deal with the complexity of constructing large and complex systems. These abstractions can also be used to guide programmers through a variety of maintenance, reengineering and enhancement activities. Unfortunately, recovering design abstractions directly from a system's implementation is a difficult task because the source code does not contain them. In this paper we describe an automatic process to infer architectural-level abstractions from the source code. The first step uses software clustering to aggregate the system's modules into abstract containers called subsystems. The second step takes the output of the clustering process, and infers architectural-level relations based on formal style rules that are specified visually. This two step process has been implemented using a set of integrated tools that employ search techniques to locate good solutions to both the clustering and the relationship inferencing problem quickly. The paper concludes with a case study to demonstrate the effectiveness of our process and tools.

1 Introduction

Programmers are routinely given limited time and resources to perform maintenance on software systems. Without a good understanding of the software architecture maintenance tends to be performed haphazardly.

Software clustering techniques [11,13] have been used successfully to create abstract views of a system's structure. These views consolidate the module-level structure into architectural-level subsystems. The subsystems can be further clustered into more abstract subsystems, resulting in a subsystem hierarchy. While helpful, the subsystem structure exposes the architectural-level components (*i.e.,* subsystems), but not the architectural-level relations.

Like subsystems, architectural-level relations are not specified in the source code. To determine these relations we developed a technique that searches for them guided by a formalism that specifies constraints on allowable subsystem-level relations. This formalism, called ISF [10], is a visual language that can be used to specify a set of rules that collectively represent an interconnection style. The goal of an interconnection style is to define the structural and semantic

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1375–1387, 2004.

properties of architectural relations. Like our approach to software clustering [12, 13], computing the architectural relations from the subsystem hierarchy for a given interconnection style is computationally intractable. We overcome this problem by using search techniques.

After providing an overview of our integrated software clustering and style-specific relation inference processes and tools, we present a case study to show how our tools can assist with the maintenance of large systems.

2 Related Work

Researchers in the reverse engineering community have applied a variety of approaches to the software clustering problem. These techniques determine clusters (subsystems) using source code component similarity [17,4,15], concept analysis [9,7,1], or information available from the system implementation such as module, directory, and/or package names [2]. Our approach to clustering differs because we use metaheuristic search techniques [5] to determine the clusters [12, 13,14].

Research into Architectural Description Languages (ADLs), and their earlier manifestations as Module Interconnection Languages (MILs), provide support for specifying software systems in terms of their components and interconnections. Different languages define interconnections in a variety of ways. For example, in MILs [6,16] connections are mappings from services required by one component to services provided by another component. In ADLs [18] connections define the protocols for integrating sets of components. Our approach uses ISF [10], which is a visual formalism for specifying interconnection styles. We also developed a tool that can generate a Java program to check the well-formedness of ISF rules.

3 The Style-Specific Architecture Recovery Process

This section provides an overview of the style-specific architecture recovery process. Figure 1 outlines the process, which consists of using a series of integrated tools. To simplify the explanation of the process, we use a common reference system throughout this section to highlight important concepts.

JUnit is an open-source unit testing tool for Java, and can be obtained online from `http://www.junit.org`. JUnit contains four main packages: the framework itself, the user interface, a test execution engine, and various extensions for integrating with databases and J2EE. For the purpose of this example, we limit our focus to the framework package. This package contains 7 classes, and 9 inter-class relations.

3.1 Preparing the Software System

The first step in our process, shown on the left side of Figure 1, involves parsing the source code, transforming the source-level entities and relations into a

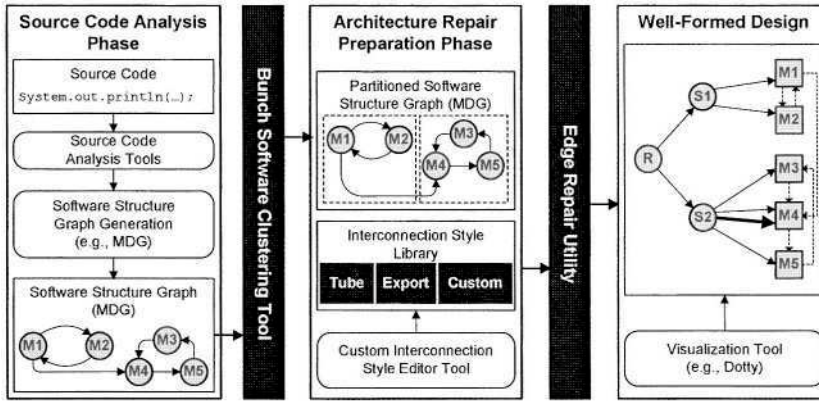


Fig. 1. The Style-Specific Architecture Recovery Environment

directed graph called the Module Dependency Graph (MDG). Readily available source code analysis tools – supporting a variety of programming languages – can be used for this step [3,8]. The MDG for JUnit, which was generated automatically, is illustrated on the left side of Figure 2.

3.2 Clustering the Software System

The software clustering process accepts the MDG as input, and produces a partitioned MDG as output. The Bunch tool can perform this activity automatically. Bunch uses search techniques to propose solutions based on maximizing an objective function [11,12,13,14]. The partitioned MDG is used as input into the architecture repair process, which is depicted in the top-center of Figure 1.

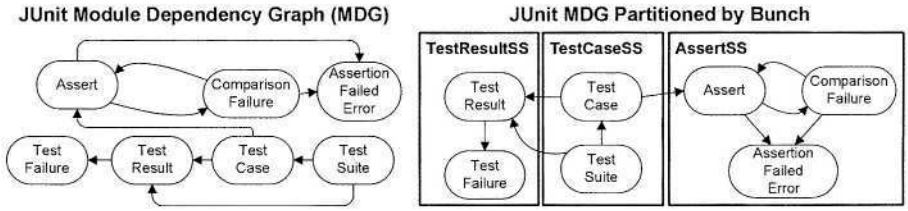


Fig. 2. The JUnit System Before and After Clustering

The right side of Figure 2 illustrates the JUnit system clustered into subsystems by Bunch. Even though this system is very small, there are still 877 unique ways to partition the MDG. Bunch produced the result shown in Figure 2 (which is the optimal solution) in 0.08 seconds, examining 47 partitions. The heuristic used to guide the search is based on an objective function that maximizes cohesion and minimizes inter-cluster coupling.

3.3 Inferring the Style-Specific Architectural Relations

The next step in the process (bottom-center of Figure 1) is to select an *interconnection* style to specify the allowable subsystem-level relations. For the purpose of this example, let's examine a useful interconnection style that we call the *Export Style.* The goal of the export style is to identify the *subsystem interfaces.* We define the subsystem interface to be the subset of the modules within a subsystem that provide services to modules in other subsystems. This information is helpful for determining the impact of a change to a subsystem.

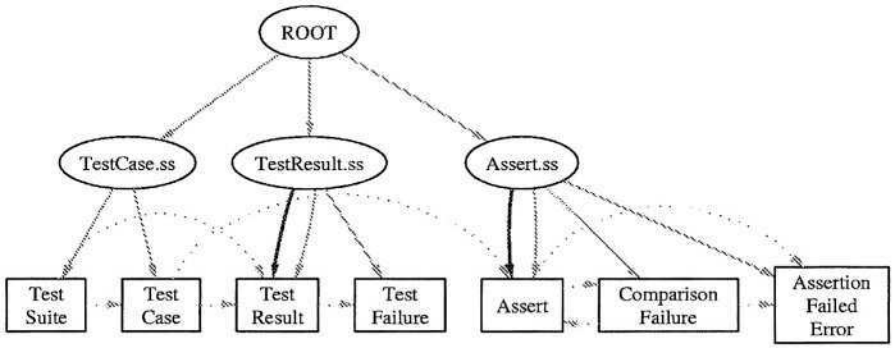


Fig. 3. Example Showing the **Export Style** Edges for JUnit

Figure 3 illustrates the result produced by our tool when the Export style is applied to the subsystem hierarchy shown in Figure 2. The leaf nodes in this figure are the modules in JUnit, and the dotted lines represent module-level dependencies (*i.e.,* the *use* relation). The intermediate elliptical nodes, connected by solid lines (*i.e.,* the *contain* relation) show the subsystem hierarchy. The results of recovering the inferred style-specific relations are denoted by the bold lines (*i.e.,* the *export* relation). Since the relation is of type Export, the bold lines are to be interpreted as: The `TestResult.ss` subsystem exports the `TestResult` module, and the `Assert.ss` subsystem exports the `Assert` module. We examine the usefulness of identifying the style-specific relations in further detail in Section 5.

4 Style-Specific Software Architecture Relations

This section describes a search process that is designed to compute architectural relations between the subsystems. To achieve this goal we developed a tool that enables software developers to specify *interconnection styles* formally. Interconnection styles allow designers to control the interactions between components by the use of rules and subsystem-level relations. Since the relations are not present in the recovered subsystem decomposition, our tool automatically infers

the subsystem-level relations that are missing in order to satisfy the constraints imposed by the interconnection style. The syntax for style specifications is based on the Interconnection Style Formalism (ISF) [10]. ISF allows for the definition of two kinds of rules:

1. *Permission* rules, which define the set of well-formed configurations of subsystems, modules, usage and architectural relations that adhere to a specific style.
2. *Definition* rules, which are used to define new relations based on patterns of components and relations.

4.1 Defining Styles

ISF enables designers to specify constraints on configurations of components and relations, and the semantics of such configurations. In the ISF notation, circles (nodes) represent system components (*e.g.,* modules, subsystems) and arrows (edges) represent relations between components (*e.g.,* import, export, use). The directed edges either depict a direct relation, or a transitive relation (represented using a double-headed arrow). Figure 4 shows the set of rules for the Export style using the ISF notation.

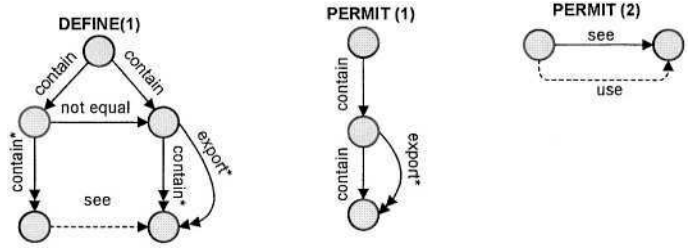


Fig. 4. Specification of the **Export Style**

In the Export style, *use* relations are extracted from the source code directly, as they represent the module-level dependencies. The clustering activity encapsulates modules into subsystems which is used to define the *contain* relations. The inferred *export* relations represent the subsystem interfaces. With this particular style a module may be used by other modules outside of its parent subsystem (container) if and only if the module is exported by its parent subsystem. Each module decorated with an inferred *export* relation is part of its parent subsystem's interface.

The Export style belongs to a family of styles that only allows relations between ancestors and decedents in the containment hierarchy. Another important family of styles are those that define relations between different subtrees in the subsystem containment hierarchy. An example of the latter family of styles is the *T*ube style, which is illustrated in Figure 5.

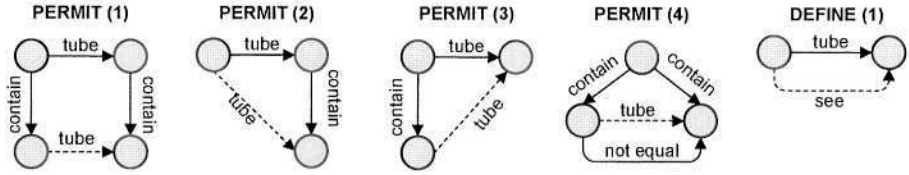


Fig. 5. Specification of the **Tube Style**

The *T*ube style definition specifies the tube relation, which enables a particular subsystem to "see" another subsystem. Specifically, two subsystems can be connected by a Tube if they are proper siblings, or if their parents are connected by a tubes according to the rules specified in Figure 5.

The ISF visual rules can be translated into formal specifications in a straightforward way. Given this capability we developed a tool that allows architectural styles to be defined using the ISF visual syntax, and then generates a Java program on the fly that is integrated dynamically into the search strategy to infer the desired style-specific relations. Figure 6 shows the GUI for the style editor, which depicts the ISF rules for the Import relation. The style editor can be used to create a reusable library of useful styles. More details on ISF Styles are available in an IJSEKE paper [10].

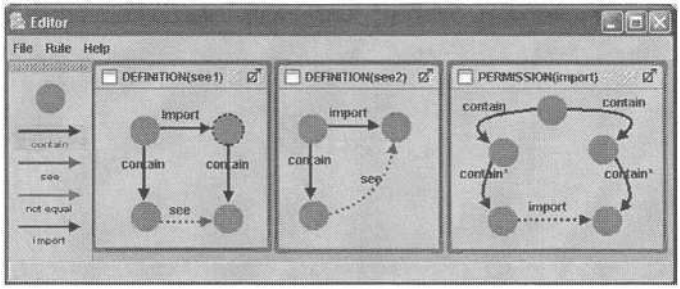


Fig. 6. The Custom Style Editor GUI

4.2 Style-Specific Edge Recovery

The goal of the style-specific edge recovery process is to take the subsystem containment hierarchy produced by the clustering process, along with a set of style rules specified in ISF, and locate the set of style relations that satisfies the ISF constraints. An exhaustive approach to recovering the style relations is to try all possible configurations permitted by the style and keep track of the one that has the fewest inferred relations (minimum visibility) that also satisfies all of the constraints of the style.

Unfortunately, exhaustive analysis is not possible. To understand why, let us consider a subsystem containment graph with N nodes. The maximum number of edges that can exist in the graph for each permitted relation type (e.g., export, import) is $E = N^2$ (i.e., one edge coming out of each node and into every other node, as well as the source node itself). If the style permits R different relation types, the graph can contain a total of $M = RE = RN^2$ style relations. What we need to know, however, is how many possible configurations exist for a specific style (based on the number of relations that the style defines).

Given that a style can contain a maximum of M total style relations, a particular configuration will introduce e style relations where $0 \leq e \leq M$. To determine how many total configurations exist for each style, we must consider how many configurations exist for when $e = 1, 2, \ldots M$. It turns out that the number of configurations is quite large:

$$\sum_{e=0}^{M} \binom{M}{e} = 2^M = 2^{RN^2}$$

Because considering all possible configurations is not possible, we use search techniques to find good (if not optimal) solutions to the style-specific relation recovery problem. The remainder of this section will focus on an approach[1] that uses a hill-climbing algorithm.

The hill-climbing approach for recovering the style-specific relations is straightforward. The first step generates a random configuration. Incremental improvement is achieved by evaluating the quality of neighboring configurations using an objective function. The set of neighboring configurations for a given configuration are determined by adding or removing a single style edge. The search iterates until no new neighboring configurations can be found with a higher objective function value. Figure 7 illustrates how neighboring configurations can be generated for a given configuration (left), by adding and removing export style relations.

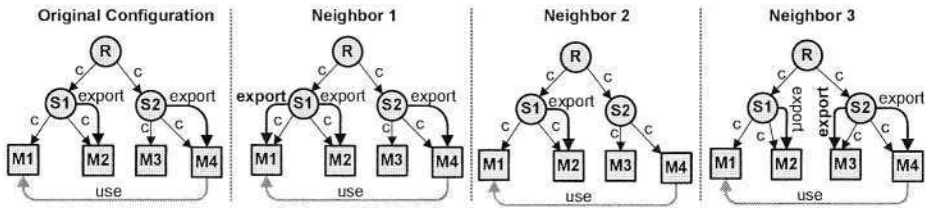


Fig. 7. Example Showing the Generation of Neighboring Configurations

Before the objective function is introduced, we need to provide several important definitions. Every configuration considered during the search will have a set

[1] We have investigated other techniques that are more efficient than hill-climbing, however they only work on certain styles.

of *use* and *style* relations. The *use* relations are extracted from the source-level dependencies and the *style* relations are defined by the ISF specification. We define *MaxS* as the maximum number of style relations that can exist for a particular configuration. Each *style* relation can either be well-formed, or ill-formed. We define *wfs* as the total number of style relations that respect the rules of the provided ISF specification, and *ifs* as the total number of style relations that violate the rules of the ISF specification. As for the *use* relations, *wfu* is defined as the number of relations that are correctly modeled by the provided ISF style, and *ifu* is the number of *use* relations that violate the ISF style.

Given the above definitions the objective function is engineered to maximize *wfu,* minimize *ifs,* and minimize visibility. Minimizing visibility by keeping the set of inferred style edges as small as possible is important since not all *wfs* edges are necessary to cover the set of *use* edges for a particular system. It should also be noted that the objective function maximizes *wfu* by minimizing *ifu* since a given use edge must be classified as either well- or ill-formed. The objective function to measure the quality of a particular configuration is shown below:

$$\text{quality}(C) = \begin{cases} \frac{\text{wfs}}{\text{ifs}+\text{ifu}} & \text{ifs} \neq 0 \ \textit{or} \ \text{ifu} \neq 0 \\ \text{MaxS} + \frac{1}{\text{wfs}} & \text{ifs} = 0, \text{ifu} = 0, \text{wfs} \neq 0 \\ \text{MaxS} + 2 & \text{ifs} = 0, \text{ifu} = 0, \text{wfs} = 0 \end{cases}$$

5 Case Study

In this section we describe how we used our tools to gain insight into the design of Bunch, a clustering tool created by our research group.

Bunch is a research tool, and as such, is prone to frequent restructuring and extension due to changes in requirements. The original version implemented the core clustering algorithms and had no graphical user interface (GUI). Later, it was redesigned to make it extensible, and several features, including a GUI were added. Adding new clustering algorithms was easier in this version. Following that, an attempt was made to improve the performance of the algorithms by implementing a distributed version of the tool. Finally, the core algorithms were replaced with much faster incremental algorithms. An important fact about the development process is that each stage was carried out by different developers.

Without the help of reverse engineering tools, understanding the structure of the current version of Bunch and how its components relate to each other would be quite difficult. It is important that future maintainers be able to understand the system well enough to be able to make changes without breaking the application.

The first thing that we did in this case study was to obtain Bunch's MDG, and then use Bunch to cluster the MDG. We then based our analysis on a modified version of the Export style, which is depicted in Figure 8. This style permits two types of export relations: local export and broad export. Local export relations enable a module to be seen by subsystems in the immediate neighborhood (the parent subsystem's siblings). Broad export relations export modules

to more distant branches of the tree. Local export allows a subsystem to export only child nodes, whereas broad export dependencies allow subsystems to export only modules deeper than the grandchildren level, thereby allowing modules in entirely different branches of the tree to "see", and hence access, them.

