

Commenced Publication in 1973

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison

Lancaster University, UK

Takeo Kanade

Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler

University of Surrey, Guildford, UK

Jon M. Kleinberg

Cornell University, Ithaca, NY, USA

Friedemann Mattern

ETH Zurich, Switzerland

John C. Mitchell

Stanford University, CA, USA

Moni Naor

Weizmann Institute of Science, Rehovot, Israel

Oscar Nierstrasz

University of Bern, Switzerland

C. Pandu Rangan

Indian Institute of Technology, Madras, India

Bernhard Steffen

University of Dortmund, Germany

Madhu Sudan

Massachusetts Institute of Technology, MA, USA

Demetri Terzopoulos

University of California, Los Angeles, CA, USA

Doug Tygar

University of California, Berkeley, CA, USA

Moshe Y. Vardi

Rice University, Houston, TX, USA

Gerhard Weikum

Max-Planck Institute of Computer Science, Saarbruecken, Germany

Francisco Almeida María J. Blesa Aguilera
Christian Blum José Marcos Moreno Vega
Melquíades Pérez Pérez
Andrea Roli Michael Sampels (Eds.)

Hybrid Metaheuristics

Third International Workshop, HM 2006
Gran Canaria, Spain, October 13-14, 2006
Proceedings

Volume Editors

Francisco Almeida

José Marcos Moreno Vega

Melquíades Pérez Pérez

DEIOC Universidad de La Laguna

Escuela Técnica Superior en Ingeniería Informática

Avda. Astrofísico Francisco Sánchez, s/n, 38271 La Laguna, Tenerife, Spain

E-mail: {falmeida, jmmoreno, melperez}@ull.es

María J. Blesa Aguilera

Christian Blum

Universitat Politècnica de Catalunya, ALBCOM research group

Omega Campus Nord, Jordi Girona 1-3, 08034 Barcelona, Spain

E-mail: {mjblesa, cblum}@lsi.upc.edu

Andrea Roli

Università degli Studi "G. D'Annunzio"

Dipartimento di Scienze

Viale Pindaro 42, 65127 Pescara, Italy

E-mail: a.roli@unich.it

Michael Sampels

Université Libre de Bruxelles

IRIDIA CP 194/6

Avenue Franklin D. Roosevelt 50, 1050 Bruxelles, Belgium

E-mail: msampels@ulb.ac.be

Library of Congress Control Number: 2006933415

CR Subject Classification (1998): F.2, F.1, G.1.6, G.1.2, G.2.1, I.2

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN 0302-9743

ISBN-10 3-540-46384-4 Springer Berlin Heidelberg New York

ISBN-13 978-3-540-46384-9 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

Springer is a part of Springer Science+Business Media

springer.com

© Springer-Verlag Berlin Heidelberg 2006

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper SPIN: 11890584 06/3142 5 4 3 2 1 0

Preface

The International Workshop on Hybrid Metaheuristics reached its third edition with HM 2006. The active and successful participation in the past editions was a clear indication that the research community on metaheuristics and related areas felt the need for a forum to discuss specific aspects of hybridization of metaheuristics.

The selection of papers for HM 2006 consolidated some of the mainstream issues that have emerged from the past editions. Firstly, there are prominent examples of effective hybrid techniques whose design and implementation were motivated by challenging real-world applications. We believe this is particularly important for two reasons: on the one hand, researchers are conscious that the primary goal of developing algorithms is to solve relevant real-life problems; on the other hand, the path toward efficient solving methods for practical problems is a source of new outstanding ideas and theories.

A second important issue is that the research community on metaheuristics has become increasingly interested in and open to techniques and methods known from artificial intelligence (AI) and operations research (OR). So far, the most representative examples of such integration have been the use of AI/OR techniques as subordinates of metaheuristic methods. As a historical and etymological note, this is in perfect accordance with the original meaning of a *metaheuristic* as a “general strategy controlling a subordinate heuristic.”

The awareness of the need for a sound experimental methodology is a third keypoint. This aspect has gained more relevance and currency, even though there are still no widely agreed standard methodologies. As research on hybrid metaheuristics is mostly based on experimental methods, similar standards to those found in the evaluation of experiments in natural sciences can be expected.

Scientific testing, a fourth notable aspect, emerges as a fundamental methodology for understanding the behavior of algorithms. The goal of scientific testing is to abstract from actual implementations and study, empirically and through predictive models, the effect of algorithmic components. This research approach can be particularly useful in the case of conjectures on metaheuristic algorithm behavior that, while being widespread in the community, have not yet been the subject of validation.

Finally, a tendency to reconsider hybrid metaheuristics from a higher and more general perspective is emerging. Providing classifications, systematic analyses and surveys on important branches underlines a certain maturity of the relatively young field.

This progression can be observed by an increasing number of submissions to the workshop: we received 42 paper submissions to HM 2006. Each submitted paper was sent to at least three reviewers. We are very grateful to the members of the Program Committee and the additional reviewers for the effort they made

in carefully examining the papers and for the many valuable comments and suggestions they gave to the authors. Based on their comments, we finally accepted 13 submissions for publication and for presentation at HM 2006, resulting in an acceptance rate of roughly 31 %. In addition, we got one invited paper. The selection of papers was rather strict in order to guarantee the high quality of the proceedings and the workshop itself. We would like to thank all authors for their interest in our workshop.

The field of hybrid metaheuristics is the result of the composition of numerous streams in the field of algorithmics. However, these streams have increasingly come together and the main issues and characteristics of the field have evolved more clearly. For the future, we envision a scenario in which some challenges have to be faced:

- It should become common practice that experimental analysis meets high quality standards. This empirical approach is absolutely necessary to produce objective and reproducible results and to anchor the successes of metaheuristics in real-world applications.
- Hybrid metaheuristic techniques have to be openly compared not just among themselves but also with state-of-the-art methods, from whatever field they are. By following this approach, researchers would be able to design techniques that meet the goal of solving a real-world problem and to consider the other approaches as rich sources of design components and ideas.
- Scientific testing and theoretical models of algorithms for studying their behavior are still confined to a limited area of research. We believe that, by being able to explain rigorously algorithm behavior by means of sound empirical investigation and formal models, researchers would give the field a firmer status and give support to the development of real-world applications.

The achievement of these goals will take some time in view of the difficult theoretical and practical problems involved in these challenges. Nevertheless, research is very active and has already produced some remarkable results and studies in this direction.

August 2006

Francisco Almeida
 María J. Blesa
 Christian Blum
 J. Marcos Moreno
 Melquiades Pérez
 Andrea Roli
 Michael Sampels

Organization

Program Chairs

María J. Blesa	Universitat Politècnica de Catalunya, Barcelona, Spain
Christian Blum	Universitat Politècnica de Catalunya, Barcelona, Spain
Andrea Roli	Università degli Studi “G. D’Annunzio”, Chieti-Pescara, Italy
Michael Sampels	Université Libre de Bruxelles, Belgium

Workshop Chairs and Local Organization

Francisco Almeida	Universidad de La Laguna, Tenerife, Spain
J. Marcos Moreno	Universidad de La Laguna, Tenerife, Spain
Melquíades Pérez	Universidad de La Laguna, Tenerife, Spain

Program Committee

Thomas Bartz-Beielstein	Universität Dortmund, Germany
Mauro Birattari	Université Libre de Bruxelles, Belgium
Ralf Bruns	Fachhochschule Hannover, Germany
Francisco Chicano	Universidad de Málaga, Spain
Óscar Cerdón	Universidad de Granada, Spain
Carlos Cotta	Universidad de Málaga, Spain
Luca Di Gaspero	Università degli Studi di Udine, Italy
Marco Dorigo	Université Libre de Bruxelles, Belgium
Joshua Knowles	University of Manchester, UK
Andrea Lodi	Università degli Studi di Bologna, Italy
Vittorio Maniezzo	Università degli Studi di Bologna, Italy
Belén Melián Batista	Universidad de La Laguna, Spain
Daniel Merkle	Universität Leipzig, Germany
Bernd Meyer	Monash University, Australia
Martin Middendorf	Universität Leipzig, Germany
José A. Moreno	Universidad de La Laguna, Spain
David Pelta	Universidad de Granada, Spain
Steven Prestwich	4C, Cork, Ireland
Günther Raidl	Technische Universität Wien, Austria
Andrea Schaerf	Università degli Studi di Udine, Italy
Thomas Stützle	Technische Universität Darmstadt, Germany

VIII Organization

El-Ghazali Talbi	École Polytechnique Universitaire de Lille, France
Fatos Xhafa	Universitat Politècnica de Catalunya, Spain
Pascal Van Hentenryck	Brown University, Providence, USA
José Luis Verdegay	Universidad de Granada, Spain

Additional Referees

Dan Ashlock, Emilie Danna, Marta Kasprzak, Michele Monaci, Alena Shmygelska,
Peter J. Stuckey, Hande Yaman

Table of Contents

A Unified View on Hybrid Metaheuristics	1
<i>Günther R. Raidl</i>	
Packing Problems with Soft Rectangles	13
<i>Toshihide Ibaraki, Kouji Nakamura</i>	
A Multi-population Parallel Genetic Algorithm for Highly Constrained Continuous Galvanizing Line Scheduling	28
<i>Muzaffer Kapanoglu, Ilker Ozan Koc</i>	
Improvement in the Performance of Island Based Genetic Algorithms Through Path Relinking	42
<i>Luis delaOssa, José A. Gámez, José M. Puerta</i>	
Using Datamining Techniques to Help Metaheuristics: A Short Survey ...	57
<i>Laetitia Jourdan, Clarisse Dhaenens, El-Ghazali Talbi</i>	
An Iterated Local Search Heuristic for a Capacitated Hub Location Problem	70
<i>Inmaculada Rodríguez-Martín, Juan-José Salazar-González</i>	
Using Memory to Improve the VNS Metaheuristic for the Design of SDH/WDM Networks	82
<i>Belén Melián</i>	
Multi-level Ant Colony Optimization for DNA Sequencing by Hybridization	94
<i>Christian Blum, Mateu Yábar Vallès</i>	
Hybrid Approaches for Rostering: A Case Study in the Integration of Constraint Programming and Local Search	110
<i>Raffaele Cipriano, Luca Di Gaspero, Agostino Dovier</i>	
A Reactive Greedy Randomized Variable Neighborhood Tabu Search for the Vehicle Routing Problem with Time Windows	124
<i>Panagiotis P. Repoussis, Dimitris C. Paraskevopoulos, Christos D. Tarantilis, George Ioannou</i>	
Incorporating Inference into Evolutionary Algorithms for Max-CSP	139
<i>Madalina Ionita, Cornelius Croitoru, Mihaela Breaban</i>	