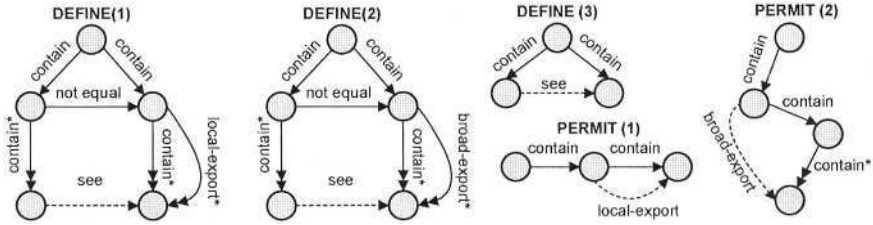


Fig. 8. Specification of the **Modified Export Style**

The results produced by our tool allow us to make several observations about the structure of the Bunch application. Subsystems that export a high percentage of their modules are particularly noteworthy, since they might reflect poor design choices. Sometimes, these subsystems might contain groups of modules that are used throughout the application and are considered to be libraries. Such situations are not bad, but raise the question whether those modules can be split into smaller modules that perform more specific functions and can be placed alongside modules that use them. Other times, those situations occur because modules have been classified into the wrong subsystems. For example, the `Graph` subsystem shown on the left side of Figure 9 contains the modules `TurboMQ` and `TurboMQIncr`, which are fitness function evaluation classes, unlike the `Graph` and `Cluster` classes which are important global data structures. This means that both of the `TurboMQ` classes have probably been misplaced in the `Graph` cluster, and would be better suited in the `ObjectiveFunctionCalculator` subsystem[2] instead.

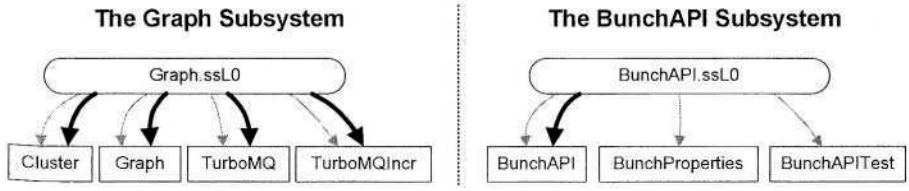


Fig. 9. Two of Bunch's Subsystems Decorated with Export Relations. The Light Edges are the Containment Relations and the Dark Edges are the Export Relations.

[2] This subsystem is not shown but it is present in the subsystem hierarchy generated by Bunch.

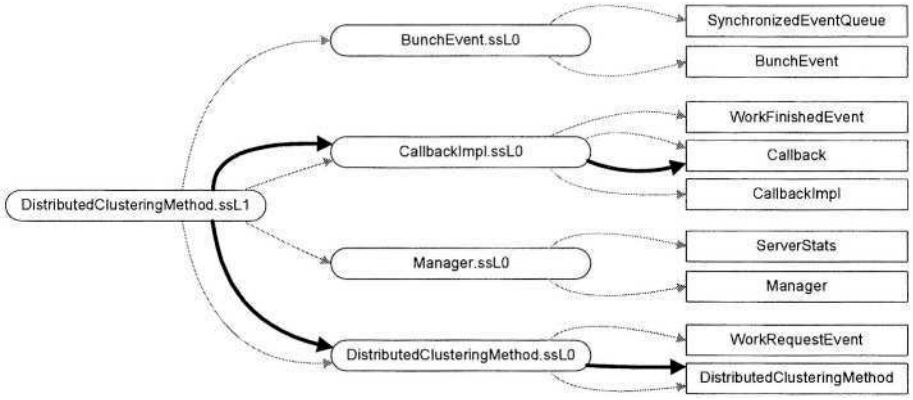


Fig. 10. The Bunch Subsystem that Supports Distributed Clustering

Since the clustering was automatically produced by Bunch, this error is probably related to the fact that the clustering algorithm does not take into account style-related properties. This is an example of how by considering the style relationships one can complement clustering tools. Regardless of whether the clustering was produced manually or by a clustering tool, a configuration where a subsystem exports a high percentage of its classes warrants further investigation.

The existence of export relations, or lack of them, gives us information on how a change to a module might affect the rest of the system. In particular, if a module is local to a subsystem (*i.e.,* it is not exported), it is clear that making modifications to it should not affect modules that are outside of the subsystem where it belongs. Maintenance efforts can be concentrated on a small subset of the system. The existence of an export dependency indicates that changes to the module may affect a wide range of modules in the system, and, thus, care must be taken when modifying the module. Even with a subsystem decomposition produced by a clustering algorithm, it is not obvious for all but the most trivial–sized systems, which modules encapsulated in a subsystem are only dependant on other modules within the subsystem. As an example, on the right side of Figure 9, we have confidence that an update to the `BunchProperties` class can at most affect the `BunchAPI` and the `BunchAPITest` classes.

Figure 10 shows the subsystem responsible for distributed clustering. Let us imagine that we want to add a new function to Bunch that depends on this subsystem. The first thing that we need to know is the interface to this subsystem. Since the documentation is often outdated or missing altogether, the most reliable way to find the information is by browsing the source code; a really tedious task, especially if the subsystem is complex. Export relations might ease the burden of such a task, since these modules are essentially the access points to the subsystem. In the case of the `DistributedClusteringSS` subsystem, we notice that the exported modules are `CallbackImpl` and `DistributedClustering-Method`. Thus, we only need to concentrate on those modules when trying to

learn how to use this subsystem, and may ignore the rest of the modules since they are not visible outside of the subsystem.

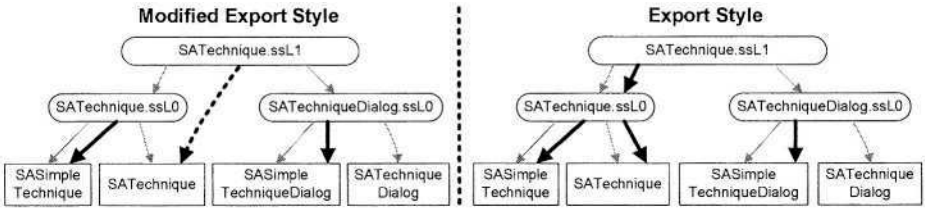


Fig. 11. The Simulated Annealing Subsystem with Local/Standard Export (bold arrow) and Broad Export (dashed arrow) Relations. The Left Side was Generated Using the Modified Export Style, and the Right Side was Generated Using the Standard Export Style.

The containment hierarchy of Bunch has a depth of four levels. This means that the application is composed of several major subsystems, which in turn, are composed of smaller subsystems. Some modules in these smaller subsystems are intended for local use, whereas, other modules are intended for use outside of the major subsystems (*i.e.,* they are part of the subsystem interface). The modified export style defined in Figure 8 can help us understand which modules play each of these roles. The `SATechniqueSS` subsystem depicted on the left side of Figure 11 implements a simulated annealing algorithm and was added quickly to try this idea. It exports module `SATechnique` which comprises the subsystem's interface. On the other hand, modules `SASimpleTechnique` and `SASimpleTechniqueDialog`, which are exported outside the smaller subsystems but not out of the `SATechnique` subsystem are intended for use only within `SATechnique`. Note that if we had analyzed this subsystem with the original Export style, we would have seen that `SASimpleTechnique` would have also been exported out of `SATechnique`, as shown in the right side of Figure 11. This outcome increases the visibility of the `SASimpleTechnique` module, giving the maintainer no choice but to assume that this module may be used from any of Bunch's subsystems.

The Bunch system contains 48K LOC packaged in 220 classes, with 764 inter-class relations. Our tools clustered Bunch in 0.57 seconds, and recovered the style-specific relations for the modified export style in 2.86 seconds. Although space prohibits a more detailed analysis, the few patterns that we examined in the clustered decomposition with the recovered style-specific relations gave us significant information on how Bunch is structured and how its modules relate to each other.

6 Conclusion

This paper highlighted techniques and tools that can be used to assist software maintainers tasked to fix, enhance or reengineer large and complex existing systems. These techniques focus on extracting abstract architectural-level artifacts directly from the detailed system implementation. This is a hard problem since these abstractions are not specified in the source code.

Ongoing research in software clustering has shown promising results for recovering useful architectural-entities directly from the source code. However, while software clustering techniques simplify understanding the relationships between modules grouped into the same subsystems, these techniques provide little help for understanding the complex inter-subsystem relationships. To address this need, this paper presented an integrated process that builds on existing software clustering technology by using formal style rules to infer architectural-level relations between subsystems.

References

1. N. Anquetil. A comparison of graphis of concept for reverse engineering. In *Proc. Intl. Workshop on Program Comprehension,* June 2000.
2. N. Anquetil and T. Lethbridge. Recovering software architecture from the names of source files. In *Proc. Working Conf. on Reverse Engineering,* October 1999.
3. Y. Chen. Reverse engineering. In B. Krishnamurthy, editor, *Practical Reusable UNIX Software,* chapter 6, pages 177–208. John Wiley & Sons, New York, 1995.
4. S. Choi and W. Scacchi. Extracting and restructuring the design of large systems. In *IEEE Software,* pages 66–71, 1999.
5. J. Clark, J. Dolado, M. Harman, R. Hierons, B. Jones, M. Lumkin, B. S. Mitchell, S. Mancoridis, K. Rees, M. Roper, and M. Shepperd. Reformulating Software Engineering as a Search Problem. *Journal of IEE Proceedings - Software,* 150(3):161–175, 2003.
6. F. DeRemer and H. H. Kron. Programming in the Large Versus Programming in the Small. *IEEE Transactions on Software Engineering,* 2(2):80–86, June 1976.
7. A. van Deursen and T. Kuipers. Identifying objects using cluster and concept analysis. In *International Conference on Software Engineering, ICSM'99,* pages 246–255. IEEE Computer Society, May 1999.
8. J. Korn, Y. Chen, and E. Koutsofios. Chava: Reverse engineering and tracking of java applets. In *Proc. Working Conference on Reverse Engineering,* October 1999.
9. C. Lindig and G. Snelting. Assessing modular structure of legacy code based on mathematical concept analysis. In *Proc. International Conference on Software Engineering,* May 1997.
10. S. Mancoridis. ISF: A Visual Formalism for Specifying Interconnection Styles for Software Design. *International Journal of Software Engineering and Knowledge Engineering,* 8(4):517–540, 1998.
11. S. Mancoridis, B.S. Mitchell, C. Rorres, Y. Chen, and E.R. Gansner. Using automatic clustering to produce high-level system organizations of source code. In *Proc. 6th Intl. Workshop on Program Comprehension,* June 1998.
12. B. Mitchell. *A Heuristic Search Approach to Solving the Software Clustering Problem.* PhD thesis, Drexel University, Philadelphia, PA, USA, 2002.

13. B. S. Mitchell and S. Mancoridis. Using heuristic search techniques to extract design abstractions from source code. In *Proceedings of Genetic and Evolutionary Computation Conference,* 2002.
14. B. S. Mitchell and S. Mancoridis. Modeling the search landscape of metaheuristic software clustering algorithms. In *Proceedings of Genetic and Evolutionary Computation Conference,* 2003.
15. H. Müller, M. Orgun, S. Tilley, and J. Uhl. A reverse engineering approach to subsystem structure identification. *Journal of Software Maintenance: Research and Practice,* 5:181–204, 1993.
16. R. Prieto-Diaz and J. M. Neighbors. Module Interconnection Languages. *The Journal of Systems and Software,* 6:307–334, 1986.
17. R. Schwanke and S. Hanson. Using Neural Networks to Modularize Software. *Machine Learning,* 15:137–168, 1998.
18. M. Shaw, R. DeLine, D. V. Klien, T. L. Ross, D. M. Young, and G. Zalesnik. Abstractions for Software Architectures and Tools to Support Them. *IEEE Transactions on Software Engineering,* 21, April 1995.

Finding Effective Software Metrics to Classify Maintainability Using a Parallel Genetic Algorithm

Rodrigo Vivanco[1,2] and Nicolino Pizzi[1,2]

[1]Institue for Biodiagnostics, National Research Council Canada, Winnipeg, MB, Canada
{Rodrigo.Vivanco, Nicolino.Pizzi}@nrc-cnrc.gc.ca
[2]University of Manitoba, Winnipeg, MB, Canada
{rvivanco, pizzi}@cs.umanitoba.ca

Abstract. The ability to predict the quality of a software object can be viewed as a classification problem, where software metrics are the features and expert quality rankings the class labels. Evolutionary computational techniques such as genetic algorithms can be used to find a subset of metrics that provide an optimal classification for the quality of software objects. Genetic algorithms are also parallelizable, in that the fitness function (how well a set of metrics can classify the software objects) can be calculated independently from other possible solutions. A manager-worker parallel version of a genetic algorithm to find optimal metrics has been implemented using MPI and tested on a Beowulf cluster resulting in an efficiency of 0.94. Such a speed-up facilitated using larger populations for longer generations. Sixty-four source code metrics from a 366 class Java-based biomedical data analysis program were used and resulted in classification accuracy of 78.4%.

1 Introduction

Software project managers commonly use various metrics to assist in the design and implementation of large software systems [1,2]. These metrics are used to quantify the various developmental stages of the project; design metrics such as the number of classes, level of inheritance, number of abstract classes and so on; implementation metrics obtained automatically from the source code, e.g., lines of codes per method, lines of comments, number of methods in a class, method complexity, number of tokens, and so on; test metrics such as the number of errors reported; and usability metrics like the number of times the user pressed the wrong button or asked for help. One important aspect of project management is the ability to identify potential problems in the design and development phases from the source code as the application is implemented. Project managers and developers can use source code metrics to model and predict the quality of the software as the application evolves.

The use of computational intelligence methods such as neuro-computing, fuzzy computing and evolutionary computing in software engineering is being recognized as an important tool for software engineers and project managers [3]. Due to the attributes of the problem, software engineering is stochastic in nature were the skill set and experience of developers and project leaders play a large factor in the final overall quality of the software. As such, search-based techniques such as evolutionary

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1388–1399, 2004.

computing are drawing the attention of researchers to help develop and fine-tune software engineering tools [4].

The ability to evaluate the quality of a software object can be viewed as a classification problem. Given a set of objects (object-oriented classes), with known features (source code metrics) and class labels (expert quality rankings) build a classifier that is able to predict the quality of a software object from its metrics. Not all metrics have the same discriminatory power when it comes to predicting the quality of a software object. For example, the number of semicolons may not be as powerful as the number of lines of code, which may not be as powerful as a measure of the amount of coupling, cohesion and complexity in predicting the quality of a software object in terms of maintainability. Also, the combination of metrics may be more important than any individual metric, a previously unforeseen combination of metrics may capture the insight that an expert uses when attributing a quality ranking to a software object. Determining which metrics, and combination of metrics, have strong discriminatory powers is paramount for generating a good classifier. However, finding an effective combination of metrics that have good discriminatory properties is not a problem that can be solved analytically as the number of metrics increases.

Genetic algorithms have been used extensively to find feature subsets in classification problems in the biomedical field [5,6,7]. However, computational intelligence techniques have not been used as extensively in the software engineering domain. [8] used a genetic algorithm to find the optimum neural network architecture for a two-class problem and identify fault-prone modules using nine software metrics. [9] utilized a genetic algorithm approach to find a model that best classifies fault-prone modules based on 5 metrics. [10] exploited genetic algorithms to determine software stability, how much a Java class changed over the course of the project, using 11 metrics based on coupling, cohesion, inheritance and complexity.

The objective of this report is two fold. First, to illustrate how a genetic algorithm was used as an effective feature sub-selection strategy to a classification problem in the software engineering domain, that is, determine which subset of object-oriented source code metrics are able to predict the quality of a software object in terms of maintainability. Secondly, to show how a parallelized version of the canonical genetic algorithm was implemented using the Message Passing Interface (MPI) library [11]. The speed up of the program execution enabled more combinations of GA parameters, such as population size, mutation rate and number of generations to be tried within a reasonable amount of time for the researcher, which facilitated finding an optimal solution.

2 Software Metrics

All 366 software objects in an in-house Java-based biomedical image data analysis system, were subjectively labeled by an experienced software architect in terms of maintainability. The architect (8 years of programming experience, 7 years with object-oriented systems and 5 years with Java) was asked to rank each software object. That is, based on personal experience, assign a value from 1 to 5 that tries to rank the overall design and implementation of a particular software object. Software objects with low rankings are determined to be difficult to modify, and should be reviewed by the development team in efforts to improve the class, either by

improving the design (which may mean refactoring classes) or improving the implementation of the methods. A class ranked 1 should definitely be subject to a review as soon as possible. A ranking of 2 meant the class indicated that the class should be reviewed but it is not critical that it be done immediately. A ranking of 3 indicates, in the opinion of the expert, an average design and implementation, neither exemplary or detrimental to product quality and maintenance. A class ranked 4 is better than average but could use some improvements (for example, better documentation). A class ranked 5 is considered very easy to understand and modify. The software architect was not instructed to focus on any particular aspects of code quality (in reference to metrics that can be measured) but to use his experience and intuition to rank the software classes. Table 1 shows the distribution of the labeled software objects by the expert familiar with the project in terms of design and implementation.

Table 1. Distribution of software object labels assigned by an expert

Rank 1	Rank 2	Rank 3	Rank 4	Rank5
2	56	75	94	139

The project has been in development for over 24 months and low ranking classes have been identified and corrected so few „must review“ software objects are present in the current dataset. This will bias the classifier towards properly identifying high ranking classes. The majority of the software classes ranked at level 5 are simple data-model classes with simple get/set methods, they basically encapsulate classes are not highly coupled to other classes and their method complexity are low.

A set of 64 metrics was obtained from an evaluation version of Borland's TogetherSoft package [12] and an in-house metrics parser. Due to the nature of the application, it was noted that data model objects were relatively simple when compared to graphical user interface (GUI) classes, so an additional metric value was used to aid the classifier, a distinction was made between GUI, data model, algorithm, and all other objects. In general, data model classes had many get/set methods and low coupling, while GUI classes had higher coupling among themselves and the data that was to be displayed. The software metrics that were used to classify the Java software objects are shown in Table 2.

Table 2. The 64 object-oriented source code metrics used as features for the classifier

Description	Metrics
TYPE	Type: GUI (=1), Data Model (=2), Algorithm (=3), Other (=4).
METH	# methods.
LOC	# lines of code.
ALOC	Mean LOC per method.
MLOC	Median LOC per method.
RCC1	Ratio of comment lines of code to total lines of code including white space and comments.
RCC2	TCR from Borland's TogetherSoft
TOK	# tokens.
ATOK	Mean TOK per method.
MTOK	Median TOK per method.

DEC	# decisions: for, while, if, switch, etc.
ADEC	Mean DEC per method.
MDEC	Median DEC per method.
WDC	Weighted # decisions based on nesting level i: Sum[i*n_i]
AWDC	Mean WDC per method.
MWDC	Median WDC per method.
INCL	# inner classes.
DINH	Depth of inheritance.
CHLD	# children.
SIBL	# siblings.
FACE	# implemented interfaces.
RCR	Code reuse: ratio of overloaded inherited methods to those that are not.
CBO	Coupling between objects.
LCOM	Lack of cohesion of methods.
RFO	Response for an object. Response set contains the methods that can be executed in response to a message being received by the object.
RFC	Response for class.
MNL1	Maximum method name length.
MNL2	Minimum method name length.
MNL3	Mean method name length.
MNL4	Median method name length.
ATCO	Attribute Complexity.
CYCO	Cyclomatic Complexity.
DAC	Data Abstraction Coupling.
FNOT	Fan Out.
HLDF	Halstead Difficulty.
HLEF	Halstead Effort.
HLPL	Halstead Program Length.
HLVC	Halstead Program Vocabulary.
HLVL	Halstead Program Volume.
HLON	Halstead # operands.
HLOR	Halstead # operators.
HLUN	Halstead # unique operands.
HLUR	Halstead # unique operators.
MIC	Method Invocation Coupling.
MAXL	Maximum # levels.
MAXP	Maximum # parameters.
MAXO	Maximum size operations.
ATTR	# attributes.
ADDM	# added methods.
CLAS	# classes.
CHCL	# child classes.
CONS	# constructors.
IMST	# import statements.
MEMB	# Members.
OPER	# operations.
OVRM	# overridden methods.
REMM	# remote methods.
PKGM	% package members.