Scheduling Social Golfers with Memetic Evolutionary Programming 150
*Carlos Cotta, Iván Dotú, Antonio J. Fernández,
Pascal Van Hentenryck*

Colour Reassignment in Tabu Search for the Graph Set T-Colouring
Problem 162
Marco Chiarandini, Thomas Stützle, Kim S. Larsen

Investigation of One-Go Evolution Strategy/Quasi-Newton
Hybridizations 178
Thomas Bartz-Beielstein, Mike Preuss, Günter Rudolph

Author Index 193

A Unified View on Hybrid Metaheuristics^{*}

Günther R. Raidl

Institute of Computer Graphics and Algorithms
Vienna University of Technology, Vienna, Austria
`raidl@ads.tuwien.ac.at`

Abstract. Manifold possibilities of hybridizing individual metaheuristics with each other and/or with algorithms from other fields exist. A large number of publications documents the benefits and great success of such hybrids. This article overviews several popular hybridization approaches and classifies them based on various characteristics. In particular with respect to low-level hybrids of different metaheuristics, a unified view based on a common pool template is described. It helps in making similarities and different key components of existing metaheuristics explicit. We then consider these key components as a toolbox for building new, effective hybrid metaheuristics. This approach of thinking seems to be superior to sticking too strongly to the philosophies and historical backgrounds behind the different metaheuristic paradigms. Finally, particularly promising possibilities of combining metaheuristics with constraint programming and integer programming techniques are highlighted.

1 Introduction

Metaheuristics have proven to be highly useful for approximately solving difficult optimization problems in practice. A general overview on this research area can be found e.g. in [1], for more information see also [2,3]. The term *metaheuristic* was first introduced by Glover [4]. Today, it refers to a broad class of algorithmic concepts for optimization and problem solving, and the boundaries are somewhat fuzzy. Voß [5] gives the following definition:

A metaheuristic is an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high-quality solutions. It may manipulate a complete (or incomplete) single solution or a collection of solutions at each iteration. The subordinate heuristics may be high (or low) level procedures, or a simple local search, or just a construction method.

According to Glover [2],

...these methods have over time also come to include any procedure for problem solving that employs a strategy for overcoming the trap of

^{*} This work is supported by the European RTN ADONET under grant 504438.

local optimality in complex solution spaces, especially those procedures that utilize one or more neighborhood structures as a means of defining admissible moves to transition from one solution to another, or to build or destroy solutions in constructive and destructive processes.

Simulated annealing, tabu search, evolutionary algorithms like genetic algorithms and evolution strategies, ant colony optimization, estimation of distribution algorithms, scatter search, path relinking, the greedy randomized adaptive search procedure (GRASP), multi-start and iterated local search, guided local search, and variable neighborhood search are – among others – often listed as examples of classical metaheuristics, and they have individual historical backgrounds and follow different paradigms and philosophies; see e.g. [2].

Especially over the last years a large number of algorithms were reported that do not purely follow the concepts of one single traditional metaheuristic, but they combine various algorithmic ideas, sometimes also from outside of the traditional metaheuristics field. These approaches are commonly referred to as *hybrid metaheuristics*.

As for metaheuristics in general, there exist various perceptions of what a hybrid metaheuristic actually is. Looking up the meaning of *hybrid* in the current issue (May 2006) of the Merriam Webster dictionary yields

- a) something heterogeneous in origin or composition,
- b) something (as a power plant, vehicle, or electronic circuit) that has two different types of components performing essentially the same function,

while the current entry in Wiktionary defines this term as

- a) offspring resulting from cross-breeding different entities, e.g. different species,
- b) something of mixed origin or composition.

The motivation behind such hybridizations of different algorithmic concepts is usually to obtain better performing systems that exploit and unite advantages of the individual pure strategies, i.e. such hybrids are believed to benefit from synergy. The vastly increasing number of reported applications of hybrid metaheuristics and dedicated scientific events such as the series of *Workshops on Hybrid Metaheuristics* [6,7] document the popularity, success, and importance of this specific line of research. In fact, today it seems that choosing an adequate hybrid approach is determinant for achieving top performance in solving most difficult problems.

Actually, the idea of hybridizing metaheuristics is not new but dates back to the origins of metaheuristics themselves. At the beginning, however, such hybrids were not so popular since several relatively strongly separated and even competing communities of researchers existed who considered “their” favorite class of metaheuristics “generally best” and followed the specific philosophies in very dogmatic ways. For example, the evolutionary computation community

grew up in relative isolation and followed relatively strictly the biologically oriented thinking. It is mostly due to the no free lunch theorems [8] that this situation fortunately changed and people recognized that there cannot exist a general optimization strategy which is globally better than any other. In fact, to solve a problem at hand most effectively, it almost always requires a specialized algorithm that needs to be compiled of adequate parts.

Several publications exist which give taxonomies for hybrid metaheuristics or particular subcategories [9,10,11,12,13,14]. The following section tries to merge the most important aspects of these classifications and at some points extends these views. Also, several examples of common hybridization strategies are given. In Section 3, we turn to a unified view on metaheuristics by discussing the pool template. It helps to extract the specific characteristics of the individual classical metaheuristics and to interpret them as a toolbox of key components that can be combined in flexible ways to build an effective composite system. Section 4 refers to a selection of highly promising possibilities for combining metaheuristics with algorithms from two other prominent research fields in combinatorial optimization, namely constraint programming and integer linear programming. Finally, conclusions are drawn in Section 5.

2 Classification of Hybrid Metaheuristics

Figure 1 illustrates the various classes and properties by which we want to categorize hybrids of metaheuristics. Hereby, we combine aspects from the taxonomy introduced by Talbi [10] with the points-of-view from Cotta [9] and Blum et al. [11]. Classifications with particular respect to parallel metaheuristics are partly adopted from El-Abd and Kamel [14] and Cotta et al. [12] and with respect to the hybridization of metaheuristics with exact optimization techniques from Puchinger and Raidl [13].

We start by distinguishing *what* we hybridize, i.e. which kind of algorithms. We might combine (a) different metaheuristic strategies, (b) metaheuristics with certain algorithms specific for the problem we are considering, such as special simulations, or (c) metaheuristics with other more general techniques coming from fields like operations research (OR) and artificial intelligence (AI). Prominent examples for optimization methods from other fields that have been successfully combined with metaheuristics are exact approaches like branch-and-bound, dynamic programming, and various specific integer linear programming techniques on one side and soft computation techniques like neural networks and fuzzy logic on the other side.

Beside this differentiation, previous taxonomies of hybrid metaheuristics [10,9] primarily distinguish the *level* (or strength) at which the different algorithms are combined: High-level combinations in principle retain the individual identities of the original algorithms and cooperate over a relatively well defined interface; there is no direct, strong relationship of the internal workings of the algorithms.

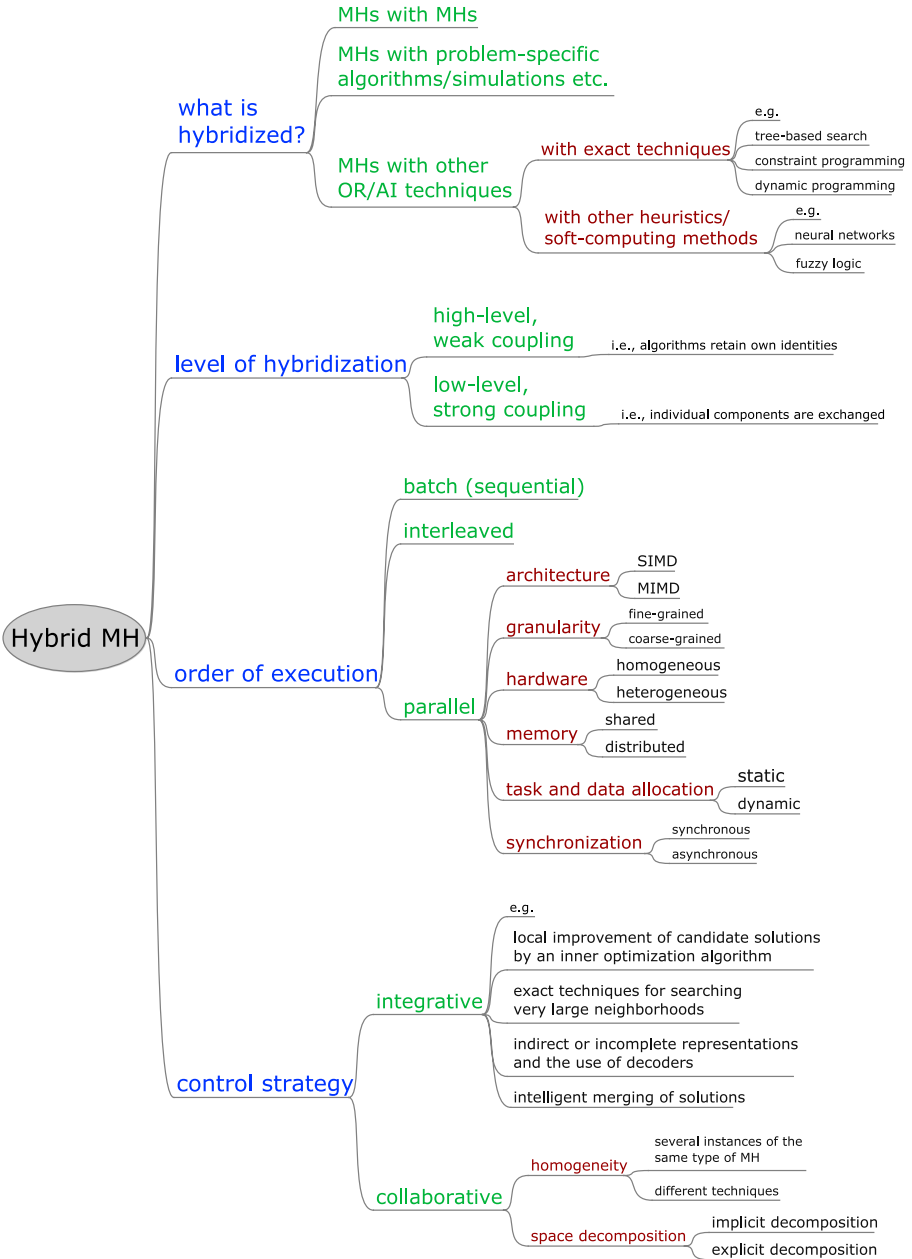


Fig. 1. A summarized classification of hybrid metaheuristics (MHs)

On the contrary, algorithms in low-level combinations strongly depend on each other – individual components or functions of the algorithms are exchanged.