PRVM	% private members.
PROM	% protected members.
PUBM	% public members.
DEMV	Violations of Demeters Law.
WMC1	Weighted Methods per class.
WMC2	Weighted methods per class.

3 Parallel Genetic Algorithm

Evolutionary computation algorithms such as genetic algorithms (GA) attempt to discover an optimal solution to a problem by simulating evolution [13]. In GA, a solution (set of software metrics) is encoded in a gene and a collection of genes (solutions) constitutes a population. GA uses natural selection and genetics as a basis to search for the optimal gene, a set of software metrics that give the best classification rate. A population of solutions is modified using directed random variations and a parent selection criteria in order to optimize the solution to a problem. They are based on the process of Darwinian evolution; over many generations, the "fittest" individuals tend to dominate the population.

The simplest way to represent a gene is to use a string of bits, where 0 means the bit is off and 1 means the bit is on. For this problem domain, the 64 metrics were encoded in a 64-bit string. A zero bit meant the metric was not to be used with the classifier. All the metrics with a corresponding bit set to one constituted the metrics sub-set to evaluate with the fitness function, the classifier. The genes of the initial population were randomly initialized.

To evaluate a gene's fitness a linear discriminant analysis (LDA) classifier was utilized using the leave-one-out method of training and testing. LDA is a conventional classifier strategy used to determine linear decision boundaries between groups while taking into account between-group and within-group variances [14]. If the error distributions for each group are the same (each group has identical covariance matrices and sampled from a normal population), it can be shown that linear discriminant analysis constructs the optimal linear decision boundary between groups. Figure 1 shows a three-class 2-dimensional classification problem and the decision hyper-planes produced by LDA. For the leave-one-out validation method, 365 classes were used to train the LDA, and then the software object left out was tested with classifier. This was repeated for all 366 software objects. The classification rate was the number of times the software object left out to be tested was correctly classified by the LDA. The final classification rate for a feature subset is a value between 0 and 100%.

The genes with higher fitness values are more likely to be chosen for reproduction and evolve into the next generation. For the implemented GA, a random probability was generated, and a random gene with an equal or larger fitness value was chosen as a parent. All the genes in the population were candidate parents. A child gene is created by picking a crossover point and exchanging the corresponding bits from the two parent genes. A single crossover point was randomly chosen for the creation of the child gene. Mutation was performed by flipping the value of each bit in the child

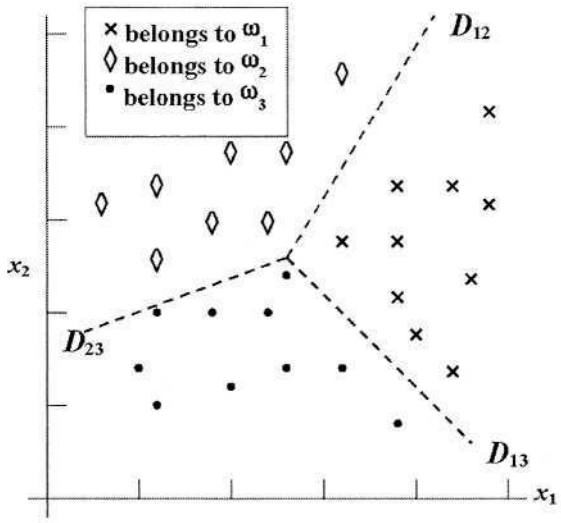


Fig. 1. Possible linear discriminant analysis decision boundaries for three groups

gene if the random probability value is greater than 1.0 - P, P being the user specified mutation probability parameter. The number of genes in the population is also a user set parameter and did not change during successive generations.

A new population is created by merging the children with the elite genes of the previous population. The number of elite genes is a parameter set by the user. The new population is sorted based on fitness values and some of the children may now be in the elite pool. The elite pool is only used for merging the next generation, as the entire population is eligible for reproduction.

The canonical genetic algorithm is an ideal candidate for coarse grain domain partition parallelization in an effort to improve the computational speed for finding a solution to a non-deterministic problem [15]. The fitness function for a solution gene can be executed independently of the other genes in the population. That means one process can calculate the fitness function for one gene, while another process does the same for another gene in parallel. After the current children population is evaluated, the previous generation and current generation are merged. The most computationally intensive function for the canonical GA is usually the calculation of a gene's fitness value. Figure 2 illustrates the basic modification of the sequential GA to make it parallelizable using a manager/worker approach.

The sequential and parallel versions of the canonical genetic algorithm were implemented using the C++ language. The MPI library which facilitates the message passing of data between processes in a parallel computer or a cluster of computers using C-style function calls. It is a well-established protocol and widely implemented on various operating systems. The parallel GA program was tested on a 20 node heterogeneous Beowulf cluster running the Linux operating system. For the sequential GA program the population was set to 100, the percent elite to 25%, mutation rate at 5% and the number of generations capped to 150. For the parallel version, which

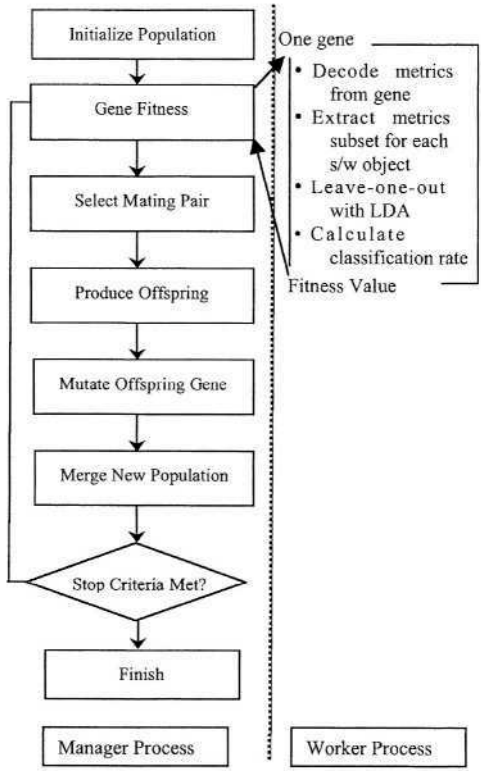


Fig. 2. Manager/worker parallel version of the canonical sequential genetic algorithm

executed in a significantly shorter amount of time, a range of population sizes were tested for various generations, with the same mutation rate and percent elite as the sequential GA.

Using a manager/worker paradigm, the parallel section of the algorithm is executed by a different worker process while the main tasks of merging the parent and children population, creating new genes and testing for the termination condition is done by the manager process. In the method where the population fitness is calculated, the manager process repeatedly receives a fitness value from a worker process, and if there are genes still to be evaluated, sends one to the idle worker process. Below is the pseudo code for method in the manager that calculates the fitness for all the genes in a population.

```
void calculate_population_fitness( population ) {
  num_workers = number of processes in MPI network
  //initialize workers, give each one a gene to process
  num_genes_evaluated = 0
  for (worker_id=0;worker_id<num_workers;worker_id++){
     // send a gene, and matching gene_id, to a worker
     MPI_Send( gene_bits, gene_id, worker_id )
```

```
      num_genes_evaluated += 1
    }
    // while there is work left, get fitness value from a
    // worker, give the worker another gene to evaluate
    while ( num_genes_evaluated < population_size ) {
      // get fitness from a worker
      MPI_Receive( fitness, gene_id, worker_id )
      set fitness in population using gene_id
      // send another gene to the worker
      MPI_Send( gene_bits, gene_id, worker_id )
      num_genes_evaluated += 1
    }
    // no more genes to evaluate, collect remaining
    // results from workers
    for (worker_id=0;worker_id<=num_workers;worker_id++){
      MPI_Receive( fitness, gene_id, worker_id )
      set fitness in population using gene_id
    }
  } // end of method
```

The worker process decodes the gene and generates a new metrics dataset. For all the software objects, only the metrics with a corresponding bit set to one are used with the classifier. Using LDA with the leave-one-out train/test technique the classification rate using the metrics subset encoded in the gene is returned as the fitness value. Once all the genes in a population are evaluated, the manager process continues with the canonical GA in a sequential manner. The algorithm is repeated until the specified number of generations is executed.

4 Results and Observations

The parallel GA program runs substantially faster than the sequential version. It took 14.5 hours to run a 100 gene population for 150 generations with the sequential program, running on a 1.6 Ghz CPU with 2 Gigs of RAM. Using the same GA parameters with the parallel version on the Beowulf cluster took 1.3 hours. With such a speed up more variations of GA parameters could be tried. Figure 3 shows the classification rate when all the available metrics were used with the classifier, the best classification from a set of 100 random metrics subsets, and the best classification when different generations were used with 100 genes, 5% mutation rate and 25% elite genes. Modifying the percent elite and mutation rates did not result in noticeable better classification rates, though increasing the size of the population did. An improved classification rate was obtained when the number of generations was increased.

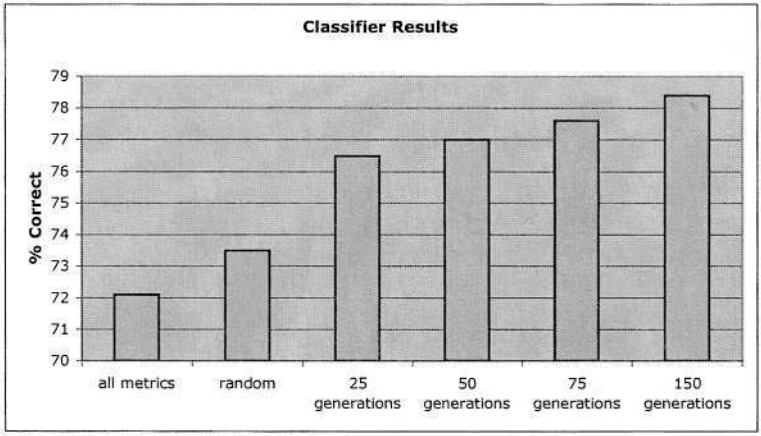


Fig. 3. Classification rates when all metrics are used, random subsets, and GA feature selection

Table 3 shows the metrics encoded by the top 3 genes, they provided the best classification for the software classes with a percent correct of 78.4%, 78.1% and 77.9%. Different genes encode slightly different metrics yet result in comparable classifier performance. It is interesting to note the metrics that were common to all 3 genes. ALOC, the mean number of lines would seem to indicate that the size of a method affects maintenance, which intuitively makes sense, along with other complexity metrics such as HLDF. One of the software objects that was ranked as must review had over 6,000 lines of code. It was commented that this class seemed like an application onto itself. It was the main data visualization GUI class for the application from which the user could select further processing options.

The MNL1 metric, method name length, would suggest that using longer names for methods (and variables, though that is not a metric used in this study) facilitates the understanding of the source code by other programmers, and thus affects the maintainability of the code, since longer labels names tend to indicate more meaningful names which express the purpose of the method and the variable. The other method name length metrics, such as MNL2, MNL3 and MNL4 are variations of the same information captured by the MNL1 metric. The other classes ranked 1 had 900 lines of code but was very poorly documented and highly coupled to other classes making the purpose of the interdependencies not clear to the reviewer.

Coupling measures were also encoded in the top genes. Indirect measures such as package members, PKGM, indicate potentially shared data members among classes. Inheritance measures CHCL (number of children) and FNOT (fan out) are also present in the metrics subset. Only the top gene explicitly used the coupling CBO metric from the CK metrics suite [16]. The only comments based metric, RCC1, was not included in the top three genes though it was included in other genes that performed well (77.6%). It is generally accepted that comments aid in the maintenance of software, and if more comment based metrics would have been generated the chances of top performing genes including that metric would have increased.

Table 3. Metrics encoded by top 3 genes and corresponding classification rates

Description	78.4%	78.1%	77.9%
TYPE	YES		
LOC	YES		YES
ALOC	YES	YES	YES
TOK			YES
ATOK	YES		YES
MTOK		YES	YES
DEC	YES		
ADEC		YES	YES
MDEC		YES	YES
WDC		YES	
AWDC	YES	YES	
DINH	YES	YES	
CHLD	YES		
FACE	YES	YES	
RCR	YES		
CBO	YES		
LCOM		YES	YES
MNL1	YES	YES	YES
MNL3		YES	
CYCO	YES	YES	
FNOT		YES	
HLDF	YES	YES	YES
HLEF	YES		
HLPL	YES	YES	YES
HLVL	YES	YES	YES
HLON		YES	
HLUN		YES	YES
MIC			YES
MAXL			YES
MAXP	YES		YES
MAXO	YES	YES	YES
ATTR	YES	YES	YES
ADDM		YES	YES
CLAS			YES
CHCL			YES
MEMB	YES		
OPER		YES	YES
OVRM	YES	YES	
PKGM	YES	YES	YES
PRVM	YES	YES	
PROM	YES	YES	YES
PUBM	YES	YES	
DEMV	YES		
WMC1	YES	YES	
WMC2		YES	YES

Figure 3 shows that using a subset of the metrics helps the classifier achieve better performance, as using a random subset results in improved performance over using all the available metrics, some of which capture redundant information about the source code. Using a directed search like a genetic algorithm enhances the classifier performance even further, and the longer the search the better the classification rate. Using a parallel genetic algorithm significantly reduces the computational time of the algorithm and facilitates longer searches. The resulting metrics subset that capture the intuitive knowledge of the expert can be further inspected and analyzed from a theoretical aspect to understand what aspects of design and implementation lead towards high quality software objects.

The initial class labels were subjectively assigned by the expert, hence the metrics of the genes that generate the best classifier will tend to reflect the aspects of the source code that the expert may intuitively deem important for distinguishing a class that is well designed and coded. On a production version of this approach, various experts knowledgeable with the problem domain and organization would label the software objects, priming the classifier to use the metrics that a particular organization deems critical in identifying problem software classes. Another area of future research would be to reduce the number of classes. Though a pass/fail labeling may not work so well, as there will tend to be many more pass software objects. A three class labeling scheme may be more appropriate. Using an in-house project does not lead to reproducibility of the method, or the comparison of using alternate methods by other researchers as in-house code is not usually available for distribution. Choosing an open-source project as a dataset would be a workable alternative and may lead to more collaborative research efforts.

Acknowledgement. The Natural Sciences and Engineering Council (NSERC) is gratefully acknowledged for its support of this investigation.

References

[1] Kan S.H.: Metrics and Models in Software Quality Engineering. Addison-Wesley Publishing Company, Reading, Massachusetts, USA (1995)

[2] Fenton N.E., Pfleeger S.L.: Software Metrics: A Rigorous and Practical Approach, 2nd Edition. PWS Publishing Company, Boston, USA (1997)

[3] Pedryc W.: Proc. ACM Software Engineering Knowledge Engineering. Computational Intelligence as an Emerging Paradigm of Software Engineering. Ischia, Italy (2002) 7-14

[4] Harman M., Jones B.F.: ACM SIGSOFT Software Engineering Notes. The SEMINAL Workshop: Reformulating Software Engineering as a Metaheuristic Search Problem, Vol. 26(2001) 62-66

[5] Nikulin A.E., Dolenko B., Bezabeh T., Somorjai R.J.: NMR Biomed. Near-optimal feature selection for feature space reduction: novel preprocessing methods for classifying MR spectra, Vol. 11 (1998) 209-216

[6] Yang, J., Honavar V.: IEEE Intelligent Systems. Feature subset selection using a genetic algorithm, Vol. 13 (1998) 44-49

[7] Raymer M.L., Punch W.F., et al.: IEEE Trans on Evolutionary Computation. Dimensionality reduction using genetic algorithms. Vol. 4 (2000) 164-171

[8] Hochman R., Khoshgoftaar T.M., Allen A.B., Hudepohl J.P.: Proc. 7th IEEE Int. Symposium on Software Reliability Engineering. Using the Genetic Algorithm to Build Optimal Neural Networks for Fault-Prone Module Detection, White Plains, New York (1996) 152-162
[9] Liu Y., Khoshgoftaar. T.M. Proc. 6th IEEE International Symposium on High Assurance Systems Engineering. Genetic Programming Model for Software Quality Classification (2001)
[10] Azar D., Precup D., Bouktif S., Kegl B., Sahraoui H.: Proc. 17th IEEE International Conference on Automated Software Engineering, Combining And Adapting Software Quality Predictive Models by Genetic Algorithms (2002)
[11] Message Passing Interface (MPI) Standard, http://www-unix.mcs.anl.gov/mpi/
[12] Borland's TogetherSoft, http://www.borland.com/together.
[13] Goldberg D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading, Massachusetts, USA (1997)
[14] Duda, C.D., Hart P.E., Stork D.G.: Pattern Classification. Wiley & Sons, New York, USA (2001)
[15] Cantu-Paz E.: Effective and Accurate Parallel Genetic Algorithms. Kluwer Academic Publishers, Boston, USA (2000)
[16] Chidamber SR, Kemerer CF.: IEEE Trans on Software Engineering, .A Metrics Suite for Object Oriented Design, Vol. 20 (1994) 476-493

Evaluation of Different Fitness Functions for the Evolutionary Testing of an Autonomous Parking System

Joachim Wegener[1] and Oliver Bühler[2]

[1]DaimlerChrysler AG, Research and Technology, Alt-Moabit 96 a, D-10559 Berlin, Germany
Joachim.Wegener@DaimlerChrysler.com
[2]STZ Softwaretechnik, Im Gaugenmaier 20, D-73730 Esslingen-Zell, Germany
Oliver.Buehler@stz-softwaretechnik.de

Abstract. The method of evolutionary functional testing allows for the automation of testing by transforming test case design into an optimization problem. To this end it is necessary to define a suitable fitness function. In this paper two different fitness functions are compared for the testing of an autonomous parking system. The autonomous parking system is executed with the test scenarios generated, the fitness for each test scenario is calculated on the basis of an evaluation of the quality of the parking maneuver calculated by the autonomous parking system. A numerical analysis shows, that the proposed area criterion supports a faster convergence of the optimization compared to the proposed distance criterion and that the proposed area criterion describes an efficient method for finding functional errors in the system in an automated way.

1 Introduction

Electronic control units (ECUs) in cars are taking over increasingly complex tasks. New applications such as autonomous parking systems, intelligent cruise control systems, which track the distance to preceding vehicles, or emergency breaking systems rely on the sensor-based calculation of distances to other objects. For such applications, errors in the ECU'S software can result in high costs. Therefore, the aim is to find as many errors as possible by testing the systems before they are released. In practice, dynamic testing is the analytical quality assurance method most commonly used. Usually, a complete test is infeasible because of the huge number of possible input situations. Therefore, test cases have to be selected according to test hypotheses, e.g. each requirement should be tested at least once or every program branch should be executed during the test.

In most cases, test case design is performed manually, requiring a considerable part of the project's resources. The evolutionary functional testing method facilitates the generation of test cases in order to detect functional errors during a directed search. The method transforms the test case design process into an optimization problem. Automated test result evaluation is a prerequisite for this process. The evaluation is carried out by means of the fitness function which assigns a numerical quality value to a test result.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1400–1412, 2004.

This paper evaluates two different approaches to the definition of fitness functions for the functional testing of an autonomous parking system. The fitness functions defined represent a quality metric and can automatically evaluate a parking maneuver calculated by the parking system, i.e. they return a numerical value which describes the quality of the parking maneuver. Both approaches are compared by means of numerical experiments for a prototype implementation of an autonomous parking system. The results show that of both the criteria proposed in this paper, the area criterion can identify critical parking maneuvers better than the distance criterion introduced in [1]. The area criterion provides a more efficient method of error detection in the parking system.

The structure of the paper is as follows: after a brief introduction to evolutionary testing in the second section, the autonomous parking system is introduced and the application of evolutionary testing to its functional testing is explained in the third section. Section 4 describes the different fitness functions, whereas section 5 shows the experimental results achieved by applying these fitness functions. The paper closes with a short summary in the sixth section.

2 Evolutionary Testing

Testing aims to find errors in the system under test and create confidence in its correct behavior by executing the system with selected input situations. Of all the test activities, test case design is assigned decisive importance. Test case design determines the type, scope and thus the quality of the test [2]. If relevant test cases are forgotten, the probability of detecting errors in the system drops. Due to the central importance of test case design, a number of testing methods have been developed to help the tester with the selection of appropriate test data. One important weakness of the testing methods available is that they cannot be automated in a straightforward way. Manual test case design, however, is time-intensive and error-prone. The test quality depends on the performance of the tester. In order to increase the effectiveness and efficiency of the test and thus to reduce the overall development and maintenance costs for systems, a test should be systematic and extensively automatable. Both these objectives are addressed by the evolutionary testing method [3].

In order to transform a test aim into an optimization task a numerical representation of the test aim is necessary, from which a suitable fitness function for the evaluation of the test data generated can be derived. Depending on which test aim is pursued, different fitness functions emerge for test data evaluation. If, for example, the temporal behavior of an application is being tested, the fitness evaluation of the individuals created by evolutionary testing will be based on the execution times measured for the test data [3]. For safety tests, the fitness values are derived from pre- and post-conditions of modules [4], and for robustness tests on fault-tolerance mechanisms, the number of controlled errors can form the starting point for the fitness evaluation [5]. Applications of evolutionary testing to structural testing result in different fitness functions again [6], [7], [8], [9]. An overview of the different applications of evolutionary testing can be found in [10].

If an appropriate fitness function can be defined, the evolutionary test proceeds as follows. The initial population of test data is usually generated at random. Each individual within the population represents a test datum with which the system under test is executed. For each test datum the execution is monitored and the fitness value is determined with respect to the test aim defined. Next, population members are selected with regard to their fitness and subjected to combination and mutation processes to generate new offspring. These are then also evaluated by executing the system under test with the corresponding test data. A new population is formed by combining offspring and parent individuals, according to the survival procedures laid down. From here on, the process repeats itself, starting with selection, until the test objective has been fulfilled or another given stopping condition has been reached.

3 The Evolutionary Testing of the Autonomous Parking System

As an automobile manufacturer, DaimlerChrysler is continuously developing new systems in order to improve vehicle safety, quality, and comfort. Within this context, prototypical vehicle systems have been developed which support autonomous vehicle parking – a function that is likely to be introduced onto the market in some years time. In order to describe the application of evolutionary testing for the functional testing of autonomous parking systems, we shall first provide a brief description of the functionality of an autonomous parking system. We shall then explain the application of the evolutionary test to the parking system. The test environment developed for testing the parking systems and the test data generator are also described.

3.1 The Autonomous Parking System

The autonomous parking systems dealt with in this paper are intended to automate parking lengthways into a parking space, as shown in Fig.1. For this purpose, the vehicle is equipped with environmental sensors which register objects surrounding the vehicle. When driving along, the system can recognize sufficiently large parking spaces and can signal to the driver that a parking space has been found. If the driver decides to park in the parking space detected, the vehicle can do this automatically.

Fig.2 shows the system environment and the internal structure of the autonomous parking system. The inputs are sensor data, containing information on the state of the vehicle, e.g. vehicle speed or steering position, and information from the environmental sensors which register objects on the left and right hand side of the vehicle. As regards output, the system possesses an interface to the vehicle actors, where the vehicle's velocity and steering angle are set.

The parking space detection component processes the data obtained from the environmental sensor systems and delivers the recognized geometry of a parking space if it has been identified as sufficiently large. The parking controller component

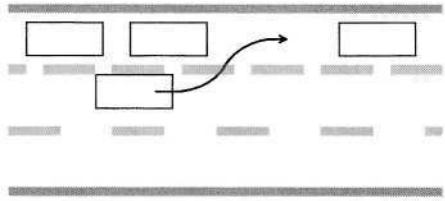


Fig. 1. Functionality of an Autonomous Parking System

uses the geometry data of the parking space together with the data from the vehicle sensors to steer the vehicle through the parking procedure. For this purpose, velocity and steering angle are set for the vehicle actors. The parking controller has to guarantee that the collision area is not entered by the vehicle in order to avoid a high probability of causing damage to adjacent parked vehicles or objects and the vehicle itself.

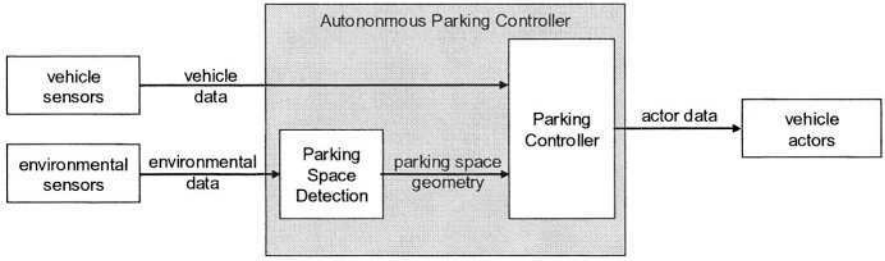


Fig. 2. System Environment and Sub-Components of the Autonomous Parking System

3.2 Applying Evolutionary Testing to the Autonomous Parking System

Comprehensive and efficient testing is essential before releasing a system such as the automatic parking system. As many tests as possible must be performed in a systematic and efficient way. Manual testing of the complete system is cost-intensive and time-consuming because every test case involves setting up a parking scenario with real cars and the manual driving of each maneuver. Furthermore, it is difficult to achieve an exact reproduction of the tests because the details of the test execution vary. In contrast, automated tests have the potential to perform a large number of test cases with less effort in a reproducible manner. Therefore, automated functional tests performed in a controlled simulation environment should be integrated into the quality assurance process of the autonomous parking system.

Evolutionary functional testing provides a way of automating functional tests as a complete process. Instead of selecting the test cases manually, a search for interesting test cases is performed automatically. This is done by translating test case selection into an optimization problem. The possible input situations of the system under test are mapped to the search space. On the one hand, the mapping should keep the size of the search space as small as possible, and on the other hand, the mapping should be

able to produce all possible input data for the system. If one considers the whole input range during the design of the test data generator, it does not mean that all the test cases in this range will actually be tested but it does provide the possibility of generating any test data required. An appropriate model has to be designed for this purpose.

Both components of the autonomous parking system have to be tested thoroughly. In the case of the parking space detection component, we must make sure that suitable parking spaces are identified precisely whereas parking spaces too small for a parking maneuver have to be rejected. Usually, the parking space detection has to be tested in natural environments since reliable simulation models for the environmental sensors are not available. Therefore, it is not considered in the subsequent sections of this paper. Nevertheless, we plan to test the parking space detection in the future by generating environmental data.

For the testing of the parking controller, test cases describing different parking scenarios are generated. The evaluation of the test cases is carried out by the fitness function. In order to test the automatic parking system, the fitness function calculates a numerical fitness value for each parking scenario generated. This fitness value represents the quality of the corresponding test case and aims to lead the evolutionary search into a direction of input situations which result in a parking controller error. Therefore, the fitness function is designed to assign good fitness values to parking scenarios which cause the system to enter the collision area or end up in an inadequate parking situation. Bad fitness values are assigned to scenarios which result in a good parking position with enough distance to the collision area.

3.3 Test Environment

The test environment of the automatic parking system (Fig. 3) comprises the simulation environment, an evolutionary algorithm toolbox, an implementation of the fitness function and the test data generator which translates individuals into actual parking scenarios. The test object is the control unit of the vehicle with the implementation of the automated parking system inside.

The GEA toolbox for Matlab (www.geatbx.com) was used to implement the evolutionary algorithms used to test the autonomous parking system. The simulation environment (built up on a Matlab R12.1 platform) simulates the properties of the vehicle as well as the surrounding environment. It runs with the “in-the-loop” control unit, meaning that the simulation environment calculates the sensor data for the vehicle and presents it to the parking controller inside the control unit. The control unit processes this sensor data and reacts to it with control data for the simulation environment. This loop simulates a complete parking maneuver for the parking space scenario generated. The parameters describing a parking space scenario for the simulation, such as the position of the car and the size of the parking space, are outputs of the test data generator (see 3.4.). After the simulation of a parking maneuver the fitness value is calculated, using the fitness functions described in the subsequent section, and is then assigned to the individual generated.

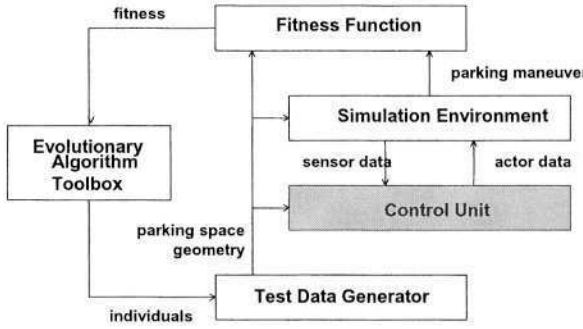


Fig. 3. Design of the Test Environment

3.4 Test Data Generator

The geometric data which characterizes a parking space comprises six points P0 to P5, and is referred to as parking space geometry. The points define the border between the drivable and impassable area of the parking situation. The model for the generation of this parking space geometry is shown in Fig.4. It is a simplified model because the borders of the parking space are always rectangular. The shape of the parking space can only vary in length and depth.

This model takes the values of five independent variables and calculates the parking space geometry from them. The independent variables define *length* and *width* of the parking space. In addition, the starting position with the *distance* of the car to the parking space *(dist2space),* the angle *psi* and the *gap* between the vehicle and the collision area on the right hand side of the car are part of the parking space scenarios generated.

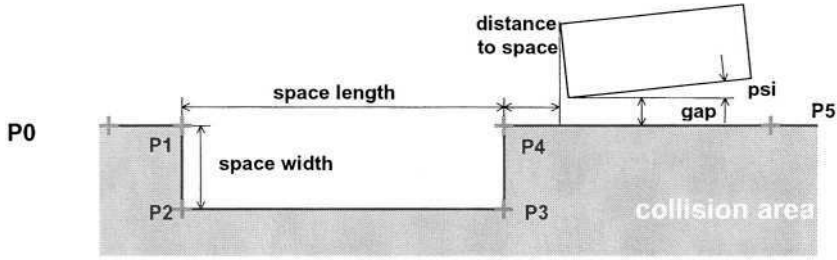


Fig. 4. Model for the Generation of Parking Space Geometry

4 Definition of Fitness Functions

This section describes the definition of two different strategies for the evaluation of the fitness of a parking scenario. One strategy uses the distance between the vehicle and the collision area as a measure of the evaluation of fitness [1], the other strategy

works with the area between the vehicle and the collision area. Both strategies separate the parking space into two parts: (1) collision with the preceding vehicle and (2) collision at the parking side. While the vehicle is pulling into the parking space its freedom of movement is limited and thus, if a collision with the preceding vehicle or the parking side occurs, either the right rear edge or the right front edge of the car will be involved. Depending on the position of the collision areas the lines between P3-P4 and P5-P4 have to be observed in order to identify a collision with the front vehicle. To identify a collision at the side, the line P2-P3 has to be observed. The definition distinguishes between the observation of a corner, defined by three points, and the observation of an edge, defined by two points.

The fitness function is intended to assess a parking scenario and to assign an adequate fitness value to it. The fitness value should correspond to the quality of the parking scenario. From the testing perspective it is a good parking scenario if a collision with other parked cars occurs or the controlled vehicle touches the parking side. Since we are minimizing the fitness values during the search process, the fitness value assigned to a parking scenario decreases with the quality of the scenario. An interesting parking scenario achieves a smaller value than a parking scenario for which the automatic parking system performs well. The fitness becomes negative when a collision occurs.

4.1 The Distance Criterion Fitness Function

The distance criterion considers the closest distance between a vehicle edge and the collision border during the parking maneuver. In separate evaluations the smallest distance of the collision corner P3-P4-P5 and the smallest distance from the collision side P2-P3 are calculated. The following subsections describe, how the evaluation of the collision corner and the collision side is carried out.

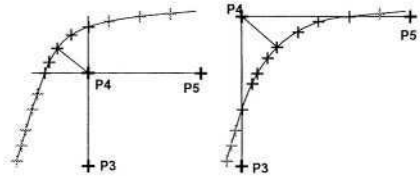


Fig. 5. Selection of Smallest Distance

Evaluation of a Collision Corner

The collision corner is defined by three points P3-P4-P5. All distances are calculated as polar coordinates with P4 as their origin (Fig. 5). The evaluation of the collision corner observes the section defined by P3-P4-P5 and the section diagonally opposite. Only the points within these sections are considered. As a quality measure for the evaluation of the parking maneuver, the smallest distance between the vehicle path positions and point P4 is taken. The value of the distance is positive, if the path is

outside the collision corner. The value is set to zero, if the path crosses P4. The value is measured as a negative value, if the path runs through the collision corner (Fig. 6).

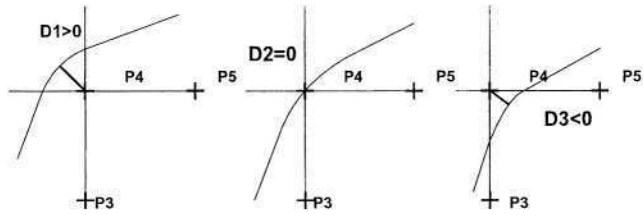


Fig. 6. Signed Distance Values

This strategy aims to ensure, that the closer the path gets to the collision corner, the lower the assigned fitness value becomes. For different paths which continuously move into the collision corner, the corresponding fitness values become continuously lower, as shown in Fig.7, where D1 > D2 > D3.

Evaluation of a Collision Side

The collision side is defined by the straight line between P2-P3. The distances calculated are between this line and the path positions of the car. In this calculation, only path points whose x-values are within the range of P2 and P3 are taken into account. The selection is carried out by comparing the x-coordinates of the path points with P2 and P3.

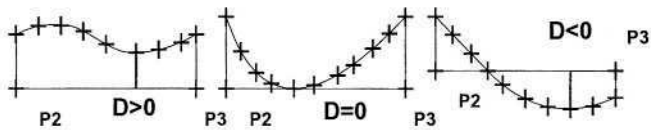


Fig. 7. Distance from Line with Positive, Zero and Negative Value

The distance is calculated positive, when the path is above the line, and is set to zero, when the path touches the line. The distance is calculated as a negative value, when the path runs through the collision area. From all the distance values calculated, the minimum is taken as the fitness value for the parking space scenario generated.

4.2 The Area Criterion Fitness Function

The area criterion fitness function considers the area between the path of the vehicle and the parking geometry. Here, the areas included in the collision corner and the collision side are calculated in separate evaluations. The following subsections describe how the evaluation of the collision corner and the collision side is carried out with the area criterion.

Evaluation of a Collision Corner

The area included between the corner lines and the vehicle path is taken as a measure of the evaluation of the parking scenario. In order to calculate this area, it is separated into smaller segments, appropriate to the points of the vehicle's path through the corner, as shown in Fig.8. The overall area A included in the collision corner is the sum of all the segments together.

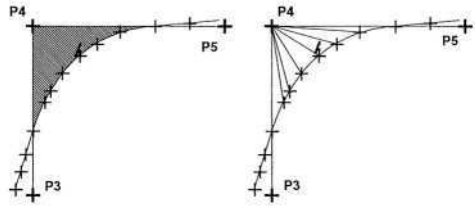


Fig. 8. Separation of Included Area into Small Segments

For an effective and fast calculation of the fitness value, an approximate description of the area of each segment can be provided by a triangle. When the distances between two points of the path T_n and T_{n+1} is significantly smaller than the distance to P4 the angle $alpha_n$ is very small.

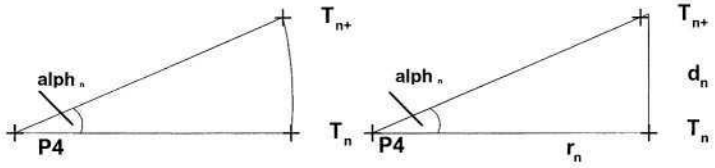


Fig. 9. Approximation of Segment by a Triangle

For small angles of $alpha_n$, the ratio d_n to r_n is approximately the $angle_n$ in radians. The area of one segment which is approximately described by a triangle is:

$$A_n = \frac{1}{2} \times r_n \times d_n \text{ and with } d_n \approx r_n \times (th_{n+1} - th_n) \tag{1}$$

the area of a segment can be calculated thus:

$$A_n \approx \frac{1}{2} \times r_n^{\,2} \times (th_{n+1} - th_n). \tag{2}$$

The angle th_n and the radius r_n for each path point T_n can be easily obtained, by transferring the path positions from Cartesian coordinates into polar coordinates with P4 as the origin of the coordinate system. The overall area of the corner is the sum of all the segments within the corner

$$A_{corner} = \sum A_n \ . \tag{3}$$

To lead the optimization towards a collision, the corner P3-P4-P5 symmetrical to point P4 is also taken into consideration. The path of the vehicle has to pass the point P4 and the aim of the optimization is to bring that path into the collision area. The idea is to rate an area included in the opposite corner as a positive value and an area included in the collision comer as a negative value. When the vehicle path crosses through the corner point P4 the corresponding value is set to zero. Thereby, the areas shown in Fig. 10 become A1 > A2 > A3 when the vehicle path keeps on shifting into the collision corner. This leads to a gradual improvement of the fitness value, depending on how near the vehicle path passes by the collision corner or crosses into it.

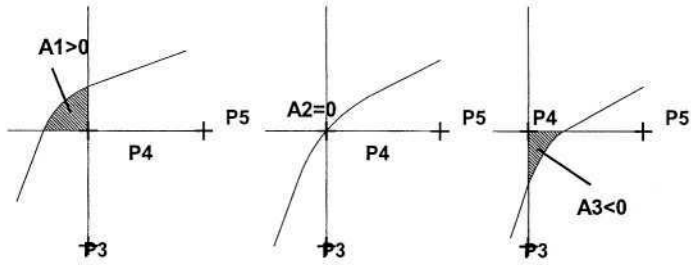


Fig. 10. Signed Area Values

Evaluation of a Collision Side

The evaluation of a collision side takes the area included between the vehicle path and the straight line P2-P3 as a measure. The calculation of the area takes only those points into consideration which have their x-coordinate in the range between P2 and P3. The calculation is carried out using an approximation, with the rectangles defined by the path positions. The area of each rectangle can be easily calculated by Δx and the distance between path and line Δy. With a sufficient number of path points in the range between P2 and P3, a good approximation of the area included can be achieved.