Another property by which we may distinguish hybrid systems is the *order of execution*. In the batch model, one algorithm is strictly performed after the other, and information is passed only in one direction. An intelligent preprocessing of input data or a postprocessing of the results from another algorithm would fall into this category. Another example are multi-level problems which are solved by considering one level after the other by dedicated optimization algorithms. On the contrary, we have the interleaved and parallel models, in which the algorithms might interact in more sophisticated ways. Parallel metaheuristics are nowadays a large and important research field for their own, see [15]. Detailed classifications of hybrid parallel metaheuristics can be found in [14,12]. Following general characterizations of parallel algorithms, we can distinguish the architecture (SIMD: single instruction, multiple data streams versus MIMD: multiple instruction, multiple data streams), the granularity of parallelism (fine- versus coarse-grained), the hardware (homogeneous versus heterogeneous), the memory strategy (shared versus distributed memory), the task and data allocation strategy (static versus dynamic), and whether the different tasks are synchronized or run in an asynchronous way.

We can further distinguish hybrid metaheuristics according to their *control strategy*. Following [9,13], there exist integrative (coercive) and collaborative (co-operative) combinations.

In integrative approaches, one algorithm is considered a subordinate, embedded component of another algorithm. This approach is extremely popular.

- For example, in memetic algorithms [16], various kinds of local search are embedded in an evolutionary algorithm for locally improving candidate solutions obtained from variation operators.
- *Very large scale neighborhood search* (VLSN) approaches are another example [17]. They utilize certain exact techniques such as dynamic programming to efficiently find best solutions in specifically designed large neighborhoods within a local search based metaheuristic.
- Also, any decoder-based metaheuristic, in which a master algorithm acts on an implicit or incomplete representation of candidate solutions and a decoder is used to obtain corresponding actual solutions, falls into this category. Such a decoder can be virtually any kind of algorithm ranging from a simple problem specific heuristic to sophisticated exact optimization techniques or other OR/AI methods. For example in the cutting and packing domain, a common approach is to represent a candidate solution as a permutation of the items that need to be cut out or packed, and an actual solution is derived by considering the items in more or less sophisticated assignment heuristics in the given order, see e.g. [18]. Weight-coding [19] and problem space search [20] are further examples of indirect, relatively generally applicable representations based on decoders.
- Merging solutions: In population based methods such as evolutionary algorithms or scatter search, a traditional variation operator is recombination.

It derives a new solution by combining features of two (or more) parent solutions. Especially in classical genetic algorithms, this operator is based on pure random decisions and therefore works without exploiting any problem specific knowledge. Occasionally, this procedure is replaced by more powerful algorithms like path-relinking [21] or by exact techniques based on branch-and-bound or integer linear programming that identify a best combination of parental features, see e.g. [22,23].

In collaborative combinations, algorithms exchange information, but are not part of each other. For example, the popular island model [24] for parallelizing evolutionary algorithms falls into this category. We can further classify the traditional island model as a homogeneous approach since several instances of the same metaheuristic are performed. In contrast, Talukdar et al. [25,26] suggested a heterogeneous framework called *asynchronous teams* (A-Teams). An A-Team is a problem solving architecture consisting of a collection of agents and memories connected into a strongly cyclic directed network. Each of these agents is an optimization algorithm and can work on the target problem, on a relaxation of it, i.e. a superclass, or on a subclass. The basic idea of A-Teams is having these agents work asynchronously and autonomously on a set of shared memories. Denzinger and Offermann [27] presented a similar multi-agent based approach for achieving cooperation between search-systems with different search paradigms, such as evolutionary algorithms and branch-and-bound.

In particular in collaborative combinations, a further question is which search spaces are actually explored by the individual algorithms. According to [14] we can distinguish between an implicit decomposition resulting from different initial solutions, different parameter values etc., and an explicit decomposition in which each algorithm works on an explicitly defined subspace. Effectively decomposing large problems is in practice often an issue of crucial importance. Occasionally, problems can be decomposed in very natural ways, but in most cases finding an ideal decomposition into relatively independent parts is difficult. Therefore, (self-)adaptive schemes are sometimes also used.

3 A Unified View on Hybrid Metaheuristics

The success of all these hybrid metaheuristics tells us that it is usually a bad idea to approach a given (combinatorial) optimization problem with a view that is too restricted to a small (sub-)class of metaheuristics, at least when the primary goal is to solve the problem as well as possible. There is nothing to say against the analogy to real-world phenomena, by which several metaheuristics are explained with or even derived from, for example evolutionary algorithms, ant colony optimization, or simulated annealing. However, one should avoid to focus too strongly on such philosophies, hereby losing the view on particular strengths and benefits of other algorithmic concepts.

Instead of perceiving the various well-known metaheuristics as relatively independent optimization frameworks and occasionally considering hybridization

Algorithm Pool Template

```

Initialize pool  $P$  by an external procedure;
while termination=FALSE do
   $S \leftarrow OF(P)$ ;
  if  $|S| > 1$  then
     $S' \leftarrow SCM(S)$ 
  else
     $S' \leftarrow S$ ;
   $S'' \leftarrow IM(S')$ ;
   $P \leftarrow IF(S'')$ ;
Apply a post-optimizing procedure to  $P$ .

```

Fig. 2. The pool template from Voß [30,31]. P : Pool; IF/OF : Input/Output Function; IM : Improvement Method; SCM : Solution Combination Method.

for achieving certain benefits, it might be advantageous to change the point-of-view towards a unified design concept. All the existing metaheuristics share some ideas and differ among each other by certain characteristic *key components*. Making these key components explicit and collecting them yields a *toolbox* of components from which we can choose in the design of an optimization algorithm as it seems to be most appropriate for the target problem at hand.

In fact, this unified point-of-view is not new. Vaessens et al. [28] already presented a template for representing various kinds of local search based approaches, in particular threshold algorithms, tabu search, variable depth search, and even population based methods such as genetic algorithms. They also addressed multi-level approaches such as genetic local search, where a local search algorithm is applied within a genetic algorithm.

Calégary et al. [29] provided a taxonomy and united view on evolutionary algorithms and exemplarily discussed them with genetic algorithms, ant colony optimization, scatter search, and an emergent colonization algorithm.

Greistorfer and Voß [30,31] introduced a pool template by which they intend to cover even more different classes of metaheuristics, but especially also population based approaches. It is shown in Figure 2 and follows the definition of metaheuristics as given by Voß in [5] and cited in Section 1. To interpret, for example, simulated annealing in terms of this template, we set $|S| = 1$ and $|P| = 2$. The latter choice seems to be unusual at first glance. However, it covers the fact that we always have a current solution in the pool for which one or more neighbors are evaluated and additionally store the overall so-far best solution. The output function OF always simply returns the current solution. The improvement method IM includes the random choice of a neighboring solution and its evaluation, while the input function IF finally applies the Metropolis criterion (or some other condition) in order to either accept the new solution or to retain the previous one. The temperature update can also be considered to be part of the input function. Obviously, also other derivatives of local search like tabu search, guided local search, iterated local search, variable neighborhood descent/search, but also population-based approaches such as genetic algorithms,

evolution strategies, scatter search, and particle swarm optimization can be interpreted as instances of this template in straight-forward ways. Multi-level algorithms like memetic algorithms, where some local search procedure is applied to created candidate solutions are also supported via the improvement method *IM* which might include complete other optimization procedures. The template even matches estimation of distribution algorithms such as an ant colony optimization: The pheromone matrix – or more generally the statistical model – is considered as an additional memory structure, the output function covers the derivation of new solution candidates in dependence of this additional memory, and the input function also includes the memory’s update. Examples for solution combination methods (*SCM*) are the classical recombination techniques from genetic algorithms, path relinking, and the merging approaches addressed in the previous section.

Interpreting metaheuristics as instances of such a common template yields a decomposition of the algorithms. In case of the pool template, we obtain individual input and output functions, improvement methods, and eventually solution combination methods. Some of these parts may use auxiliary functions and data structures. From the perspective of functionality, a subset of these parts obtained from the decomposition of the different metaheuristics represents the algorithms’ key components that have been pointed out before. They can be considered to form a joined toolbox from where we can select the most promising parts and combine them in order to build effective (hybrid) optimization approaches tailored to the specific characteristics of the problems at hand. Table 1 summarizes important key components provided by popular metaheuristics.

Some software libraries for metaheuristics, such as HotFrame [32] and EALib [33], partly support this way of thinking by their object oriented structure and allow flexible combinations of key components when implementing problem-specific optimization algorithms; see also [34].

4 Some Promising Hybridization Possibilities with Other Prominent Combinatorial Optimization Techniques

The previous section mainly addressed low-level hybrids between different types of metaheuristics. Most existing hybrid metaheuristics probably fall into this category. To some degree, the described point-of-view can also be extended towards hybrids of metaheuristics with other OR and AI techniques. In fact, it is often a bad idea to prematurely restrict the possible choices in the design of an optimization algorithm too early to metaheuristic techniques only.