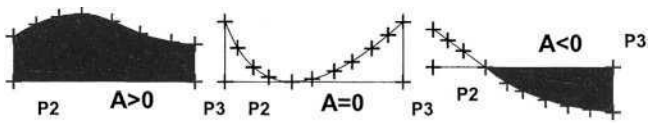


Fig. 11. Area with Positive, Zero and Negative Value

Fig.11. (1) shows when the path is above the line, (2) the path touches the line or (3) the path crosses the straight line into the collision area. In the first case, the area included has a positive value, in the second case the value for the area is set to zero. In the third case the area below the line has a negative value. The distinction drawn between the three cases prevents a larger area above the line from compensating for small areas below the line. It also prevents a situation when the line is touched from being concealed by adjacent areas so that it cannot be observed.

5 Experiments

The aim of the experiments is to analyze and compare the suitability of the fitness functions described in the previous section for the testing of autonomous parking systems. A number of experiments were performed to analyze the fitness landscapes for both fitness functions. For each experiment, two variables from the test data generation input vector were varied within a defined range and a defined number of samples. The remaining three variables were kept constant resulting in two-dimensional variation plots of the fitness landscapes. In order to calculate the fitness value for each parking scenario, a parking maneuver carried out by the autonomous parking system was simulated as described in section 3.2.

Fig. 12 shows the results for the experiment in which the variables *dist2space* and *psi* were varied, and in which the *length* and *width* of the parking space as well as the *gap* to the right of the vehicle are kept constant: *length* was set to 8.0 m, *width* to 2.5 m and the *gap* to 0.7 m. The axis to the right shows the *distance* to the parking space in the range of 0.0 m to 7.0 m, with a resolution of 70 points. The axis at the bottom shows the angle *psi* from +10° to –10°, with a resolution of 40 points. Both functions return negative fitness values when angle *psi* reaches +10° and *dist2space* goes towards 7 m. A considerable difference between the shapes of both surfaces is that for fitness values greater than zero the distance criterion returns constant small values where the area criterion lowers its values towards the border to zero. This appears as a flat plateau in the distance criterion surface (on the right hand side of Fig. 12) where the area criterion surface slopes continuously (left hand side of Fig. 12).

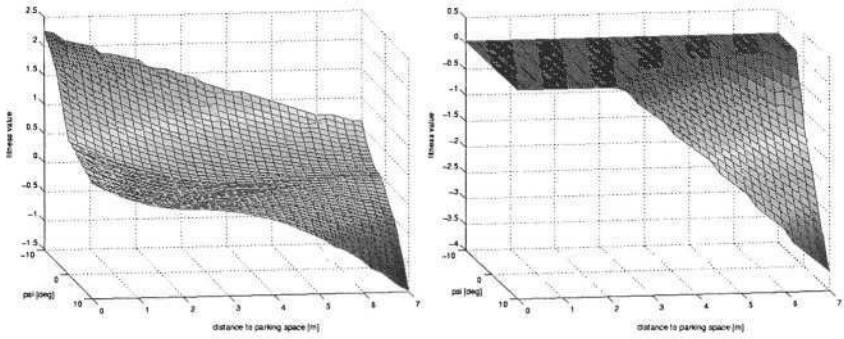


Fig. 12. Fitness Landscapes for Area Criterion (left) and Distance Criterion (right) as a Function of Angle psi and Distance to Parking Space

Comparable results were found in most experiments, e.g. Fig. 13 shows the plots for the variation of *dist2space* and parking space *length.* The range of the *length*, shown on the axis to the right, is between 5 and 10 m. *Dist2space* is shown on the axis to the left and ranges from 0 to 7 m. The remaining variables are kept constant: *width* of parking space was set to 2.5 m, the *gap* to the right of the vehicle was set to 0.7 m and the angle *psi* was set to 0°. As in the preceding example, the surfaces for

the area criterion and the distance criterion differ. The distance criterion surface has a flat plateau of equal fitness values for small distances to the parking space and longer parking spaces. In contrast, the area criterion shows a sloping characteristic for that domain.

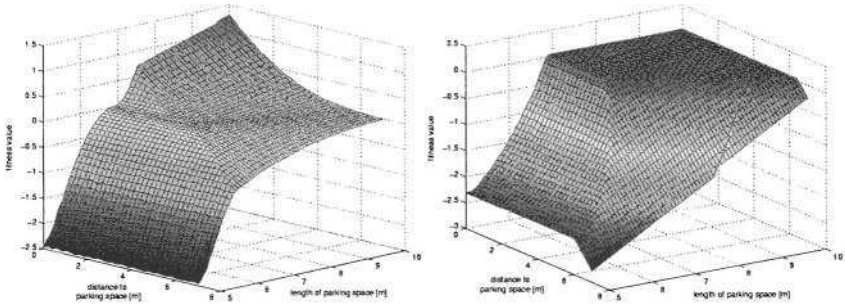


Fig. 13. Fitness Landscapes for Area Criterion (left) and Distance Criterion (right) as a Function of Distance to and Length of the Parking Space

6 Conclusion

The experiments have shown that both fitness functions have a sloping characteristic for collision maneuvers, where their fitness value is less than zero. The slope of the distance criterion function is steeper in that domain than that of the area criterion function. Furthermore, the distance criterion function shows an angle at the point where the values enter the negative range whereas the area criterion function has a smooth transition for fitness values around zero. Both fitness functions guide the search towards scenarios in which a collision occurs. The area criterion function provides a sloping characteristic for maneuvers with positive fitness values, in contrast to the distance criterion function, which returns constant fitness values resulting in plateaus of equal fitness in the search space for scenarios without a clash. It does not differentiate between good and better maneuvers. As a consequence, the area criterion function is better suited to directing the search towards collision maneuvers than the distance criterion function. It helps us to find test cases more quickly for which the autonomous parking system does not react correctly. Therefore, the area criterion will be used for the evolutionary testing of autonomous parking systems in the future.

Acknowledgments. The work described was carried out as part of the SysTest project., funded by the EC under the 5th framework programme (GROWTH, project ref. G1RD-CT-2002-00683).

References

1. Buehler, O., Wegener, J.: Evolutionary Functional Testing of an Automated Parking System. Proceedings of the International Conference on Computer, Communication and Control Technologies (CCCT '03) and the 9th. International Conference on Information Systems Analysis and Synthesis (ISAS '03), Florida, USA (2003).
2. Grochtmann, M., Grimm, K.: Classification-Trees for Partition Testing. Software Testing, Verification & Reliability, vol. 3, no. 2, pp. 63-82 (1993).
3. Wegener, J., Grochtmann, M.: Verifying Timing Constraints of Real-Time Systems by Means of Evolutionary Testing. Real-Time Systems, vol. 15, no. 3, pp. 275-298 (1998).
4. Tracey, N., Clark, J., Mander, K.: The Way Forward for Unifying Dynamic Test Case Generation: The Optimisation-Based Approach. Proceedings of the IFIP International Workshop on Dependable Computing and Its Applications, South Africa, pp. 169-180 (1998).
5. Schultz, A., Grefenstette, J., Jong, K.: Test and Evaluation by Genetic Algorithms. IEEE Expert, vol. 8, no. 5, pp. 9-14 (1993).
6. Jones, B., Sthamer, H., Eyres, D.: Automatic Structural Testing Using Genetic Algorithms. Software Engineering Journal, vol. 11, no. 5, pp. 299-306 (1996).
7. Pargas, R., Harrold, M., Peck, R.: Test-Data Generation Using Genetic Algorithms. Software Testing, Verification & Reliability, vol. 9, no. 4, pp. 263-282 (1999).
8. Michael, C., McGraw, G., Schatz, M.: Generating Software Test Data by Evolution. IEEE Transactions on Software Engineering, vol. 27, no. 12, pp. 1085-1110 (2001).
9. Tracey, N., Clark, J., Mander, K., McDermid, J.: An Automated Framework for Structural Test-Data Generation. Proceedings of the 13th IEEE Conference on Automated Software Engineering, Hawaii, USA (1998).
10. McMinn, P.: Search-based Software Test Data Generation: A Survey. To appear in Software Testing, Verification & Reliability & Reliability (2004).

Search Based Automatic Test-Data Generation at an Architectural Level

Yuan Zhan and John Clark

Department of Computer Science
University of York
YorkYO10 5DD, UK
{yuan, jac}@cs.york.ac.uk

Abstract. The need for effective testing techniques for architectural level descriptions is widely recognised. However, due to the variety of domain-specific architectural description languages, there remains a lack of practical techniques in many application domains. We present a simulation-based testing framework that applies optimisation-based search to achieve high-performance testing for a type of architectural model. The search based automatic test-data generation technique forms the core of the framework. *Matlab/Simulink* is popularly used in embedded systems engineering as an architectural-level design notation. Our prototype framework is built on *Matlab* for testing *Simulink* models. The technology involved should apply to the other architectural notations provided that the notation supports execution or simulation.

1 Automatic Testing at the Architecture Level

Software testing is an expensive procedure. It typically consumes more than 50% of the total development budget [1]. Failure to detect errors can result in significant financial loss or even disaster in the case of safety critical systems. Complete testing is impossible due to the huge input spaces involved. It is desirable, therefore, to seek techniques that will achieve testing rigour (i.e. be effective) at an acceptable cost (i.e. be efficient).

Test-data generation is one of the most tedious tasks in the software testing process. As system size grows, manual test-data generation places a great strain on resources (both mental resources and budget). This problem becomes especially serious when developers want to achieve sufficient confidence in system rigour. Automated test-data generation is one way forward to solve this problem and to increase testing efficiency. Automation lies at the heart of our proposed research.

The modern aim of 'testing' is to discover faults at the earliest possible stage because the cost of fixing an error increases with the time between its introduction and detection. Thus high-level models have become the focus of much modern-day verification effort and research. *Matlab/Simulink* is a widely used notation in dynamic systems development industry that allows models to be created and exercised. *Mat-*

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1413–1424, 2004.

lab/Simulink models can be architectural level designs of software systems. The simulation facilities allow such models to be executed and observed. This property of *Simulink* turns out to be an advantage for effective dynamic testing. We are aware that *Matlab* itself provides a '*Simulink Performance Tool Set*', which aids auto testing. However it's functionality is restricted to only measuring test completeness. In this work, we focus on automatically generating effective test-data for testing *Matlab/Simulink* models. Other authors have recognized the practical significance of such modeling and the need to provide assurance information automatically, e.g. the worst case execution times for such models [11].

An *adequacy criterion* is a criterion that defines what constitutes an *adequate* test-set [5]; it provides a measure of how effective a given test-set is. The ability to compare test-sets allows the tester to identify how to add tests to an existing set to improve effectiveness of the overall set. (Additional tests should lead to an improvement in the measure of effectiveness.) [5] introduced different types of test adequacy criteria; these can be generally categorized as: structural-based, fault-based and error-based. Some types of adequacy criteria are more suitable than others on particular problems. Generally, different adequacy criteria are complementary and are often combined in practice.

Code level coverage criteria are explained in [5]. However, these can be adapted to specification and architecture level testing too. Our work is concerned with interpretations of widely used structural coverage criteria. We implemented an automatic test-data generation tool to cover particular *paths* of *Simulink* models. Combined with random test generation, this tool enables efficient automation of structural coverage test-data generation. The construction of the structural coverage test generation tool is detailed in the next section.

2 Search Based Automatic Test-Data Generation

Test data generation has been a very successful branch of search based software engineering. Most work however, has been at the code level (e.g. [2,6,8,9, 11,12,13,14]). The reader is referred to the authoritative survey [18] for a thorough overview of the field. Below we explain how we implemented the automatic structural coverage test-data generation tool for testing *Simulink* models. First, we give some background on *Simulink* and then describe our strategies for applying the test-data generation technique.

Simulink Introduction. *Simulink* [1] is a software package for modelling, simulating, and analysing system-level designs of dynamic systems. *Simulink* models/systems are made up of blocks connected by lines. Each block implements some function on its inputs and outputs the results. Outputs of blocks form inputs to other blocks (represented by lines joining the relevant input/output ports). Models can be hierarchical.

[1] Developed by the MathWorks Inc: http://www.mathworks.com.

Each block can be a subsystem comprising other blocks and lines. Fig. 1 is a simple *Simulink* model.

Simulink models have their special way of forming branches compared to programs. Basically *Simulink* uses the 'Switch' block or its derivatives, like the 'Multiport Switch' block, to form branches. A 'Switch' block has three 'in' ports, one 'out' port and there is a threshold value associated with the block. When the value of the second 'in' port is greater than or equal to the threshold parameter, the output will equal to the value carried on the first 'in' port, otherwise the value carried on the third 'in' port will be channelled through to the output. Therefore a 'Switch' block can map to an 'if ... then ... else' branching structure in code.

Simulink models execute (calculate the outputs of) all branches of the models, whether the branches are selected or not, while for programs, only the selected branches are executed. For example, the following code matches the model in Fig. 1. In the *Simulink* model, both 'x–y' and 'y–x' are calculated although only one of these results is channelled through to the output by the 'Switch' block. However, in the code, only one of them will be executed depending on the evaluation of the predicate (x>=y).

```
program calculation;
input x,y;
output z;
begin
  if x>=y
    z = x-y;
  else
    z = y-x;
end;
```

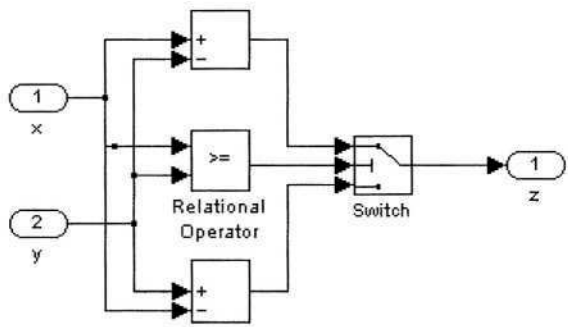


Fig. 1. An example of a *Simulink* model

Simulink is generally used for designing embedded systems – of which a significant feature is that they maintain state. The systems have continuous inputs and outputs and the execution step is controlled by some timer trigger, e.g. a step size can be

1 millisecond. Therefore, for the model in Fig. 1, the input to the system over n time steps should be a sequence $< (x_1, y_1), (x_2, y_2), \ldots (x_n, y_n) >$, and the corresponding output should also be a sequence $< z_1, z_2, \ldots z_n >$.

Interpreting a Test-Data Generation Problem as a Search Problem. In the prototype tool implementation we consider only models whose branching blocks are 'Switch' blocks. A requirement for the generation of a particular structural coverage test input comprises specifying a subset of all 'Switch' blocks involved together with the required condition values (satisfied or unsatisfied). We can consider such a requirement as the equivalent of a 'sub-path' coverage requirement in programs. A single test-data generation requirement for the model in Fig. 2[2] might be: Switch2 = satisfied, Switch3 = unsatisfied (which means the outcome of block 'Product1' is channelled through 'Switch2', and the outcome of block 'Switch2' is channelled through 'Switch3' to the final output). Some combinations of 'Switch' conditions may be over-restrictive, e.g. in Fig. 2, if we require that 'Switch3' predicate is to be satisfied, which means the first (top) input of it is put through to the output, it is over-restrictive to specify weather block 'Switch2' is to be satisfied or not because the outcome of 'Switch2' would not affect the model outcome anyway. Over-restrictive combinations may also be infeasible.

Fulfilment of structural adequacy criteria will require a test-set to exercise identified combinations of 'Switch' predicates. We may impose a simple 'branch coverage' criterion (each branch of a 'Switch' must be exercised by at least one test input vector) through to 'exhaustive coverage' of each possible combination of 'Switch' predicates (we shall term this *all-paths-coverage*).

The automatic test-data generation for satisfying each path coverage requirement is fairly straightforward. Firstly, we need to locate the 'Switch' blocks listed by the coverage requirement in the model and insert probes into the second input signal/line of those 'Switch' blocks. (The purpose of inserting probes is to view the runtime values of those points and use the information collected to direct moves of the test-data search. Therefore the probes are inserted by connecting the signal to an 'Outport' block for observation.) The second step is to design the cost-function to evaluate the quality of an input test-datum according to three types of information: path requirement (satisfiability of 'Switch' blocks), threshold parameter value for each 'Switch' block, and values observed by probes. Detailed cost function construction will be described in the next sub-section. Then we need to apply dynamic search procedure to search for desired test-data. The search will be based on the simulation of models with candidate test-data as inputs. Each simulation provides information about how good the current candidate test-datum is. The usage of the optimisation search techniques will be detailed in sub-section after the next.

[2] The 'Threshold' parameter of all three 'Switch' blocks in the figure has value '0'.

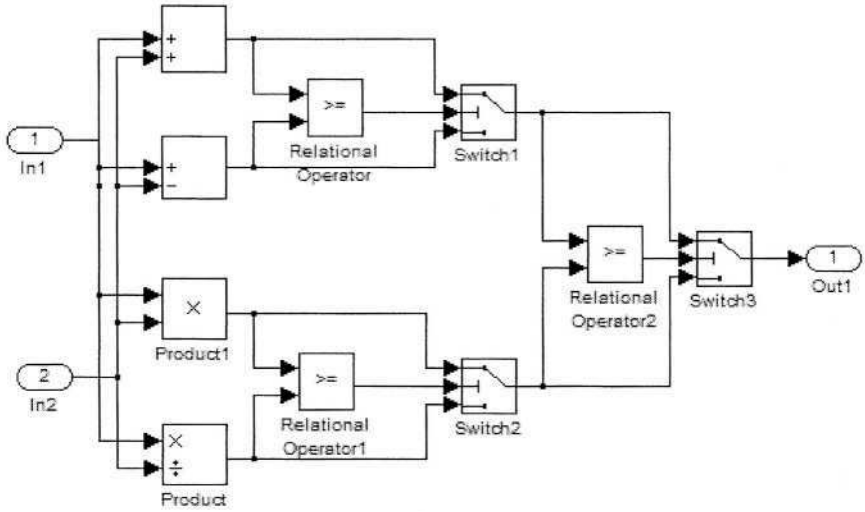


Fig. 2. *Simulink* model branching structure

Cost Function Design. We wish to guide the search towards test-data that causes identified 'Switch' block branches to be taken. With each 'Switch' block we call the 'Threshold' parameter 'para'. If the run-time value 'Vp' of the second input port (whose value our probe monitors) of the 'Switch' block satifies '**Vp ≥ para**' then input port 1 is selected for output. If 'Vp < para' then input port 3 is selected. For any such identified condition, we can associate a cost indicating how far the current data is from satsifying the condition. Thus, if we require '**Vp≥20**', say, then a 'Vp' value of 0 should have greater cost than a 'Vp' value of 19, since the latter 'nearly' causes the required predicate to be true, and the former clearly does not. The cost function encoding scheme we apply for such relational predicates is illustrated in Table 1. Similar approaches have been used by Korel [9], Tracey et al. [6], Wegener et al. [11], Jones et al. [8] etc. But cost function encoding for logical operations also needs to be defined because when combining all the branching requirements together, we need to calculate the cost of a *conjunction* of various relational predicates. In general (with models that maintain state) a test-datum will comprise a *sequence* of consecutive test inputs $<TI_1,\ldots, TI_k>$ over k time steps. We will need to evaluate this sequence based on the degree of achievement of goals at each step. We need our goal (predicate) to be met at *any* step and so need to evaluate the cost of a *disjunction* of predicates. Bottaci [2] suggested a set of cost function encodings for logical operations that can be more accurate in reflecting the fitness of test-data compared to that of other researchers. We adapt his idea for our application as shown in Table 1.

Table 1. Cost function encoding method

Predicate	Value of Cost Function F
Boolean	if TRUE then 0, else *maxcost*
$E_1 < E_2$	if $E_1 - E_2 < 0$ then 0, else $E_1 - E_2 + \delta$
$E_1 \leq E_2$	if $E_1 - E_2 \leq 0$ then 0, else $E_1 - E_2$
$E_1 > E_2$	if $E_2 - E_1 < 0$ then 0, else $E_2 - E_1 + \delta$
$E_1 \geq E_2$	if $E_2 - E_1 \leq 0$ then 0, else $E_2 - E_1$
$E_1 = E_2$	if $Abs(E_1 - E_2) = 0$ then 0, else $Abs(E_1 - E_2)$
$E_1 \neq E_2$	if $Abs(E_1 - E_2) \neq 0$ then 0, else K
$E_1 \vee E_2$ (E_1 unsatisfied, E_2 unsatisfied)	(cost (E_1) × cost (E_2))/(cost (E_1) + cost (E_2))
$E_1 \vee E_2$ (E_1 unsatisfied, E_2 satisfied)	0
$E_1 \vee E_2$ (E_1 satisfied, E_2 unsatisfied)	0
$E_1 \vee E_2$ (E_1 satisfied, E_2 satisfied)	0
$E_1 \wedge E_2$ (E_1 unsatisfied, E_2 unsatisfied)	cost (E_1) + cost (E_2)
$E_1 \wedge E_2$ (E_1 unsatisfied, E_2 satisfied)	cost (E_1)
$E_1 \wedge E_2$ (E_1 satisfied, E_2 unsatisfied)	cost (E_2)
$E_1 \wedge E_2$ (E_1 satisfied, E_2 satisfied)	0

Here is an example of using the above encoding scheme (see the model in Figure 3):

Assume that:

The testing requirements are: Switch**1** to be unsatisfied, Switch**2** to be satisfied;

Threshold parameters are: Switch**1**para = 100, Switch**2**para = 50;

Input: step1: In1 = 5, In2 = 10; step2: In1 = 10, In2 = 100; step3: In1 = (-10), In2 = 50.

Therefore, the probes observed for the three steps should be:

Step1: Switch**1**probe = 15, Switch**2**probe = 15;

Step2: Switch**1**probe = 110, Switch**2**probe = (-90);

Step3: Switch**1**probe = 40, Switch**2**probe = 40.

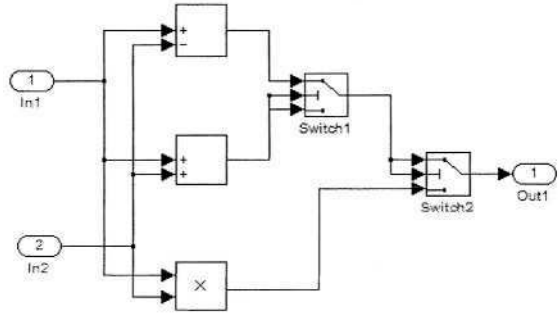


Fig. 3. *Simulink* model for cost function encoding scheme demonstration

The total cost of this test case will be:

cost(((15<100)∧(15>=50)) ∨ ((110<100)∧(-90>=50)) ∨ ((40<100)∧(40>=50)))

C_1=cost((15<100)∧(15>=50))=35;

C_2=cost((110<100)∧(-90>=50))=(10+140)=150;

C_3=cost ((40<100)∧(40>=50))=10;

Cost=(C_1 C_2 C_3)/(C_1 C_2+ C_1 C_3+ C_2 C_3)=7.394.

We have carried out multi-step evaluation for illustration only. (The system obviously has no feedback within it, so a sequence of length 1 would be entirely appropriate in practice). As a result of such cost function encoding scheme, the test-data generation problem can be interpreted as a search for a test-datum that can minimize the underlying cost function. The target is zero. Next we provide a brief outline of the optimization technique we used – simulated annealing, and its application details.

Optimization Techniques. In this framework, we have used the well-established technique of simulated annealing [7] to search for the desired test-data. Simulated annealing is a global optimization heuristic that is based on the *local* descent search strategy. The annealing algorithm we apply is shown below.

Select an initial solution $testData_0$;
Select an initial temperature $t_0>0$;
Select a temperature reduction function α(=0.9 here);
Repeat
  Repeat
    Generate a move $testData \in N(testData_0)$;
    $\delta = f(testData) - f(testData_0)$;
    If $\delta < 0$
    Then $testData_0 = testData$;
    Else
    Generate random x uniformly in the range (0, 1);
      If $x < \exp(-\delta/t)$ then $testData_0 = testData$;
  Until *innerLpCount* = *maxInnerLpNo* or $f(testData_0)$satisfies the requirement;
  Set $t = \alpha(t)$;
Until *outerLpCount* = *maxOuterLpNo* or *nonAcceptCount* = *maxNonAcceptNo* or $f(testData_0)$ satisfies the requirement.
$testData_0$ is the desired test-data if $f(testData_0)$ satisfies the requirement.

The initial solution is usually randomly generated. Then the search keeps generating, considering and possibly moving to local neighborhood solutions of the current solution. A move is accepted if it improves the evaluation of the cost function. A worsening move may also be accepted probabilistically in a way that depends on the temperature t in the search. The higher the temperature is, the easier a worsening solution can be accepted. Initially the temperature is high and a lot of worsening solutions may be accepted. As the time passes by, the temperature drops and eventually it 'freezes' and therefore no worsening solutions can be accepted. A number of moves are considered at each temperature. If no move has been accepted for some time then

the search halts. For a problem that does not require reaching global optima, the search procedure halts at any time when a satisfactory solution is found.
Interested readers are referred to [15], [7] and [10] for more details about the annealing algorithm. In our application a move effectively perturbs the value of one of the inputs in the current test sequence by a value less than or equal to 1 percent of the range of the input. We applied a geometric cooling rate of 0.9. The number of attempted moves at each temperature was 500, with a maximum of 100 iterations (temperature reductions) and a maximum number of 30 consecutive unproductive iterations (i.e. with no move being accepted). These parameters may be thought to be on the 'small' side, but the computational expense of simulation requires us to make pragmatic choices.

3 Automating Structural Coverage Test Generation

In terms of structural coverage, we evaluate test-sets by assessing the percentage of paths being covered by the tests. As has been explained in section 2, in *Simulink*, the branches are typically caused by 'Switch' blocks. (We exclude the usage of the other branching blocks in this prototype framework.) Therefore the *all-paths-coverage* can be defined as having all *combinations* of the 'Switch' block satisfaction conditions being covered. For example, in Fig., there are altogether four *full* paths[3]. They are:

1) 'Switch1'=satisfied and 'Switch2'=satisfied;
2) 'Switch1'=satisfied and 'Switch2'=unsatisfied;
3) 'Switch1'=unsatisfied and 'Switch2'=satisfied;
4) 'Switch1'=unsatisfied and 'Switch2'=unsatisfied.

A test-set having 10 test cases but covering only path 1) and 3) would be evaluated as having 50% path coverage. By this means, we can evaluate the structural coverage capability of test-sets by running all the test cases within the underlying test-set against the model under test and recording the paths being covered by those test cases. The more paths can be covered, the better the test-set is.

There are usually multiple test cases executing the same path in a random test-set. For the sake of efficiency, we may want to remove the test cases that can not increase the structural coverage of the test-set. For our *all-paths-coverage* testing, each test case covers one and only one full path. So the redundant test case removal is straightforward. For less stringent sub-path coverage requirements (e.g. the equivalent of 'branch coverage') test-set reduction may be more sophisticated.

To efficiently automate the structural coverage test data generation, we propose to combine the random test-data generation and our targeted test-data generation together, which means we generate a moderate sized random test-set and check the coverage capability of it first, then use our automatic targeted test-data generation tool to generate test-data that can cover those paths which were not covered by the

[3] A *full path* specifies the branching preference of all 'Switch' blocks, while a *sub-path* specifies the branching preference of only a subset of the 'Switch' blocks in the model. A sub-path coverage test requirement is less stringent than a full path coverage requirement.

initial random set. We use random testing because it generally can achieve a certain amount of coverage at a very low cost (cheaper than applying the optimization based search technique). For those paths that are difficult to be covered by the random test-set, we use our instrumented test-data generation tool, expecting to find the desired test-data quicker than random search.

Since our test-data generation tool is geared towards the 'hard'[4] targeted testing aims, To demonstrate its effectiveness and efficiency, we compare it with random test-data generation in both the coverage capability and the number of test cases tried during the searching of test-data.

In the experiment, we automatically generate a random test-set of the size[5] that doubles the path number first. E.g. for a model that has 3 'Switch' blocks, and therefore has 8 full paths, we generate a random test set of the size of 16. Then we mark the paths that the random set can cover. For those paths that are not covered by the random test-set under evaluation, we use our automatic test-data generation tool to generate test-data to cover them and increase the structural coverage of our test-set.

Therefore we compare both the coverage capability and the number of test cases tried during the searching of test-data. The comparison results are recorded in Table 2. We tried both approaches (random test generation and simulated annealing search based test generation) on 4 models: 'SmplSw', 'Quadratic', 'RandMdl', and 'Combine'. All the models used are hand-crafted and designed for providing hardness in generating test-data for covering some paths. In the table, for model 'Quadratic', we generate a random test set with 16 test cases, and it covered 3 paths. For the remaining 5 paths, we use both simulated annealing approach and random approach to generate test data that can cover them, attempting each path execution aim in turn. Simulated annealing on its own used 1,641 cases and random approach tried 25,377 cases. Both approaches reached a full coverage eventually. Therefore the total test case number used by each of them are 1,657 and 25,393 respectively (16 cases from the initial random set).

For each path execution aim up to 50,000 tests were allowed (both for annealing and for random generation). Therefore there are some paths that cannot be reached within our effort allowance for model 'RandMdl' and 'Combine'.

Our observation is that the first model 'SmplSw' is rather straightforward for locating test-data. All paths can be covered by the initial small random test set of 8 test cases. And there was no need to use the our instrumented test generation tool. However the next three models present greater difficulty. Our instrumented test-data generation approach achieved greater coverage with fewer executions. We noticed that for the 'Combine' model, there are a couple of paths that failed to be covered by our search-based test-data generation in the batch run. For these two paths, we tried to run our search-based test-data generation once again and found out that both desired

[4] By 'hard' we mean those testing aims that are difficult to be covered by a random test-set.

[5] The optimal size of the initial random test-set varies from model to model. Research needs to be done to investigate how to set this size.

Table 2. Case study result for the automatic test-data generation tool

Model Name	Model Size	'Switch' Block No.	SimAnneal Case No	Random Case No	SimAnneal Coverage	Random Coverage
SmplSw	8 blocks	2	8	8	4/4	4/4
Quadratic	15 blocks	3	1,657	25,393	8/8	8/8
RandMdl	14 blocks	4	38,161	347,605	16/16	10/16
Combine	29 blocks	7	1,062,993	3,907,080	126/128	52/128

test-data could be found by this technique within the lengths we allowed to try. The simulated annealing approach may sometimes result in convergence of a local optimum. In our test-data search, it may result in not being able to provide a satisfactory solution. To overcome this problem, the approach of repeating the algorithm using several different starting solutions is suggested.

4 Conclusions and Future Work

The basic aim of this work is to facilitate test automation at the architectural level. We have adopted a two-pronged attack strategy. The major tool is the automated test generation facility. This has been applied to generate the architectural equivalent of structural coverage tests and is entirely automatic. We believe that this can easily be extended to provide architectural analogues of the various code-level test applications (e.g. the safety analysis, exception generation and falsification testing of Tracey et al. [12,13,14]). For example, to carry out safety analysis, safety invariant checkers are inserted into the model under test as probes. Therefore for different test inputs, we may observe from the safety invariant checkers how close the test-data comes to breaking the safety invariant. Such information can direct our optimisation heuristic search for the test-data. This method can be used at the early stage of system safety analysis; it may show cheaply (because it is fully automated) that some safety property does not hold. However to *show rigorously* that some safety property *holds*, we will have to rely on formal methods or other rigorous techniques.

The second avenue of attack is simply to observe that random test-sets are usually cheap in achieving a moderate coverage but they contain significant redundancy. We propose to combine the random test generation with our targeted test generation to achieve high structural coverage with comparatively low cost. Meanwhile we remove the redundant test cases from the random set. Currently our full path coverage test requirement ensures such redundancy removal to be straightforward. As stated in the text, less stringent requirements will make the redundancy removal a computationally hard problem. We plan to use optimisation techniques to provide a subset extraction facility later on. This approach requires only that you know what each test input actually achieves. The test-data could be generated by any method. Thus, this has the benefit of being able to be directly applied in almost any industrial test process. Similar ideas have been applied to regression test-sets [16].

This framework is conceptually extensible. As has been mentioned in the text, we intend extending the test-data generation tool to automatically generate test-data that can detect particular faults. The conceptual framework should extend to other architectural notations provided that the notation selected supports execution or simulation. Should there be any other advanced optimisation-based search technique or constraint solving techniques proved to be superior for some problems, such emerging tools can be easily incorporated. Many interesting questions arise. Can the architectural level test-data be used for code testing? What kind of refinement needs to be done? Can the refinement be done automatically? If so, there will be a large payback. If the developers maintain a fairly straightforward mapping of inputs and outputs when refining to code then the task is greatly facilitated.

Testing and analysis at the architectural level is now considered a crucial part of effective software development. We believe that our emerging automated test-data generation and test-set reduction techniques applied at the architectural level can form a useful complement to other automated techniques such as model checking and proofs of correctness.

References

1. B. Beizer: Software Testing Techniques. Thomason Computer Press, 2nd edition. 1990.
2. Leonardo Bottaci: Predicate Expression Cost Functions to Guide Evolutionary Search for Test Data. GECCO 2003.
3. J. Beiman, D. Dreilinger and L. Lin: Using Fault Injection to Increase Software Test Coverage. In Proc. 7th Int. Symp. on Software Reliability Engineering (ISSRE'96).
4. Jeffrey Voas and Gary McGraw: Software Fault Injection: Innoculating Programs Against Errors. By John Wiley & Sons, 1997
5. Hong Zhu, Patrick A. V. Hall and John H. R. May: Software Unit Test Coverage and Adequacy. ACM Computing Surveys, Vol. 29, No. 4 December 1997.
6. Nigel Tracey, John Clark, Keith Mander and John McDermid: An Automated Framework for Structural Test Data Generation. Automated Software Engineering 1998, Honolulu.
7. S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by Simulated Annealing. Science, 220(4598): 671-680, May 1983.
8. B. F Jones, H. Sthamer, and D. E. Eyres. Automatic Structural Testing Using Genetic Algorithms. Software Engineering Journal, 11(5): 299-306, 1996.
9. B. Korel. Automated Software Test Data Generation. IEEE Transactions on Software Engineering, 16(8): 870-879, August 1990.
10. C. R. Reeves (Ed.). Modern Heuristic Techniques for Combinatorial Problems. Blackwell Scientific Publications, Oxford, 1993.
11. J. Wegener, A. Baresel, and H. Sthamer. Evolutionary Test Environment for Automatic Structural Testing. Information and Software Technology, 43: 841-854, 2001.
12. Nigel Tracey, John Clark, John McDermid and Keith Mander. Integrating Safety Analysis with Automatic Test-Data Generation for Software Safety Verification. 17th International System Safety Conference. Pages 128-137. August 1999.
13. Nigel Tracey, John Clark, Keith Mander and John McDermid. Automated test-data generation for exception conditions. Software Practice and Experience, January 2000.

14. Nigel Tracey, John Clark and Keith Mander. Automated Program Flaw Finding using Simulated Annealing. International Symposium on Software Testing and Analysis (ISSTA) 1998.
15. N. Metropolis, A. W. Rosenbluth, A. H. Teller, and E. Teller. Equation of State Calculation by Fast Computing Machine. Journal of Chem. Phys., 21:1087-1091, 1953.
16. Ghinwa Baradhi and Nashat Mansour. A Comparative Study of Five Regression Testing Algorithms. In the Proceedings of the Australian Software Engineering Conference, 1997.
17. R. Kirner, R. Lang, G. Freiberger and P. Puschner. Fully Automatic Worst-Case Execution Time Analysis for Matlab/Simulink Models. 14th Euromicro Conference on Real-Time Systems (ECRTS'02), Austria, 2002.
18. Search Based Software Test Data generation: A Survey. Phil McMinn. Preprint (to appear in STVR). http://www.dcs.shef.ac.uk/~phil/pub/sbst.pdf

Search-Based Techniques for Optimizing Software Project Resource Allocation

G. Antoniol[1], M. Di Penta[1], and M. Harman[2]

[1] RCOST - Research Centre on Software Technology
University of Sannio, Department of Engineering
Palazzo ex Poste, Via Traiano 82100 Benevento, Italy antoniol@ieee.org, dipenta@unisannio.it
[2] Department of Information Systems and Computing,
Brunel University, Uxbridge, Middlesex, UB8 3PH, UK
mark.harman@brunel.ac.uk

Keywords: Software Project Management, Genetic Algorithms, Queuing Networks

1 Introduction

In software development, testing and maintenance, as in other large scale engineering activities, effective project planning is essential. Failure to plan and/or poor planning can cause delays and costs that, given timing and budget constraints, are often unacceptable, leading to business–critical failures. Traditional tools such as the Project Evaluation and Review Technique (PERT), the Critical Path Method (CPM), Gantt diagrams and Earned Value Analysis help to plan and track project milestones. While these tools and techniques are important, they cannot assist with the identification of optimal scheduling assignment in the presence of configurable resource allocation. However, most large scale software projects involve several teams of programmers and many individual project work packages. As such, the optimal allocation of teams of programmers (the primary resource cost drivers) to Work Packages (WPs) is an important problem which cannot be overlooked.

In this work we study this problem from the perspective of a massive software maintenance project. Typical examples are the Y2K remediation, Euro conversion or phone numbering change, involving a large number of applications simultaneously. Such maintenance activities present particularly acute problems for managers, since they have fixed hard deadlines and cut right across an entire software portfolio, touching almost every software asset possessed by the organisation.

When a massive maintenance request arrives, it is split in WPs according to the project Work-Breakdown Structure (WBS). An analogy estimate can be used to determine the effort required to maintain each WP. Having obtained estimates for effort, the next task is to determine the order in which WPs flow into the queuing system to be dealt with by the next available team of programmers [1].