In particular constraint programming (CP) and integer linear programming (ILP) shall be mentioned here as two research fields with long histories and also much success in solving difficult combinatorial optimization problems; see [35] and [36] for introductory textbooks to the two fields, respectively. As metaheuristics, the methods from these fields also have their specific advantages and limits. Especially in the last years, combinations between such techniques and metaheuristics have shown to be often extremely successful and promising.

Table 1. Some key components of popular metaheuristics

Ant colony optimization	<i>OF</i> : derivation of new solution candidates by considering a pheromone matrix; <i>SCM</i> : implicitly via pheromone matrix; <i>IF</i> : includes update of pheromone matrix
Genetic algorithms	<i>OF</i> , <i>IF</i> : selection techniques; <i>SCM</i> : crossover operators; <i>IM</i> : mutation operators, repair schemes, decoding functions
Guided local search	<i>IM</i> , <i>IF</i> : augmentation of evaluation function to escape local optima
GRASP	initialization, <i>OF</i> : creation of meaningful solutions from scratch by a randomized greedy heuristic
Iterated local search	<i>IM</i> : perturbation of a solution for diversification
Multi start approaches	initialization, <i>OF</i> : creation of (random) solutions from scratch for diversification
Path relinking	<i>SCM</i> : more sophisticated method for combining solutions
Pilot method	<i>IF</i> : more sophisticated evaluation and acceptance criterion
Scatter search	<i>IF</i> : diversification generation methods, subset generation methods; <i>IM</i> : improvement methods; <i>SCM</i> : solution combination methods; <i>OF</i> : reference set update methods
Simulated annealing	<i>IF</i> : acceptance criterion, annealing schedule
Tabu search	<i>IM</i> , <i>IF</i> : consideration and maintenance of tabu list, aspiration criteria
Variable depth search	<i>IM</i> : search of a more sophisticated neighborhood
Variable neighborhood descent	<i>IM</i> : search of multiple neighborhoods
Variable neighborhood search	<i>IM</i> : shaking in different neighborhoods for diversification
Very large neighborhood search	<i>IM</i> : efficient search of a large neighborhood

An overview on hybrids of local search based approaches and constraint programming is given in [37]. Basic concepts include:

- CP can be used as preprocessing for reducing the search space.
- CP techniques can be used to more efficiently search neighborhoods, especially under the existence of difficult problem-specific constraints.
- Special large neighborhoods can sometimes be defined by introducing appropriate artificial constraints, and CP is again used for efficiently searching these neighborhoods.
- In constructive heuristics like GRASP or ant colony optimization, CP can be utilized to make better choices in the selection of the next solution component to be added.

An overview on promising combinations of metaheuristics and integer linear programming (ILP) techniques is given in [13]. Basic concepts include:

- Solving a linear programming or Lagrangian relaxation of the problem often yields valuable information that can be effectively exploited in construction heuristics or variation operators.
- As CP, also ILP has been used to search large neighborhoods.
- ILP can be used for merging solutions.
- Exact ILP techniques are often based on tree search, and good upper and lower bounds are of crucial importance. While for a minimization problem lower bounds are obtained via relaxations, heuristics are important for obtaining upper bounds. Metaheuristics can here be very beneficial.
- In cutting plane techniques such as branch-and-cut inequalities need to be identified which are violated by the current solution to the linear programming (LP) relaxation, but which are valid for the integer optimum. These inequalities are then added to the system and the LP is resolved, yielding an improved bound. The identification of such violated inequalities often is a hard problem for its own, which can be approached by metaheuristics.
- Column generation techniques such as branch-and-price start with a small, restricted set of variables. When having solved this reduced problem, variables being part of the model but currently not included are identified whose insertion enable a further improvement of the current solution; the whole process is repeated. The task of finding such variables is often difficult and metaheuristics have been successfully used for solving it [38].
- Last but not least, some promising concepts such as local branching [39] exist which bring the idea of local search based metaheuristics into linear programming based branch-and-bound: Specific neighborhood-defining inequalities are added to subproblems and branching is controlled in order to perform a “virtual” metaheuristic within tree search.

5 Conclusions

Manifold possibilities of hybridizing individual metaheuristics with each other and/or with algorithms from other fields exist. A large number of publications documents the great success and benefits of such hybrids. Based on several previously suggested taxonomies, a unified classification and characterization of meaningful hybridization approaches has been presented. Especially with respect to low-level hybrids of different metaheuristics, a unified view based on a common pool template can be advantageous. It helps in making different key components of existing metaheuristics explicit. We can then consider these key components as a toolbox and build an effective (hybrid) metaheuristic for a problem at hand by selecting and combining the most appropriate components. This approach of thinking seems to be superior to sticking too strongly to the philosophies and historical backgrounds behind the different metaheuristic paradigms. Finally, particularly promising possibilities of combining metaheuristics with constraint programming and integer programming techniques were pointed out.

References

1. Blum, C., Roli, A.: Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys* **35**(3) (2003) 268–308
2. Glover, F., Kochenberger, G.A.: *Handbook of Metaheuristics*. Kluwer (2003)
3. Hoos, H.H., Stützle, T.: *Stochastic Local Search*. Morgan Kaufmann (2005)
4. Glover, F.: Future paths for integer programming and links to artificial intelligence. *Decision Sciences* **8** (1977) 156–166
5. Voß S., Martello, S., Osman, I.H., Roucairo, C.: *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Kluwer, Boston (1999)
6. Blum, C., Roli, A., Sampels, M., eds.: *Proceedings of the First International Workshop on Hybrid Metaheuristics*, Valencia, Spain (2004)
7. Blesa, M.J., Blum, C., Roli, A., Sampels, M., eds.: *Hybrid Metaheuristics: Second International Workshop*. Volume 3636 of LNCS. (2005)
8. Wolpert, D., Macready, W.: No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation* **1**(1) (1997) 67–82
9. Cotta, C.: A study of hybridisation techniques and their application to the design of evolutionary algorithms. *AI Communications* **11**(3–4) (1998) 223–224
10. Talbi, E.G.: A taxonomy of hybrid metaheuristics. *Journal of Heuristics* **8**(5) (2002) 541–565
11. Blum, C., Roli, A., Alba, E.: An introduction to metaheuristic techniques. In: *Parallel Metaheuristics, a New Class of Algorithms*. John Wiley (2005) 3–42
12. Cotta, C., Talbi, E.G., Alba, E.: Parallel hybrid metaheuristics. In Alba, E., ed.: *Parallel Metaheuristics, a New Class of Algorithms*. John Wiley (2005) 347–370
13. Puchinger, J., Raidl, G.R.: Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In: *Proceedings of the First International Work-Conference on the Interplay Between Natural and Artificial Computation, Part II*. Volume 3562 of LNCS., Springer (2005) 41–53
14. El-Abd, M., Kamel, M.: A taxonomy of cooperative search algorithms. In Blesa, M.J., Blum, C., Roli, A., Sampels, M., eds.: *Hybrid Metaheuristics: Second International Workshop*. Volume 3636 of LNCS., Springer (2005) 32–41
15. Alba, E., ed.: *Parallel Metaheuristics, a New Class of Algorithms*. John Wiley, New Jersey (2005)
16. Moscato, P.: Memetic algorithms: A short introduction. In Corne, D., et al., eds.: *New Ideas in Optimization*. McGraw Hill (1999) 219–234
17. Ahuja, R.K., Ergun, Ö., Orlin, J.B., Punnen, A.P.: A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics* **123**(1–3) (2002) 75–102
18. Puchinger, J., Raidl, G.R.: Models and algorithms for three-stage two-dimensional bin packing. *European Journal of Operational Research*, Feature Issue on Cutting and Packing (to appear 2006)
19. Julstrom, B.A.: Strings of weights as chromosomes in genetic algorithms for combinatorial problems. In Alander, J.T., ed.: *Proceedings of the Third Nordic Workshop on Genetic Algorithms and their Applications*. (1997) 33–48
20. Storer, R.H., Wu, S.D., Vaccari, R.: New search spaces for sequencing problems with application to job-shop scheduling. *Management Science* **38** (1992) 1495–1509
21. Glover, F., Laguna, M., Martí, R.: Fundamentals of scatter search and path re-linking. *Control and Cybernetics* **39**(3) (2000) 653–684
22. Applegate, D., Bixby, R., Chvátal, V., Cook, W.: On the solution of the traveling salesman problem. *Documenta Mathematica* **Vol. ICM III** (1998) 645–656

23. Cotta, C., Troya, J.M.: Embedding branch and bound within evolutionary algorithms. *Applied Intelligence* **18** (2003) 137–153
24. Goldberg, D.E.: *Genetic Algorithms in Search, Optimization, and Learning*. Addison-Wesley, Reading, MA (1989)
25. Talukdar, S., Baeretzen, L., Gove, A., de Souza, P.: Asynchronous teams: Cooperation schemes for autonomous agents. *Journal of Heuristics* **4** (1998) 295–321
26. Talukdar, S., Murty, S., Akkiraju, R.: Asynchronous teams. In: *Handbook of Metaheuristics*. Volume 57. Kluwer Academic Publishers (2003) 537–556
27. Denzinger, J., Offermann, T.: On cooperation between evolutionary algorithms and other search paradigms. In: *Proceedings of the Congress on Evolutionary Computation 1999*, IEEE Press (1999)
28. Vaessens, R., Aarts, E., Lenstra, J.: A local search template. In Manner, R., Manderick, B., eds.: *Parallel Problem Solving from Nature*, Elsevier (1992) 67–76
29. Calégari, P., Coray, G., Hertz, A., Kobler, D., Kuonen, P.: A taxonomy of evolutionary algorithms in combinatorial optimization. *Journal of Heuristics* **5**(2) (1999) 145–158
30. Greistorfer, P., Voß, S.: Controlled pool maintenance in combinatorial optimization. In Rego, C., Alidaee, B., eds.: *Metaheuristic Optimization via Memory and Evolution – Tabu Search and Scatter Search*. Volume 30 of *Operations Research/Computer Science Interfaces*. Springer (2005) 382–424
31. Voß, S.: Hybridizing metaheuristics: The road to success in problem solving (2006) Slides of an invited talk at the EvoCOP 2006, the 6th European Conference on Evolutionary Computation in Combinatorial Optimization, Budapest, Hungary, <http://www.ads.tuwien.ac.at/evocop/Media/Invited-talk-EvoCOP2006-voss.pdf>.
32. Fink, A., Voß, S.: HotFrame: A heuristic optimization framework. In Voß S., Woodruff, D.L., eds.: *Optimization Software Class Libraries*. OR/CS Interfaces Series. Kluwer Academic Publishers (1999)
33. Wagner, D.: Eine generische Bibliothek für Metaheuristiken und ihre Anwendung auf das Quadratic Assignment Problem. Master's thesis, Vienna University of Technology, Institute of Computer Graphics and Algorithms (2005)
34. Voß S., Woodruff, D.L., eds.: *Optimization Software Class Libraries*. OR/CS Interfaces Series. Kluwer Academic Publishers (2002)
35. Marriott, K., Stuckey, P.: *Programming with Constraints*. The MIT Press (1998)
36. Nemhauser, G., Wolsey, L.: *Integer and Combinatorial Optimization*. John Wiley (1988)
37. Focacci, F., Laburthe, F., Lodi, A.: Local search and constraint programming. In: *Handbook of Metaheuristics*. Volume 57. Kluwer Academic Publishers (2003) 369–403
38. Puchinger, J., Raidl, G.R., Gruber, M.: Cooperating memetic and branch-and-cut algorithms for solving the multidimensional knapsack problem. In: *Proceedings of MIC2005, the 6th Metaheuristics International Conference*, Vienna, Austria (2005) 775–780
39. Fischetti, M., Lodi, A.: Local Branching. *Mathematical Programming Series B* **98** (2003) 23–47

Packing Problems with Soft Rectangles

Toshihide Ibaraki and Kouji Nakamura

Kwansei Gakuin University, Sanda, Japan 669-1337

ibarak@ksc.kwansei.ac.jp

<http://ist.ksc.kwansei.ac.jp/~ibarak/>

Abstract. We consider the problems of packing rectangles, whose shapes are adjustable within given perimeter and area constraints. Using “sequence pairs” to specify relative positions of rectangles, we solve the resulting linear or convex programming problems to determine sizes and locations of all rectangles. To find good sequence pairs, we then resort to local search techniques. This is therefore a hybrid of local search and mathematical programming. The resulting algorithm can solve problem instances with up to 50 rectangles in reasonable amount of time.

1 Introduction

Packing a given set of rectangles into a small region without overlap is one of the representative problems in combinatorial optimization. In particular, when all rectangles have given sizes, there is a long list of literature, e.g., [6,7,9,10,13], many of which are based on metaheuristic algorithms. As a result of these studies, problem instances of fairly large sizes, containing several hundred rectangles, can be solved, in the sense that very good solutions can be found in reasonable amount of time.

In this paper, we generalize the problem by assuming that each rectangle is soft. Namely, the width w_i and the height h_i of rectangle i can be adjusted within given constraints. For example, the constraints may specify their lower and upper bounds:

$$\begin{aligned}w_i^L &\leq w_i \leq w_i^U, \\h_i^L &\leq h_i \leq h_i^U,\end{aligned}\tag{1}$$

where w_i^L and w_i^U (resp., h_i^L and h_i^U) are given lower and upper bounds on w_i (resp., h_i). We may also add the constraint that the *aspect ratio* h_i/w_i is bounded between its lower bound r_i^L and upper bound r_i^U :

$$r_i^L w_i \leq h_i \leq r_i^U w_i.\tag{2}$$

Another type of constraint common in applications is that each rectangle i must have either a given perimeter L_i or a given area A_i (or both):

$$w_i + h_i \geq L_i,\tag{3}$$

$$w_i h_i \geq A_i.\tag{4}$$

In addition, we may consider that the locations of rectangles are pre-determined in some intervals:

$$\begin{aligned} x_i^L &\leq x_i \leq x_i^U, \\ y_i^L &\leq y_i \leq y_i^U, \end{aligned} \tag{5}$$

where x_i (resp., y_i) denotes the x -coordinate (resp., y -coordinate) of the lower left corner of rectangle i .

To our knowledge, there is not much literature on this type of problems using soft rectangles, except such papers as [2,5,11,15], even though it has various applications.

Applications can be found, for example, in VLSI floorplan design [2,5,10,11,15] and in resource constrained scheduling. In the VLSI design, each rectangle represents a block of logic circuits consisting of a certain number of transistors, which occupy certain area, and must have at least some perimeter length to accommodate connection lines to other blocks. The shape of each rectangle is adjustable, but required to satisfy the constraints as stated above. In a scheduling application, each rectangle may represent a job, to be assigned to an appropriate position on the time axis (horizontal), where its width gives the processing time of the job and its height represents the amount of resource (per unit time) invested to process the job. In this case, the area of the rectangle represents the total amount of resource consumed by the job, which is again required to satisfy the above constraints.

Our approach utilizes a standard tool known as “sequence pair” [10] to specify relative position between each pair of rectangles (e.g., rectangle i is placed to the left of rectangle j , i is above j and so forth) to obtain a packing without overlap. Once a sequence pair is given, the remaining problem is to determine exact sizes and locations of all rectangles (i.e., a *packing*). We show that this problem can be formulated as a linear programming problem or nonlinear programming problem (more exactly, convex programming problem) depending on the constraints and objective functions. Thanks to recent progress of mathematical programming techniques, such problems can be solved very efficiently in the practical sense. To find good sequence pairs that yield packings of high quality, we then resort to local search techniques. We test various types of neighborhoods, and recommend some combination of neighborhoods. Although our computational experiment is rather preliminary, we could show that problem instances with $30 \sim 50$ rectangles are readily solvable. Of course, we would encounter problem instances of much larger sizes in real applications, and further improvement will be necessary to meet such demands. One comment here is that, in many applications, problems can be decomposed into subproblems of manageable sizes, and our algorithms may prove useful to solve such subproblems. Also, with further elaborations such as using sophisticated metaheuristic ideas and enhancing the computation of mathematical programming part, we believe it not difficult to improve the performance of our algorithm to a great extent.

2 Sequence Pairs and Problem Statement

2.1 Sequence Pairs

We first explain the idea of a sequence pair, which was introduced by [10], and then define our packing problems. A *sequence pair* is a pair of permutations $\sigma = (\sigma_+, \sigma_-)$ on $I = \{1, 2, \dots, n\}$, where $\sigma_+(l) = i$ (equivalently $\sigma_+^{-1}(i) = l$) means that rectangle i is the l th rectangle in σ_+ . The σ_- is similarly defined. In a non-overlapping packing, every pair of rectangles i and j must satisfy at least one of the four conditions: (1) i is to the left of j , (2) j is to the left of i , (3) i is above j , and (4) j is above i . A sequence pair $\sigma = (\sigma_+, \sigma_-)$ specifies which of the four conditions holds, using the partial orders \preceq_σ^x and \preceq_σ^y defined by

$$\begin{aligned} \sigma_+^{-1}(i) \leq \sigma_+^{-1}(j) \text{ and } \sigma_-^{-1}(i) \leq \sigma_-^{-1}(j) &\iff i \preceq_\sigma^x j, \\ \sigma_+^{-1}(i) \geq \sigma_+^{-1}(j) \text{ and } \sigma_-^{-1}(i) \leq \sigma_-^{-1}(j) &\iff i \preceq_\sigma^y j, \end{aligned}$$

for any pair i and j of rectangles. This says that, if i appears before j in both σ_+ and σ_- , then $i \preceq_\sigma^x j$, i.e., i is located to the left of j in the packing, while if i appears after j in σ_+ , but before j in σ_- , then $i \preceq_\sigma^y j$, i.e., i is located under j in the packing. Since exactly one of $i \preceq_\sigma^x j$, $j \preceq_\sigma^x i$, $i \preceq_\sigma^y j$, $j \preceq_\sigma^y i$ always holds for a given pair of i and j , we see that a sequence pair σ imposes exactly one of the above four conditions on relative positions. These constraints on relative positions can be described by the following inequalities.

$$\begin{aligned} x_i + w_i &\leq x_j & \text{if } i \preceq_\sigma^x j, \\ x_j + w_j &\leq x_i & \text{if } j \preceq_\sigma^x i, \\ y_i + h_i &\leq y_j & \text{if } i \preceq_\sigma^y j, \\ y_j + h_j &\leq y_i & \text{if } j \preceq_\sigma^y i. \end{aligned} \tag{6}$$

2.2 Various Problems with Soft Rectangles

As the objective function of our problem, we may choose to minimize the perimeter of the rectangular region that contains all the given rectangles (we call this the *container*), i.e.,

$$\text{minimize } W + H \tag{7}$$

or to minimize its area,

$$\text{minimize } WH, \tag{8}$$

where W and H are the variables that satisfy

$$\begin{aligned} x_i + w_i &\leq W & \text{for all } i, \\ y_i + h_i &\leq H & \text{for all } i. \end{aligned} \tag{9}$$

In a variation called the *strip packing problem*, the height of the container is fixed to $H = H^*$, where H^* is a given constant, and its width W is minimized:

$$\text{minimize } W, \quad (10)$$

under the constraint

$$\begin{aligned} x_i + w_i &\leq W && \text{for all } i, \\ y_i + h_i &\leq H^* && \text{for all } i. \end{aligned} \quad (11)$$

Therefore, if a sequence pair σ is specified, we are required to solve the mathematical programming problem $P(\sigma)$ to minimize objective function (7), (8) or (10) under the constraints:

$$\begin{aligned} &(1), (2), (3) \text{ (and/or (4)), (5), (9) (or (11)) for all } i, \\ &(6) && \text{for all } i \text{ and } j, \\ &x_i, y_i \geq 0 && \text{for all } i. \end{aligned} \quad (12)$$

Here we assume that all bounds $w_i^L, w_i^U, h_i^L, h_i^U, r_i^L, r_i^U, L_i, A_i, x_i^L, x_i^U, y_i^L, y_i^U, H^*$ are nonnegative, and we restrict the locations of all rectangles to the first quadrant.