The order of presentation of WPs is a way of describing the allocation of programmer teams to WPs. Such a resource allocation problem is an example of

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1425–1426, 2004.

a bin packing problem, the solution of which is NP-hard and, for which, evolutionary algorithms are known to be effective [2,4,3]. We performed an empirical study, using historical data from a real–world massive Y2K maintenance intervention, aiming at addressing the applicability of evolutionary algorithms to software resource allocation problems. While optimization techniques have been widely used in other fields, they have never been used for that specific field.

Three optimization techniques, namely hill climbing, simulated annealing and genetic algorithms, have been evaluated. Each was applied to two very different encoding strategies. Each encoding represents the way in which the work packages of the overall project are to be allocated to teams of programmers. The first encoding used is a scanning genome encoding, which combines the evolutionary algorithms with a queuing simulation model (the genome encoding represent the order in which the WPs flow to the maintenance process queuing model). The second genome is a vector-based genome, encoding the allocation of WPs to different maintenance teams.

Overall, the scanning genome encoding was found to outperform a vector-based genome encoding. In particular, when comparing different optimization techniques, we found that for the less optimal encoding (vector–based) the GA performed significantly better than the other approaches. For the optimal encoding (scanning), though GA starts better simulated annealing and hill climbing approaches soon catch up, so that the overall difference between the three approaches appears to be small, compared to the problem of establishing an effective encoding. Work-in progress is to perform in–depth analyses as well as to model more complex maintenance process, in which the ordering is also constrained by the precedence between maintenance tasks.

References

1. G. Antoniol, A. Cimitile, G. A. Di Lucca, and M. Di Penta. Assessing staffing needs for a software maintenance project through queuing simulation. *IEEE Transactions on Software Engineering,* 30(1):43–58, Jan 2004.
2. L. Davis. Job-shop scheduling with genetic algorithms. In *International Conference on GAs,* pages 136–140. Lawrence Erlbaum, 1985.
3. E. Falkenauer. *Genetic Algorithms and Grouping Problems.* Wiley-Inter Science, Wiley - NY, 1998.
4. E. Hart, D. Corne, and P. Ross. The state of the art in evolutionary scheduling. *Genetic Programming and Evolvable Machines,* 2004 (to appear).

Applying Evolutionary Testing to Search for Critical Defects

André Baresel, Harmen Sthamer, and Joachim Wegener

Software Methods and Tools, Research and Technolgy, Daimlerchrysler AG,
Alt-Moabit 96a , 10559 Berlin, Germany
{andre.baresel,harmen.sthamer,
joachim.wegener}@daimlerchrysler.com

Abstract. Software systems are used regularly in safety-relevant applications. Therefore, the occurrence of critical defects may not only cause costly recalls but may also endanger human lives. Accordingly, the development of software systems in industrial practice must comply with the highest quality requirements and standards. In practice, the most important analytical quality assurance method is dynamic testing and the most important activity to ensure this quality is test case determination. The effectiveness and efficiency of the test process can be clearly improved by Evolutionary Testing. Evolutionary Testing is a metaheuristic search technique for the generation of test cases.

1 Introduction

Testing is the most important quality assurance method for embedded systems. To increase the effectiveness and efficiency of the test and thus reduce the overall development costs, DaimlerChrysler Research works in the area of *Evolutionary Testing* [2]. Critical defects are determined using the classical Evolutionary Structural testing approach [1]. The search for defects has to generate inputs executing the statement under investigation as well as an additional condition which will lead to a defect situation. For this reason, we designed a new set of instrumentation functions that monitors during test executions actual values, array index values, pointer addresses, etc... We defined test goals for all possible locations of defects, e.g. arithmetic operations and conditions which could lead to erroneous program behavior. These test goals were investigated separately by evolutionary optimizations. To check the feasibility of this approach the software of a navigation system is used.

2 Critical Defects

Critical defects emerge from statements or conditions which are executed and result in a defect situation and thereby violate the system's integrity. Critical defects can cause data corruption, hang-ups, system crash and can thus even endanger human life. Often, the problem is difficult to be identified because only specific input scenarios lead to one of the defects known for software systems. In this project the following five defects were under investigation:

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1427–1428, 2004.

Erroneous Memory Accesses can be caused by erroneous array indices (out of bounds errors) or incorrect pointer usage (null pointer usage). Three defects are defined here in the category **Arithmetic Calculation Errors.** These are the well-known problems of overflow, underflow and division by zero. Depending on the processor, the programming language and the runtime system, underflow may be ignored and zero substituted for the unrepresentable value, though this might lead to a later division by zero error which cannot be ignored. The third critical defect is the **Violation of Assertions.** Assertions are software codes which check and monitor whether certain conditions within the software code are fulfilled. A major problem in software development are incorrect program constructs which lead to so-called **Endless Loops.** Our approach searches for find input scenarios which execute loops with the highest possible number of iterations. Producing erroneous values in case of **Casting to Smaller Data Types** may cause lost of data, leading to silent bugs. An example for this is the transformation of a number encoded with 32 bits into a number with 16 bits. The test goal detecting incorrect casting operations targets statements with explicit casting operations with the condition of maximizing/minimizing the operand of the cast function.

3 Conclusion

This paper introduces the idea of using evolutionary algorithms to generate test data for detecting critical defects in software functions. New instrumentation rules and the construction of a third component for the fitness function of have been developed for the classical Evolutionary Structural Testing. The authors argue that the application of search techniques to find test data, detecting critical defects, improves the test quality. Even in the case where no solution was found leading to an error, the evolutionary search creates test data that are close to the boundaries of error situations and therefore improves the confidence in the correctness of the software. Automatic test case design increases the effectiveness and efficiency of the test and thus reduce the overall development costs. Test data generation targeting critical defects can be used as an additional criterion to improve the quality of software products.

Acknowledgements. The work described has been performed within the SysTest project. The SysTest project is funded by the European Community under the 5th Framework Programme (GROWTH), project reference G1RD-CT-2002-00683.

References

[1] Wegener, J. ; Baresel, A.; Sthamer, H.: *Evolutionary test environment for automatic structural testing.* Information and Software Technology Special Issue on Software Engineering using Metaheuristic Innovative Algorithms. Vol. 43(14) (2001) 841 – 854

[2] Wegener, J.; Grochtmann, M.: *Verifying time constraints of real-time systems by means of evolutionary testing.* Real-Time Systems 15 (3). (1998) 275 – 298

Input Sequence Generation for Testing of Communicating Finite State Machines (CFSMs)

Karnig Derderian, Robert M. Hierons, Mark Harman, and Qiang Guo

Brunel University, Department of Information Systems and Computing
Uxbridge UB8 3PH, UK
{karnig.derderian,rob.hierons,mark.harman,qiang.guo}@brunel.ac.uk

1 Introduction

Finite State Machines (FSMs) have been used to model systems in different areas like sequential circuits, software development and communication protocols [1]. FSMs have been an effective method of modelling because a variety of techniques and automated tools exist that work with them.

Testing is an important part of the software engineering process and can account for up to 50% of the total cost of software development [2]. This motivates the study of testing FSMs to ensure the correct functioning of systems.

Some systems are more naturally modelled as a set of FSMs that operate concurrently or use messages to communicate with each other, rather than as a single FSM. FSMs that communicate by passing messages to each other are called Communicating Finite State Machines (CFSMs). CFSMs have input queues and communicate when a CFSM produces an output that is placed in the input queue of another CFSM. When a model *M,* consisting of CFSMs, receives an input this input triggers a sequence of local transitions within the individual CFSMs forming a global transition in *M*.

Under certain conditions a set of CFSMs $M_1, ..., M_n$ can be converted into an equivalent single FSM called the product machine [3]. However if n_i denotes the number of states in M_i, the product machine has $O(\prod_i n_i)$ states and hence suffers from a combinatorial explosion. This poster explores new alternatives to state and transition testing of CFSMs using Genetic Algorithm (GA) heuristics, without the use of a product machine.

Local transitions for CFSMs represent an internal transition of individual CFSMs. The testing effort can be reduced by testing all the local transitions instead of all the global transitions. Global transitions of the CFSM set involve at least one local transition in one of the CFSMs. Therefore discovering a fault in a local transition t that is used by a number of global transitions could reduce the testing effort. After passing a transition its final state has to be verified. In order to verify that the current local state of CFSM M_i is s it is sufficient to use an input sequence $u(s)$ such that it distinguishes global state σ, with $\sigma(i) = s$ from all other global states in which only the local state of M_i differs. A transition sequences with such property, given a number of conditions on the global state of M are met is known as a Constrained Identification Sequences (CIS) [4].

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1429–1430, 2004.

Similarly transition sequences called Unique Input/Output (UIO) sequences are sometimes used in state verification of FSMs [5].

This poster extends a previously suggested idea regarding local transitions and their corresponding final states and suggests the use of genetic algorithms to automatically find a test set. This poster shows how the problem of finding a robust set of test data for a set of CFSMs may be represented as an optimisation problem, where GAs are proven to contribute [6].

2 Input Sequence Generation

The focus is on how to use a novel method of finding CISs using genetic algorithms. Before the fitness function can be used a list of all the local transitions are linearly scanned and all the local transitions are ranked according to how common the input output pair is within the rest of the transitions. The penalty of the input and output pair of a transition will be calculated. A unique input output pair will have no penalty, while the most repeated input output pair will have the maximum cost. This notion of ranking the input output pairs will be used in determining the fitness of a potential CIS. The fitness of an input sequence is calculated by adding all the ranks of the transitions involved.

GAs using similar fitness measure have been successful in generating UIOs for up to 62% more of the states for an FSM compared to a random search [7]. Since the problem of generating CISs for CFSMs can be related to the problem of generating UIOs for FSMs [4] GAs might provide a good solution to the problem. The issues of concurrency and the invisibility of local transition remains with CIS, but the fitness functions and verification algorithm attempt to address these complications. Work in [5] also showed how GAs can outperform random search, however using a different fitness measure.

Future work will consider the problem of sequencing a set of CISs in order to minimise the cost of reaching the transition to be tested. This could be also represented as an optimisation problem and a solution attempted using GAs.

References

1. Tanenbaum, A.: Computer Networks. Prentice Hall (1996) 3rd edition.
2. Beizer, B.: Software testing techniques. Technical report (1990) 2nd edition.
3. Petrenko, A., Yavtushenko, N., Dssouli, R.: Testing strategies for communicating fsms. Protocol Test Systems (November 1995) 195–208
4. Hierons, R.: Checking states and transitions of a set of communicating finite state machines. Microprocessors and Microsystems,Special Issue on Testing and testing techniques for real-time embedded software systems, **24** (2001) 443 452,
5. Guo, Q., Hierons, R., Harman, M., Derderian, K.: Computing unique input/output sequences using genetic algorithms. In: LNCS vol. 2931, Springer (2004) 169–184
6. Goldderg, D.E.: Genetic Algorithms in search, optimisation and machine learning. Addison-Wesley Publishing Company (1989)
7. Derderian, K., Hierons, R., Harman, M., Guo, Q.: Generating unique input output sequences for finite state machine testing using genetic algorithms. IEE Electronics Letters (Under consideration) (2004)

TDSGen: An Environment Based on Hybrid Genetic Algorithms for Generation of Test Data

Luciano Petinati Ferreira and Silvia Regina Vergilio

Federal University of Parana (UFPR), CP: 19081,
CEP: 81531-970, Curitiba - Brazil
{petinati,silvia}@inf.ufpr.br

The software testing has gained importance and can be considered a fundamental activity to ensure the quality of the software being developed. In the literature, there are three groups of testing techniques proposed to reveal a great number of faults with minimal effort and costs: functional technique, structural technique and fault-based technique. These techniques are generally associated to testing criteria. A criterion is a predicate to be satisfied to consider the testing activity ended and the program tested enough [3]. The criterion generally requires the exercising of certain elements of the source code (decision statement) called required elements. To satisfy a testing criterion (coverage of 100%), it is necessary to provide input data to execute paths that exercise all the required elements. This is a very hard task because it is not possible its complete automatization due to several testing limitations.

The generation of test data has been addressed by many authors and different techniques, such as: random generation, generation based on symbolic execution and generation based on dynamic execution, have been applied with varying degrees of success, maybe by the testing limitations and the complexity inherent to the test generation task. Therefore some authors proposed the use of meta-heuristics algorithms, such as Genetic Algorithm, originating a new research field named Evolutionary Testing [4].

Many works address GA for test generation but most of them do not offer a testing environment for supporting the organization and the complete application of a testing strategy including different testing criteria.

We explore the use of GA for test data generation and describe an environment, named TDSGen (**T**est **D**ata **S**et **Gen**erator), that evolves a population of test inputs to satisfy different testing criteria. The goal is to obtain a population that represents a set that better satisfies the chosen criterion. TDSGen uses the coverage measurement for the fitness evaluation and implements different mechanisms based on the strategies: elitism, sharing and tabu lists.

TDSGen integrates two different testing tools, Poketool [1] and Proteum [2], allowing to apply structural and fault-based criteria of C programs. These tools are responsible for the generation of the required elements and for the coverage evaluation, by producing the list of the covered elements. TDSGen presents other advantages: the tester provides a configuration file and initial functional test set, no analysis of the program is necessary, it implements a mechanism to encode the program input allowing the testing of different types of programs

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1431–1432, 2004.

We have conducted some experiments using TDSGen with three strategies: a) random generation; b) GA based generation; c) HGA (Hybrid Genetic Algorithm) based generation, the uses the mechanisms implemented by TDSGen. These strategies were used for criteria: mutation analysis (MA), all-nodes (AN), all-edges (AE), all-potential-uses (PU), all-potential-uses/du (PUDU) and potential-du-paths (PDU). Table 1 presents the coverage obtained for each criterion and strategy.

Table 1. Coverage Obtained for Each Criterion and Strategy

Strategy	Size	AN	AE	PU	PDU	PUDU	MA
Random	10	68.18	7.34	7.34	7.34	8.67	36.27
	50	70.22	11.33	8.02	14.0	13.34	48.74
	200	75.68	26.67	28.00	31.33	32.00	63.50
GA	10	68.63	9.33	9.33	8.00	8.00	34.44
	50	69.31	10.67	12.00	11.33	13.33	48.75
	200	75.00	36.67	32.67	45.00	36.00	63.10
HGA	10	69.09	11.33	9.33	9.34	11.33	38.29
	50	74.54	32.00	30.67	40.0	40.00	52.72
	200	91.14	89.34	90.00	90.66	88.67	67.10

The results obtained in the experiments reveals an increase in the performance by using the mechanisms implemented by TDSGen, mainly the HGA strategy, without increase the costs and execution time. The mean runtime of the HGA strategy is lower than the GA strategy runtime.

The experiments showed yet that TDSGen is a promising tool for generation of test data sets. It has three important characteristics, that makes it different from the most works found in the literature: 1) uses a fitness function based on the coverage of the test case with the goal of satisfying a given testing criterion; 2) has mechanisms of memorization and hybridization to increase the performance of the GA; and 3) integrates two different testing tools.

An observed limitation is the difficulty to get a complete coverage because it is not always possible in due infeasible elements or equivalent mutants.

References

1. M.L. Chaim. *POKE-TOOL - Uma Ferramenta para Suporte ao Teste Estrutural de Programas Baseado em Análise de Fluxo de Dados.* Master Thesis, DCA/FEEC/Unicamp, Campinas - SP, Brazil, April 1991. (in Portuguese).
2. M. E. Delamaro and J.C. Maldonado. A tool for the assesment of test adequacy for c programs. In *Proceedings of the Conference on Performability in Computing Systems,* pages 79–95. East Brunswick, New Jersey, USA, July 1996.
3. S. Rapps and E.J. Weyuker. Selecting software test data using data flow information. *IEEE Trans, on Soft. Engineering,* SE-11(4):367–375, April 1985.
4. J. Wegener, A. Baresel, and H. Sthamer. Evolutionary test environment for automatic structural testing. *Information and Software Technology,* 43:841–854, 2001.

Author Index

Acan, Adnan II-838
Adamopoulos, Konstantinos II-1338
Agarwal, Amit II-850, II-859
Agogino, Adrian I-1
Agogino, Alice M. II-1030
Aguilar-Ruiz, Jesus S. I-493, I-828
Ahn, Chang Wook I-840
Akama, Kiyoshi II-222, II-246
Alba, Enrique I-852, I-864, I-889, I-1138
Albrecht, Andreas A. I-642
Alet, Fabien II-36
Ali, Walid II-869
Amamiya, Makoto II-1307
Andrews, Mark W. II-379
Antoniades, Athos I-1282
Antoniol, G. II-1425
Aporntewan, Chatchawit I-877
Araujo, Lourdes I-889
Areibi, Shawki II-1138
Ash, A.S. II-1330
Azad, Atif II-654

Bacardit, Jaume I-828, II-726
Bader-Natal, Ari I-585
Bae, Seung-Hee II-381
Bailleux, Olivier I-175
Balan, Gabriel Catalin II-422
Ballester, Pedro J. I-901, II-1299
Bambha, Neal K. II-383
Bandte, Oliver II-883
Banzhaf, Wolfgang II-557
Barán, Benjamín I-259
Barbieri, Alan II-1301
Barbosa, Helio J.C. I-368
Baresel, Andre II-1350, II-1427
Bassett, Jeffrey K. I-914
Bayarou, Kpatscha M. I-299
Belda, Ignasi I-321
Beretta, Mauro II-1251
Berlik, Stefan I-786
Bernstein, Yaniv II-702
Berro, Alain I-251
Beslon, Guillaume II-406
Beyer, Hans-Georg I-654
Bhanu, Bir I-587, II-896
Bhattacharyya, Shuvra S. II-383
Blume, Christian I-790
Bonabeau, Eric II-1151
Bongard, Josh C. I-333
Brabazon, Anthony I-12, I-163, II-447, II-617
Branke, Jürgen I-923, II-434
Bretthauer, Georg I-790
Britt, Winard I-1078
Brizuela, Carlos A. II-1305
Brown, Douglas I-263
Bruck, Torben II-1078
Brüderle, Daniel I-1316
Bucci, Anthony I-501
Buehler, Erik C. I-495
Bühler, Oliver II-1400
Bui, Thang N. I-24, I-36, II-908
Buller, Andrzej I-627, II-408
Burns, Daniel J. I-497
Bush, Keith II-282
Butz, Martin V. II-739, II-751
Buyukbozkirli, Bulent I-935
Byde, Andrew I-1066

Cagnoni, Stefano II-1301
Cain, Krystal I-263
Cantú-Paz, Erick I-947, I-959
Cardona, Cesar I-1401
Carter, Jonathan N. I-901, II-1299
Carvalho, Andre de I-678
Castagliola, Philippe II-90
Castanier, Bruno II-90
Cattolico, Mike II-531
Chabrier, Jean-Jacques I-175
Chang, Ming I-513
Chen, Jian-Hung I-737
Chen, Jun II-318
Chen, Yen-Chih II-385
Chen, Yen-Fu I-481
Chen, Ying-ping I-971, I-1426, II-1314
Cheng, Chihyung Derrick I-983
Chew, Chan Yee II-850
Chew, Lawrence II-1078
Chia, Henry Wai-Kit II-836
Chicano, J. Francisco I-852