To reduce the data size of problem description, we can remove the redundant constraints (i.e., those derivable by transitivity law) from (6); e.g., if there is a k satisfying $i \preceq_\sigma^x k \preceq_\sigma^x j$, then the constraint $i \preceq_\sigma^x j$ between i and j is redundant. It is not difficult to carry out this task in $O(n^2)$ computation time, where n is the number of rectangles.

It is important to note that the feasible region defined by constraints (12) is in general nonlinear but is *convex*, where that of the constraints (1), (2), (3) and (4) on w_i and h_i is illustrated in Figure 1. Therefore, if the objective function is convex, we obtain a convex programming problem, and can solve it by existing efficient algorithms (e.g., [1,14]). This is the case when we want to minimize (7) or (10). The problem with objective function (8) is not a convex programming problem, but is a so-called multiplicative programming problem for which some efficient approaches are also known (e.g., [8]).

2.3 Two Test Problems

From the above varieties, we choose two simple problems for our experiment, emphasizing perimeter and area constraints of rectangles, respectively.

The first type minimizes the perimeter of the container under the perimeter constraints (3) of rectangles.

$$\begin{aligned} P_{\text{peri}}(\sigma) : & \text{minimize } W + H \\ & \text{subject to } (1), (3), (9) && \text{for all } i \\ & (6) && \text{for all } i \text{ and } j \\ & x_i, y_i \geq 0 && \text{for all } i. \end{aligned} \quad (13)$$

This gives rise to a linear programming problem for each given sequence pair σ . We call this the *perimeter minimization problem*.

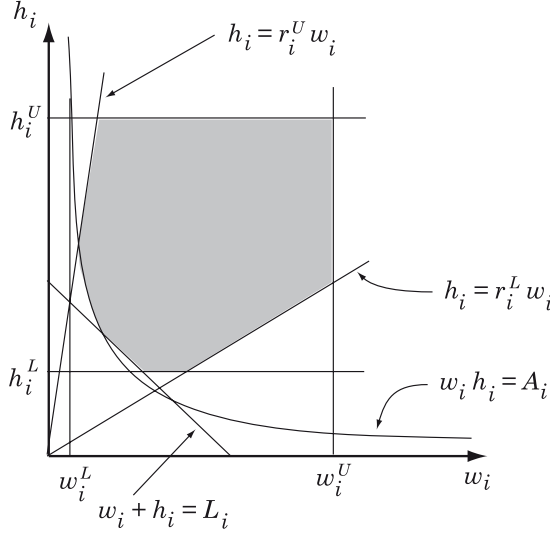


Fig. 1. Feasible region for the w_i and h_i of rectangle i

The second type is the strip packing problem under the area constraints (4) of rectangles, which is formulated as a convex programming problem.

$$\begin{aligned}
 P_{\text{area}}(\sigma) : & \text{minimize } W + \alpha v \\
 & \text{subject to } (1), (4) && \text{for all } i \\
 & x_i + w_i \leq W && \text{for all } i \\
 & y_i + h_i \leq H^* + v && \text{for all } i \\
 & (6) && \text{for all } i \text{ and } j \\
 & v \geq 0 \\
 & x_i, y_i \geq 0 && \text{for all } i.
 \end{aligned} \tag{14}$$

Here the variable v is introduced to keep the problem feasible, by adding penalty term αv to the objective function (10) with a large positive constant α . We call this the *area minimization problem*.

3 General Framework of the Algorithm

The framework of our algorithm is local search (see e.g., [3,4]), which tries to find sequence pairs σ such that the associated mathematical programming problems $P(\sigma)$ offer good packing solutions. We denote the optimal solution of $P(\sigma)$ as $X(\sigma)$ and its objective value as $z(\sigma)$. A local search algorithm is in general defined by the *neighborhood* $N(\sigma)$ (a subset of the set of all sequence pairs) of the current σ , and is described in the following manner.

Algorithm. LOCAL_SEARCH

Input: Data for the constraints (12), and objective function (7), (8) or (10).

Output: A packing that satisfies constraints (12) and has a good objective value.

Step 1 (initialization): Construct an initial sequence pair σ (randomly or by a simple heuristic algorithm). Solve $P(\sigma)$ and let $X := X(\sigma)$ and $z := z(\sigma)$, where X and z denote the incumbent solution and its value, respectively.

Step 2 (local search): Repeat the following procedure until all sequence pairs in $N(\sigma)$ have been tested.

Select a new $\sigma' \in N(\sigma)$ and solve $P(\sigma')$. If $z(\sigma') < z$ holds, then let $X := X(\sigma')$, $z := z(\sigma')$, $\sigma := \sigma'$ and return to Step 2.

If there is no new sequence pair left in $N(\sigma)$, go to Step 3.

Step 3 (termination): Output X , and halt.

The output solution is *locally optimal* in terms of neighborhood $N(\sigma)$. Its performance critically depends on how to define the neighborhood $N(\sigma)$. In the subsequent sections, we investigate various neighborhoods and proposes a combination of some neighborhoods as a reasonable candidate for our implementation.

4 Constructing Effective Neighborhoods

4.1 Standard Neighborhoods

A neighborhood is a set of sequence pairs obtained from the current σ by applying certain local operations. Typical operations used for packing problems are *shift*, *swap* and *swap** [6,10], defined as follows.

1. Shift: This operation moves element j in σ_+ (or σ_-) to the next position of element i . The shift neighborhood is defined by applying this to all pairs of $i, j \in I$. If only one of σ_+ and σ_- is considered, it is the *single-shift neighborhood*, while if each shift operation of i and j is simultaneously applied to both σ_+ and σ_- , then it is the *double-shift neighborhood*. The sizes of these neighborhoods are $O(n^2)$ since we consider all pairs of $i, j \in I$.

2. Swap: This operation exchanges the positions of i and j in σ_+ (or in σ_-). The *single-swap neighborhood* and *double-swap neighborhood* are then defined in a similar manner to the case of shift neighborhood. The sizes of the resulting neighborhoods are $O(n^2)$.

3. Swap*: Let i and j in σ_+ have locations α and β , i.e., $\sigma_+(\alpha) = i$ and $\sigma_+(\beta) = j$, such that $\alpha < \beta$. Then for each γ with $\alpha \leq \gamma \leq \beta$ we move i and j to the location γ in the manner $\sigma'_+(\gamma) = j$ and $\sigma'_+(\gamma + 1) = i$, while keeping the same relative positions of other elements. This swap* operation can also be defined for σ_- in a symmetric manner. We usually apply swap* operations to only one of σ_+ and σ_- , yielding the *swap* neighborhood*. If we consider all combinations of i, j, γ , its size becomes $O(n^3)$.

The effects of these operations may be intuitively explained as follows, where we assume for simplicity that i is constrained to the left of j by σ . A shift

operation brings j immediately to the right of i , causing side effects on relative positions of j and other rectangles (which are different in cases of single-shift and double shift). A single-swap operation on σ_+ (resp., σ_-) changes the relative position of “ i is to the left of j ” to “ j is above i ” (resp., “ i is above j ”). On the other hand, a double-swap operation exchanges only the locations of i and j , without changing the relative positions of other rectangles. A swap* operation brings i and j together to their middle locations specified by γ .

As the sizes of these standard neighborhoods are large, it is not appropriate to use them directly in our local search, since for each candidate solution we have to solve a mathematical programming problem as discussed in Section 2.2, which is computationally rather expensive. Therefore, in the following, we consider how to reduce the neighborhood sizes without sacrificing its power of improvement. This direction has been studied in many papers on local search, but not in the context of packing soft rectangles.

4.2 Critical Paths

Given a solution $X(\sigma)$, we call a maximal sequence of rectangles i_1, i_2, \dots, i_k a (horizontal) *critical path*, if it satisfies the following condition: (i) i_j is constrained to the left of i_{j+1} by σ for all j , and (ii) $x_{i_j} + w_{i_j} = x_{i_{j+1}}$ holds for all j . A vertical critical path can be similarly defined. There may be more than one critical path horizontally and vertically. It is known that critical paths can be efficiently computed by using dynamic programming. To reduce the sizes of neighborhoods as defined in Section 4.1, it is often attempted (e.g., [6]) to restrict i to be in a critical path, while j can be any. We call the resulting neighborhoods like *single-swap critical neighborhood*, *swap* critical neighborhood* and so forth.

In our packing problems with soft rectangles, a packing solution $X(\sigma)$ tends to have many critical paths, since each rectangle is adjusted so that it directly touches horizontally adjacent rectangles (i.e., $x_{i_j} + w_{i_j} = x_{i_{j+1}}$) or vertically adjacent rectangles (i.e., $y_{i_j} + h_{i_j} = y_{i_{j+1}}$). As a result, in a solution $X(\sigma)$ that represents a good packing, most of the rectangles turn out to belong to some critical paths, implying that the restriction to critical paths is not very effective in reducing the neighborhood size. To remedy this to some extent, we define the *single-swap lower-bounding critical neighborhood* by restricting i to be in a critical path and to satisfy $w_i = w_i^L$ if the critical path is horizontal (or $h_i = h_i^L$ if vertical), since such a rectangle i cannot be shrunk any further. Similar argument applies also to other types of neighborhoods, resulting in the *single-shift lower-bounding critical neighborhood* and others.

4.3 Computational Comparison of Neighborhoods

To evaluate the power of the above various neighborhoods, we conducted preliminary computational experiment for problem instances of small sizes. Two instances are respectively constructed by taking the first 20 and 30 rectangles

in the benchmark called ami33¹, which represents a problem instance of VLSI floorplan.

Here we use the perimeter minimization problem in Section 2.3. Table 1 gives the results of algorithm LOCAL_SEARCH in Section 3 implemented with each of the following neighborhoods, abbreviated as

Sg-shift: single-shift neighborhood,
 Db-shift: double-shift neighborhood,
 Sg-swap: single-swap neighborhood,
 Db-swap: double-swap neighborhood,
 SgCr-shift: single-shift critical neighborhood (similarly for DbCr-shift,
 SgCr-swap, DbCr-swap),
 swap*: swap* neighborhood,
 SgLb-shift: single-shift lower-bounding critical neighborhood (similarly
 for SgLb-swap and Lb-swap*),
 SgAd-swap: single-swap adjacent lower-bounding critical neighborhood.

The algorithms were run from randomly generated initial solutions to local optimal solutions, where the rows in the table have the following meanings:

Time: CPU time in seconds for executing LOCAL_SEARCH,
 Density: Total area of all rectangles over the area of the container (%),
 W+H: Objective value (i.e., perimeter of the container).

All the data are the average values of five runs starting from independent random initial solutions.

Table 1. Comparison of neighborhoods

n		Sg-shift	Db-shift	Sg-swap	Db-swap	SgCr-shift	DbCr-shift
20	Time (secs)	195.2	171.4	74.2	104.4	86.1	21.4
	Density	95.9	95.1	94.7	94.5	93.7	77.0
	W+H	1617.9	1621.2	1643.3	1647.7	1613.5	1906.7
30	Time (secs)	684.7	781.5	574.3	788.7	477.7	86.6
	Density	94.6	94.7	95.7	93.2	95.2	78.1
	W+H	1967.7	1980.2	1960.2	1986.7	1944.0	2288.5

SgCr-swap	DvCr-swap	swap*	SgLb-shift	SgLb-swap	Lb-swap*	SgAd-swap
48.5	45.0	71.0	55.8	38.8	14.7	3.6
91.0	85.9	86.3	94.8	92.7	91.5	77.6
1688.9	1762.6	1844.5	1660.5	1672.5	1713.8	1943.2
381.9	316.1	94.6	325.0	236.9	197.4	7.5
93.4	88.6	93.4	85.1	94.2	90.3	76.9
1967.2	2116.6	2069.8	2117.7	1995.7	2059.4	2400.5

In Table 1, the last neighborhood SgAd-swap was not explained yet, and is based on the following observation. Among the first twelve neighborhoods,

¹ Available from http://www.cbl.ncsu.edu/CBL_Docs/lys90.html

SgLb-swap appears to be reasonably stable and gives good results in most cases. However, this still requires rather large computation time. To shorten its time, we further restrict rectangles i and j to be swapped to those which are lower bounding (i.e., $w_i = w_i^L$ or $h_i = h_i^L$ holds depending on the direction of the critical path) and are adjacent in some critical path. The resulting neighborhood is denoted as SgAd-swap. The quality of the solutions obtained by SgAd-swap is not good, but it consumes very little time compared with others.

4.4 Further Elaborations

To reduce the size of neighborhood further while maintaining high searching power, we added three more modifications.

The first idea is to look at a rectangle which belongs to both horizontal and vertical critical paths. We call such a rectangle as a *junction* rectangle. It is expected that removing a junction rectangle will break both the horizontal and vertical critical paths, and will have a large effect of changing the current packing. Thus we apply single-shift or double-swap operations to a junction rectangle i with any other rectangles j which are not junctions (in the case of double-swap we further restrict j to have a smaller area than i). We then apply these operations to all junction rectangles i . If an improvement is attained in this process, we immediately move to local search with SgLb-swap neighborhood for attaining further improvement. This cycle of “junction removals” and “local search with SgLb-swap” is repeated until no further improvement is attained. The resulting algorithms are denoted Jc(Sg-shift)+SgLb-swap or Jc(Db-swap)+SgLb-swap, respectively, depending on which operation is used to move the junction rectangle.

To improve the efficiency further, we then tried to replace the SgLb-swap neighborhood in the above iterations with SgAd-swap, which was defined at the end of the previous subsection. Using this neighborhood in place of SgLb-swap, we obtain algorithms Jc(Sg-shift)+SgAd-swap or Jc(Db-swap)+SgAd-swap.

Table 2 shows some computational results with three of these four algorithms, where Jc(Sg-shift)+SgAd-swap is omitted as it gives less effective results. All the numbers are the averages of five runs from independent random initial solutions. We observe that these three attain similar quality, but the last one Jc(Db-swap)+SgAd-swap consumes much less computation time than others. We also emphasize that the last one has much higher performance than SgLb-swap, which was considered to be the best in Table 1.

The last idea is to make use of vacant areas existing in a given packing. To find some of such vacant areas by a simple computation, we use the following property. Let the current sequence pair σ satisfy $i \preceq_\sigma^x j$ and there is no rectangle k such that $i \preceq_\sigma^x k \preceq_\sigma^k j$ (i.e., i is immediately to the left of j). In this case, if $x_i + w_i < x_j$ holds, there is some vacant area between i and j . We pick up the largest one among such vacant areas, in the sense of maximizing $x_j - (x_i + w_i)$. Let i^* and j^* be the resulting pair. Then we apply Sg-swap operations on σ^+ between those i and j such that $i \in \sigma^+$ is located in distance at most 5 from

Table 2. Comparison of neighborhoods using junction rectangles

n		Jc(Sg-shift) +SgLb-swap	Jc(Db-swap) +SgLb-swap	Jc(Db-swap) +SgAd-swap
20	Time	86.4	66.8	22.5
	Density	95.3	95.3	95.6
	W+H	1625.6	1658.0	1645.6
30	Time	588.4	393.5	82.2
	Density	94.8	97.5	96.7
	W+H	1945.9	1935.1	1993.4

i^* (forward or backward, i.e., $|\sigma_+^{-1}(i) - \sigma_+^{-1}(i^*)| \leq 5$), and $j \in \sigma^+$ is located in distance at most 5 from j^* (forward or backward).

This neighborhood is derived by horizontal argument. Analogous argument can also be applied vertically, and we consider the local search based on the resulting two types of neighborhoods, denoted SgVc-swap. Table 3 shows a result of the local search with SgVc-swap, which starts from a local optimal solution of instance ami33 obtained by local search with Jc(Db-swap)+SgAd-swap, where

Candidates: The number of $P(\sigma)$ solved in LOCAL_SEARCH,

Improvements: The number of improved solutions found in LOCAL_SEARCH,

and Density, W+H and Time were already defined with Table 1. As this appears to give further improvement without consuming much time, we decided to add this modification in all the subsequent experiments.

Table 3. Improvement by SgVc-swap (ami33)

	before SgVc-swap	after SgVc-swap
Density	95.0	96.5
W+H	2079.3	2044.6
Candidates		529
Improvements		11
Time(secs)		5.6

As a conclusion of this section, we propose the following combined neighborhood for solving packing problems with soft rectangles.

Neighborhood A: Neighborhood Jc(Db-swap)+SgAd-swap with the addition of neighborhood SgVc-swap.

In our experiment, the two neighborhoods in A are combined in the following manner: First apply local search with Jc(Db-swap)+SgAd-swap until a local optimal solution is obtained, and then improve it by local search with SgVc-swap. The best solution obtained is then output.

5 Computational Results

5.1 Benchmarks and Experiment

We used three benchmarks² known as ami33, ami49 and rp100, involving 33, 49 and 100 hard rectangles, whose widths and heights are denoted w_i^0 and h_i^0 , respectively. In the case of the perimeter minimization problem, we set the lower and upper bounds on widths and heights as follows

$$\begin{aligned} w_i^L &= (1 - e)w_i^0, \quad w_i^U = (1 + e)w_i^0 \\ h_i^L &= (1 - e)h_i^0, \quad h_i^U = (1 + e)h_i^0, \end{aligned} \quad (15)$$

where e is a constant like 0.1, 0.2, etc. The perimeter L_i in (3) of each rectangle i is set to $L_i = w_i^0 + h_i^0$.

On the other hand, for the area minimization problem, we first set the areas A_i in constraint (4) by $A_i = w_i^0 h_i^0$ for all i , the bounds on h_i as in (15), and then the bounds on w_i by

$$w_i^L = A_i/h_i^U \quad \text{and} \quad w_i^U = A_i/h_i^L. \quad (16)$$

Our algorithm was coded in C language, and run on a PC using Pentium 4 CPU, whose clock is 2.60 GHz and memory size is 780 MB. The linear and convex programming problems are solved by a proprietary software package NUOPT of Mathematical Systems Inc., where the linear programming is based on the simplex method and the convex programming is based on the line search method.

5.2 Perimeter Minimization Problem

The first set of instances of the perimeter minimization problem (13) are generated from ami33 and ami49 by setting constants e in (15) to $e = 0.0, 0.1, 0.2, 0.3$, respectively. For each e , five runs are conducted from independent random initial solutions. The data in Table 4 are the averages of these five runs, where Candidates, ..., W+H were already defined with Tables 1 and 3. Note that Density and W+H are given for both the average and best values in five runs.

From these results we see that our local search could obtain reasonably good packings in practically acceptable computation time, except for the case of $e = 0.0$ (i.e., all rectangles are hard). As we reduced the neighborhood size to a great extent, in order to make the whole computation time acceptable, the resulting size appears not sufficient for handling hard rectangles. Existing algorithms such as [6,7,10,13] developed for the hard case are much more efficient, partially because there is no need of mathematical programming problems to determine locations of rectangles and hence much larger neighborhoods can be used. However, if rectangles are soft, the solution quality improves quickly with e , indicating the feasibility of our approach in practical applications.

² See the footnote in Section 4.3.

We then solved the perimeter minimization problem of minimizing $W + \alpha v$ under the height constraint (11), for the comparison purpose with similar experiment for the area minimization problem in the next subsection. Three different H^* are tested for ami33 and ami49, respectively, where e is always set to 0.2. The results are shown in Table 5, where the data are the averages of five runs from independent random initial solutions.