Chitty, Darren M. I-48, I-253
Chiu, Yi-Yuan I-481
Choi, Sung-Soon I-994, II-150, II-398
Choi, Yoon-Seok II-1303
Chongstitvatana, Prabhas I-877
Chow, Rick I-1006
Chryssomalakos, Chryssomalis I-1018
Ciesielski, Vic II-702, II-1114
Clark, John A. II-569, II-1413
Clergue, Manuel II-690
Clevenger, Lauren M. I-666
Cliff, Dave I-1066
Coello Coello, Carlos A. I-225, I-700
Coffey, Christopher S. I-438
Colavolpe, Giulio II-1301
Collard, Philippe II-690
Collet, Pierre I-175
Cornforth, David I-60
Costa, Ernesto I-12, II-416, II-666
Crane, Ellery Fussell II-593
Cripe, Greg II-330
Cruz, R. II-1330
Cully, Jack F. I-495
Cunningham, Hurley I-1078
Curran, Dara I-72

Dai, Honghua I-275
Dallaali, Mohammad Amin II-387
Dardenne, Laurent E. I-368
Das, Sanjoy I-356,I-495
Dasgupta, Dipankar I-287
Day, Richard O. II-402
Deb, Kalyanmoy I-1042, I-1054, II-920
Delbem, A.C.B. I-678
Dempsey, Ian II-447
Derderian, Karnig II-1429
Devireddy, Venkat II-390
Dewri, Rinku II-920
Dicke, Elizabeth I-1066
Dignum, Stephen I-255
Divina, Federico I-828
Doom, Travis E. I-426
Downing, Keith L. I-81
Dozier, Gerry I-263, I-1078
Drewes, Rich I-257, I-1365
Droste, Stefan I-1088
Duong, Vu I-792
Durand, Frédo I-188

Eckert, Claudia I-299
Edmonds, Camilla I-308
Eikelder, Huub M.M. ten I-549
Elliott, Lionel II-932, II-945
Er, Meng Joo II-850, II-859
Eryigit, Gulsen II-271
Eskridge, Brent E. II-459
Estrada-Esquivel, Hugo II-1

Fan, Zhun II-722
Farooq, Muddassar II-1334
Favrel, Joël II-406
Ferguson, Ian I-792
Fernandez, Thomas II-471
Fernlund, Hans II-704
Ferra de Sousa, Tiago I-12
Ferrandi, Fabrizio II-763
Fischer, Simon I-1100, I-1113
Foster, James A. I-449
Frommer, Ian II-392
Fukunaga, Alex S. II-483
Funes, Pablo II-434

Gabriel, Philip I-1340
Garcia, Anderson C. I-678
Garibay, Ivan I. I-1125
Garibay, Ozlem O. I-1125
Garrell, Josep Maria II-726
Giacobini, Mario I-1138
Giraldez, Raul I-493
Giralt, Ernest I-321
Goldberg, David E. I-840, I-971, I-1426, II-11, II-114, II-126, II-355, II-367, II-739, II-751, II-1314
Golden, Bruce II-392
Gomez, Faustino J. II-957
Gomez, Jonatan I-1150, I-1162, II-1312
Gómez, Osvaldo I-259
Gómez de Silva Garza, Andrés II-394
Gonzalez, Avelino J. II-704
González, Luis C. II-1305
Goodman, Erik D. I-935, I-1220, II-722
Goodman, Philip I-257
Gourdin, Thierry II-810
Grahl, Jörn I-1174
Grajdeanu, Adrian I-1186
Grasemann, Uli II-969
Greene, William A. I-1197
Grogono, P. I-93
Grosan, Crina I-788

Groumpos, P.P. I-402
Guo, Qiang II-1429
Guo, Xin I-792
Gupta, Naveen Kumar I-1042

Hahn, Lance W. I-392
Hamza, Karim II-981
Handa, Hisashi II-396
Haney, Keith II-1078
Hang, Xiaoshu I-275
Harman, Mark II-1338, II-1425, 11-1429
Harmon, Scott II-706
Hart, William E. I-666
Hernandez, Marcel L. I-48
Hey wood, Malcolm I. II-581, II-678
Hidović, Džena I-725, II-1005
Hierons, Robert M. II-1338, II-1429
Hilbers, Peter A.J. I-549
Hinkemeyer, Brenda II-82
Hirasawa, Kotaro II-710
Ho, Shinn-Jang I-737
Ho, Shinn-Ying I-737
Hoai, Nguyen Xuan II-605
Hodjat, Babak II-1307
Holcombe, Mike II-1363
Honda, Karen I-678
Hong, Q.Y. II-1310
Hornby, Gregory S. II-495
Hougen, Dean F. II-459
Howard, Brian I-1208
Hsu, William II-706
Hu, Guang-da I-318
Hu, Jianjun I-1220, II-722
Hu, Jinglu II-710
Huang, Chien-Feng I-1233
Huang, Chung-Yuan II-774
Hung, Ming-Hao I-737
Hurley, John I-263
Hussain, Talib II-1017

Iba, Hitoshi I-414, I-590, I-774, 11-708
Ingham, Derek B. II-932,II-945
Inoue, Yutaka I-590
Iorio, Antony W. I-537
Ishibuchi, Hisao I-1246,I-1259
Ito, Junichi II-1307

Jakob, Wilfried I-790
Janikow, Cezary Z. II-507
Jarvis, Alex II-593
Jennings, Annika II-1078
Jensen, Chris II-1090
Ji, Zhou I-287
Jiang, Ning II-418
Jin, Yaochu I-688
Johnston, Roy L. II-1316
Jong, Edwin D. de I-501, I-525, I-1030
Jong, Kenneth A. De I-914, I-1186
Jordaan, Elsa II-1078
Julstrom, Bryant A. I-1272, I-1282, II-82
Jung, Jae-Yoon II-519
Jung, Soonchul I-1292
Jung, Sungwon II-1214
Just, Winfried I-499

Kamalian, Raffi II-1030
Kamper, Andreas I-923
Kang, Lishan II-1336
Kao, Cheng-Yan II-385
Keedwell, Edward II-1042
Keenan, Peter II-617
Keijzer, Maarten II-531
Kerber, Manfred II-785
Keymeulen, Didier I-792
Kharma, Nawwaf I-93,II-1090
Khu, Soon-Thiam II-1042
Kim, Haejoong II-1214
Kim, Jong-Pil II-1054
Kim, Jung-Hwan II-398
Kim, Yong-Hyuk I-346, II-400, II-1054, II-1065
Kirley, Michael I-60
Kirshenbaum, Evan II-543
Klau, Gunnar W. I-1304
Kleeman, Mark P. II-402
Knibbe, Carole II-406
Knight, James II-294
Koduru, Praveen I-356
Kordon, Arthur II-1078
Kosorukoff, Alexander I-983
Kovacs, Tim II-785
Kowaliw, T. I-93,II-1090
Kwon, Yung-Keun II-404, II-1102
Kwong, Sam II-1310
Kyaw, Maung Ye Win II-859
Kyne, Adrian G. II-932, II-945

Lam, Brian II-1114
Lammermann, Frank II-1350
Lamont, Gary B. II-402

Langdon, W.B. II-343
Langeheine, Jörg I-1316
Lanzi, Pier Luca II-739, II-751, II-763
Layzell, Paul I-1066
Lee, Su-Yeon I-346
Lefort, Virginie II-406
Leier, André II-557
Leon, Elizabeth II-1312, II-1324
Leong, Yew Kong II-850
Leung, Kwong-Sak II-1126
Li, Mingbiao II-1336
Li, Xiaobo II-1320
Li, Xiaodong I-105, I-117, I-537, II -702
Liang, Yong II-1126
Liekens, Anthony M.L. I-549
Lim, Meng-Hiot II-850, II-859
Lima, Cláudio F. I-1328, II-412
Lin, Tz-Kai II-24
Lin, Yingqiang II-896
Lipson, Hod I-333
Lischka, Mario II-1334
Liu, Hongwei II-708
Liu, Juan II-408
Liu, Xiaohui II-162
Liu, Ziwei II-420
Ljubić, Ivana I-1304
Llorà, Xavier I-321, II-797, II-1314
Lloyd, Lesley D. II-1316
Lobo, Fernando G. I-1328, II-410, II-412
Louis, Sushil J. I-257, I-1365
Lu, Guangfa II-1138
Luke, Sean I-573,II-422,II-630
Lunacek, Monte I-1340, II-294
Luque, Gabriel I-864, I-889

Mabu, Shingo II-710
Maciokas, James I-257
Magalhães, Camila S. de I-368
Majeed, Hammad II-654
Majumdar, Saptarshi II-920
Malinchik, Sergey II-883, II-1151
Mancoridis, Spiros II-1375
Maniadakis, Michail I-640
Martikainen, Jarno II-1162
Martinell, Marc I-321
Mártires, Hugo II-412
Maslov, Igor V. II-1177
Massey, Paul II-569
Mateos, Daniel I-493
Matsui, Shouichi II-1318
Matthews, Robin I-12
Mauri, Giancarlo I-380
May, Kevin T. I-497
Mc Garraghy, Sean II-617
McIntyre, A.R. II-581
McKay, R.I. II-605
McMinn, Phil II-1363
McPhee, Nicholas Freitag II-593
Meier, Karlheinz I-1316
Menon, Anil I-1352
Mera, Nicolae S. II-932, II-945
Mezura-Montes, Efrén I-700
Miikkulainen, Risto II-69, II-957, II-969, II-1226
Miles, Chris I-1365
Miller, Julian Francis I-129
Miramontes Hercog, Luis II-993
Mitchell, Brian S. II-1375
Mitra, Kishalay II-920
Moghnieh, Hussein II-1090
Mohr, Jonathan II-1320
Mok, Tony Shu Kam II-1126
Monson, Christopher K. I-140
Montana, David II-1017
Moon, Byung-Ro I-346, I-994, I-1292, II-150, II-381, II-398, II-400, II-404, II-1054, II-1065, II-1102, II-1303
Moore, Jason H. I-392, I-438
Moore, Rashad L. II-1322
Moraglio, Alberto I-1377
Mosca, Roberto I-380
Moser, Andreas I-1304
Moura Oliveira, P.B. de I-615
Mumford, Christine L. I-1389
Munetomo, Masaharu II-222, II-246
Murata, Tadahiko II-712
Mutzel, Petra I-1304

Nadimi, Sohail I-587
Nagata, Yuichi II-1189
Nakamura, Takashi II-712
Nakano, Tadashi I-151
Narukawa, Kaname I-1246
Nasraoui, Olfa I-1401, II-1312, II-1324
Neumann, Frank I-713
Neuner, Philipp I-1304
Nicolau, Miguel I-1414, II-617
Northern, James II-1326

O'Neill, Michael I-12, I-163, II-447, II-617
O'Reilly, Una-May I-188
O'Riordan, Colm I-72
Ocenasek, Jiri II-36
Ohkura, Kazuhiro I-513
Ohnishi, Kei I-1426, II-1314
Oliver-Morales, Carlos II-719
Oppacher, Franz II-642, II-1263
Ovaska, Seppo J. II-1162

Paine, Rainer W. I-603
Pal, Koushik I-1054
Palacios-Durazo, Ramón Alfonso I-561
Panait, Liviu I-573, II-630
Papageorgiou, E.I. I-402
Park, Jinwoo II-1214
Parsopoulos, K.E. I-402
Paul, Topon Kumar I-414
Pavesi, Giulio I-380
Paz-Ramos, Marco Antonio II-1
Pedersen, Gerulf K.M. II-11
Pelikan, Martin II-24, II-36, II-48
Penta, M. Di II-1425
Pereira, Francisco B. II-416
Pesch, Hans-Josef II-258
Peterson, Michael R. I-426
Petinati Ferreira, Luciano II-1431
Pferschy, Ulrich I-1304
Piaseczny, Wojciech II-715
Pilat, Marcin L. II-642
Pinto, Adriano K.O. I-678
Pizzi, Nicolino II-1388
Planatscher, Hannes I-471
Poli, Riccardo I-255, I-1377, II-343
Policastro, Claudio A. I-678
Pollack, Jordan B. I-501, I-585, I-804
Poo, Tong Kiang II-850
Pope, Nigel II-1332
Potter, Mitchell A. I-914
Pourkashanian, Mohamed II-932, II-945
Premaratne, Malin II-387
Pulavarty, Sandeep II-343
Pundoor, Guruprasad II-392

Qi, Zhen-qiang I-318
Quan, Wei II-717
Quintero-Marmol-Marquez, Enrique II-1

Raidl, Günther I-1304
Ramakrishna, R.S. I-840
Rashidi, Farzan II-60
Rashidi, Mehran II-60
Ray, Thomas I-627
Raymer, Michael L. I-426
Reed, Patrick II-390
Reggia, James A. II-519
Reisinger, Joseph II-69
Renslow, Mark A. II-82
Rieffel, John I-804
Rigal, Laure II-90
Riquelme, Jose C. I-493
Ritchie, Marylyn D. I-438
Rizzo, Joseph R. I-24
Rocha, Luis M. I-1233
Rodríguez, Edwin II-706
Rodriguez-Tello, Eduardo II-102
Rodríguez-Vázquez, Katya II-719
Roe, Judith L. I-356
Rojas, Carlos I-1401
Romero, Heidi J. II-1305
Rosenberg, Ronald C. II-722
Rotar, Corina II-414
Rothlauf, Franz I-1174, II-258
Rowe, Jonathan E. I-725, II-282, II-1005
Ryan, Conor I-1414, II-531, II-654

Saitou, Kazuhiro II-981
Salhi, Said II-1316
Salzberg, Christopher II-379
Sammartino, Luca II-1251
Sanchez, Eduardo I-816
Sanchez, Stephane I-251
Sanderson, Rian II-1201
Sapin, Emmanuel I-175
Sariel, Sanem II-271
Sastry, Kumara I-1426, II-48, II-114, II-126
Sato, Yuji II-1328
Sawai, Hidefumi II-715
Schemmel, Johannes I-1316
Schmeck, Hartmut I-923
Schmitt, Lothar M. II-138
Sciuto, Donatella II-763
Semet, Yann I-188
Sendhoff, Bernhard I-688
Seo, Dong-Il II-150
Seo, Kisung II-722
Seppi, Kevin D. I-140
Shanblatt, Michael II-1326

Shen, Tsai-Wei I-481
Sheneman, Luke I-449
Sheng, Weiguo II-162
Sheppard, John I-1208, II-1322
Shibata, Youhei I-1259
Shu, Li-Sun I-737
Sigaud, Olivier II-810
Silva, Arlindo I-12
Silva, Sara II-666
Sim, Eoksu II-1214
Simonov, Mikhail II-1251
Smits, Guido II-1078
Solteiro Pires, E.J. I-615
Soltoggio, Andrea II-174
Song, Andy II-702
Song, Shen-min I-318
Soroldoni, Massimo II-1251
Soule, Terence II-307, II-717
Speer, Nora I-461
Spieth, Christian I-461,I-471
Stanley, Kenneth O. II-69, II-1226
Stautner, Marc II-1287
Stephens, Christopher R. I-1018, II-343, II-1330
Stephens, Graeme I-1340
Stepney, Susan II-569
Sthamer, Harmen II-1427
Stibor, Thomas I-299
Stoica, Adrian I-792
Storch, Tobias I-748
Streeter, Matthew J. II-186
Streichert, Felix I-461, I-471, II-1239
Stringer, Hal II-198
Suda, Tatsuya I-151
Suermondt, Henri J. II-543
Sugiyama, Masaharu I-513
Sun, Chuen-Tsai II-774
Sun, Xiaolu I-499
Sundarraj, Gnanasekaran I-36
Suzuki, Hideaki II-715

Takagi, Hideyuki II-1030
Talbott, Walter A. I-201
Talley, S. II-1330
Tan, Chew-Lim II-836
Tan, Xuejun II-896
Tanev, Ivan I-213,I-627
Tang, Zhilong II-1336
Tani, Jun I-603
Taniguchi, Ken II-724
Tarragó, Teresa I-321
Tavares, Jorge II-416
Tay, Joc Cing II-210
Teich, Jürgen II-383
Tenreiro Machado, J.A. I-615
Terano, Takao II-724
Terrio, M. David II-678
Tettamanzi, Andrea I-1138, II-1251
Tezuka, Masaru II-222
Thiele, Frederik II-434
Thierens, Dirk I-1030, II-234
Thoma, Yann I-816
Timmis, Jon I-308
Tohge, Takahiro I-590
Tokoro, Ken-ichi II-1318
Tomassini, Marco I-1138, II-690
Topchy, Alexander II-869
Torres-Jimenez, Jose II-1, II-102
Toscano Pulido, Gregorio I-225
Trahanias, Panos I-640
Traverso, Martin II-1375
Trebst, Simon II-36
Trefzer, Martin I-1316
Troyer, Matthias II-36
Tsai, Chi-Hung II-385
Tsuji, Miwako II-246
Tucker, Allan II-162
Tulai, Alexander F. II-1263
Tumer, Kagan I-1
Tzschoppe, Carsten II-258

Ueda, Kanji I-513
Ulmer, Holger I-471, II-1239
Unveren, Ahmet II-838
Uyar, Sima II-271

Valenzuela-Rendón, Manuel I-561
Vanneschi, Leonardo II-690
Ványi, Róbert II-1275
Vérel, Sébastien II-690
Vergilio, Silvia Regina II-1431
Vidaver, Gordon II-1017
Vivanco, Rodrigo II-1388
Voges, Kevin II-1332
Vrahatis, M.N. I-402

Waelbroeck, H. II-1330
Wang, H.L. II-1310
Watanabe, Isamu II-1318
Wedde, Horst F. II-1334

Wegener, Ingo I-713, I-1113
Wegener, Joachim II-1350, II-1400, II-1427
Weinert, Klaus II-1287
Weiskircher, René I-1304
Welch, Stephen I-356
Welge, Michael E. II-1314
Whitley, Darrell I-1340, II-282, II-294
Whittaker, Sean II-932
Wibowo, Djoko II-210
Wiegand, R. Paul I-573
Williams, Ashley II-1322
Willis-Ford, Christopher II-307
Wilson, Christopher W. II-945
Wilson, Stewart W. II-797, II-824
Wineberg, Mark II-318
Witt, Carsten I-761
Wright, Alden II-330, II-343
Wu, Annie S. I-1125, II-198, II-418
Wu, Zhijian II-1336
Wuillemin, Pierre-Henri II-810

Xie, Xiao-Feng I-238, I-261

Yanai, Kohsuke I-774
Yang, Jinn-Moon I-481, II-385
Yang, Zhao-hua I-318
Yao, Jie II-1090
Youssef, Waleed A. II-908
Yu, Han II-418
Yu, Jiangang II-896
Yu, Tian-Li II-355, II-367
Yuta, Kikuo I-213

Zamora Lores, Aram II-394
Zebulum, Ricardo I-792
Zell, Andreas I-461, I-471, II-1239
Zhan, Yuan II-1413
Zhang, Fu-en I-318
Zhang, Funing I-1078
Zhang, Wen-Jun I-238, I-261
Zhong, Christopher II-706
Zhu, Kenny Q. II-420
Zitzler, Eckart II-383
Zou, Jun II-1336