Table 4. Perimeter minimization problem with different e

Benchmarks		$e = 0.0$	$e = 0.1$	$e = 0.2$	$e = 0.3$
ami33	Candidates	588.6	2639.6	2652.6	2189.0
	Improvements	44.4	96.0	102.8	110.4
	Time (secs)	24.6	127.2	135.2	118.1
	Density(av.)	67.5	93.6	97.1	97.4
	W+H(av.)	2633.4	2198.6	2116.0	2066.6
	Density(best)	74.4	96.3	97.9	99.8
	W+H(best)	2499.0	2159.9	2103.6	2041.6
ami49	Candidates	413.8	7877.6	12400.0	11472.2
	Improvements	34.6	153.0	218.8	236.2
	Time (secs)	28.0	641.8	1142.4	897.4
	Density(av.)	61.3	92.6	97.1	97.6
	W+H(av.)	15268.4	12346.3	11719.9	11401.9
	Density(best)	66.3	97.7	98.5	98.7
	W+H(best)	14644.0	11903.5	11686.8	11677.9

Table 5. Perimeter minimization problem with fixed heights H^*

Benchmarks		$H^* = 800$	$H^* = 1000$	$H^* = 1200$
ami33	Candidates	2063.0	2635.0	2762.6
	Improvements	92.8	104.6	113.6
	Time (secs)	100.1	134.0	146.2
	Density(av.)	91.4	95.4	96.7
	W(av.)	1530.6	1132.5	940.6
	Density(best)	94.5	97.2	98.2
	W(best)	1458.0	1090.7	926.0
		$H^* = 4400$	$H^* = 5500$	$H^* = 6600$
ami49	Candidates	10538.8	9871.2	7892.6
	Improvements	258.8	209.4	191.0
	Time (secs)	839.2	852.3	748.1
	Density(av.)	97.7	96.9	95.4
	W(av.)	7742.9	6354.3	5403.6
	Density(best)	99.1	98.2	97.8
	W(best)	7728.8	6192.9	5425.7

5.3 Area Minimization Problem

We solved the area minimization problem (14) with the same settings as Table 5, where five runs from random initial solutions were again carried out. The results

Table 6. Area minimization problem with fixed heights H^*

Benchmarks		$H^* = 800$	$H^* = 1000$	$H^* = 1200$
ami33	Candidates	1670.6	1477.4	867.0
	Improvements	114.6	111.2	92.8
	Time (secs)	179.4	155.9	90.7
	Density(av.)	99.0	97.4	96.6
	W(av.)	1461.3	1188.3	1000.8
	Density(best)	99.7	100.0	99.0
	W(best)	1449.2	1156.7	973.6
		$H^* = 4400$	$H^* = 5500$	$H^* = 6600$
ami49	Candidates	4156.2	3200.6	2195.0
	Improvements	241.2	184.2	140.4
	Time (secs)	967.2	743.2	511.6
	Density(av.)	98.8	95.9	91.9
	W(av.)	8152.5	6744.5	5868.5
	Density(best)	99.5	99.5	98.5
	W(best)	8097.6	6479.5	5454.4

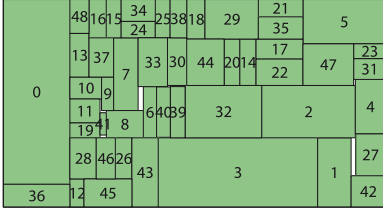
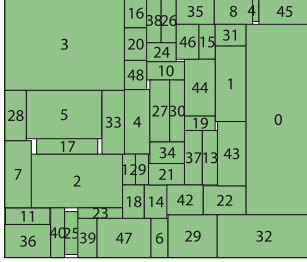
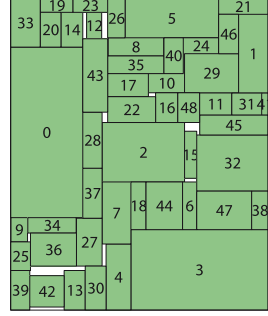
are shown in Table 6. Although, in this case, the convex programming problems $P_{\text{area}}(\sigma)$ are used instead of the linear programming problems, the computation time does not increase much, and very dense packings are obtained in most of the tested instances. Comparing with the results in Table 5, we see that the area minimization problem attains higher packing density, reflecting the nature of constraints. Figure 2 shows best results with ami49 for $H^* = 4400, 5500, 6600$, respectively.

5.4 Larger Problem Instances

To see how computation time and quality of solutions change with problem size, we tested a larger benchmark rp100 with 100 rectangles. Table 7 gives the results of problems (13) and (14) with $e = 0.2$ and $H^* = 450$ (in the case of (14)). The obtained result for the area minimization is shown in Figure 3. Considering that $2 \sim 3$ hours were consumed for each run, it appears difficult to handle larger problems than these with our current approach.

Table 7. Results with 100 rectangles

Benchmarks		Perimeter	Area
rp100	Candidates	35012	15933
	Improvements	536	500
	Time (secs)	7268.5	10101.5
	Density(%)	97.6	98.8
	Obj.	888.8	461.4

(a) $H^* = 4400$ (b) $H^* = 5500$ 

an effective combination of neighborhoods. As was shown by our experiment, it can provide packings of good quality for problem instances with up to 50 rectangles in reasonable amount of time. Our experiment is just preliminary, however, and we believe that the efficiency will be further improved by elaborating the use of mathematical programming techniques such as sensitivity analysis, and by adopting more sophisticated metaheuristic frameworks. Also, as observed in Table 4, our algorithm performs poorly for problem instances with hard rectangles, suggesting that more versatile neighborhoods are needed. But these directions remain as topics of future research.

References

1. D. P. BERTSEKAS, *Nonlinear Programming* (2nd Edition), Athena Scientific, 1999.
2. C.C.N. CHU AND E.F.Y. YOUNG, Nonrectangular shaping and sizing of soft modules for floorplan-design improvement, *IEEE Trans. Computer Aided Design of Integrated Circuits and Systems* (2004) **23**, 71-79.
3. J. DRÉO, A. PÉTROWSKI, P. SIARRY, E. TAILLARD, *Metaheuristics for Hard Optimization*, Springer, 2006.
4. F.W. GLOVER, G.A. KOCHENBERGER (EDS.), *Handbook of Metaheuristics*, Springer, 2003.
5. H. ITOGA, C. KODAMA AND K. FUJIYOSHI, A graph based soft module handling in floorplan, *IEICE Trans. Fundamentals* (2005) **E88-A**, 3390-3397.
6. S. IMAHORI, M. YAGIURA AND T. IBARAKI, Local search algorithms for the rectangle packing problem with general spatial costs, *Mathematical Programming* (2003) **B97**, 543-569.
7. S. IMAHORI, M. YAGIURA AND T. IBARAKI, Improved local search algorithms for the rectangle packing problem with general spatial costs, *European Journal of Operational Research* (2005) **167-1-16** 48-67.
8. H. KONNO AND T. KUNO, Multiplicative programming problems, In *Handbook of Global Optimization*, edited by R.Horst and P.M.Pardalos, Kluwer Academic Publishers (1995), 369 - 406.
9. A. LODI, S. MARTELLO AND M. MONACI, Two-dimensional packing problems: A survey, *European Journal of Operational Research* (2002) **141**, 241-252.
10. H. MURATA, K. FUJIYOSHI, S. NAKATAKE AND Y. KAJITANI, VLSI module placement based on rectangle-packing by the sequence-pair, *IEEE Transactions on Computer Aided Design* (1996) **15-12**, 1518-1524.
11. H. MURATA AND E.S. KUH, Sequence-pair based placement method for hard/soft/preplaced modules, *Proc. Int. Symp. Physical Design*, (1998) 167-172.
12. K. NAKAMURA, Packing problems with adjustable rectangles under perimeter and/or area constraints, Graduation thesis, Department of Informatics, School of Science and Technology, Kwansei Gakuin University, March 2006.
13. S. NAKATAKE, K. FUJIYOSHI, H. MURATA AND Y. KAJITANI, Module placement on BSG-structure and IC layout applications, Proceedings of International Conference on Computer Aided Design 15-12 (1996) 484-491. SIAM Pub., 1994.
14. Y. NESTEROV AND A. NEMIROVSKII, *Interior Point Polynomial Algorithms in Convex Programming*, SIAM Pub., 1994.
15. F.Y. YOUNG, C.C.N. CHU, W.L. LUK AND Y.C. WONG, Handling soft modules in general nonslicing floorplan using Lagrangean relaxation, *IEEE Transactions on Computer-Aided Design of Integrated Circuit and Systems* (2001) **20**, 687-692.

A Multi-population Parallel Genetic Algorithm for Highly Constrained Continuous Galvanizing Line Scheduling

Muzaffer Kapanoglu and Ilker Ozan Koc

Department of Industrial Engineering Eskişehir Osmangazi University
26030 Eskişehir, Turkey
{muzaffer, ikoc}@ogu.edu.tr

Abstract. The steelmaking process consists of two phases: primary steelmaking and finishing lines. The scheduling of the continuous galvanizing lines (CGL) is regarded as the most difficult process among the finishing lines due to its multi-objective and highly-constrained nature. In this paper, we present a multi-population parallel genetic algorithm (MPGA) with a new genetic representation called k^{th} nearest neighbor representation, and with a new communication operator for performing better communication between subpopulations in the scheduling of CGL. The developed MPGA consists of two phases. Phase one generates schedules from a primary work in process (WIP) inventory filtered according to the production campaign, campaign tonnage, priorities of planning department, and the due date information of each steel coil. If the final schedule includes the violations of some constraints, phase two repairs these violations by using a secondary WIP inventory of steel coils. The developed scheduling system is currently being used in a steel making company with encouraging preliminary results.

Keywords: multi population genetic algorithm, real world application, continuous galvanizing line, scheduling.

1 Introduction

The steelmaking process (for steel sheet products) consists of two phases: primary steelmaking and finishing lines. In the primary steelmaking, slabs created by the upstream processes are transformed to hot coils at a hot strip mill. Although some papers have been published on production scheduling in primary steelmaking ([5], [8], [9], [13]), there could be found only one research related to the finishing line scheduling [10]. Okano, et al., [10] have worked on the problem of creating production campaigns and sequencing of coils within each campaign so that productivity and product quality are maximized and tardiness is minimized. They have proposed a construction heuristic and an improvement heuristic to generate schedules for finishing lines excluding CGL which is regarded as the most difficult process in the finishing lines in terms of sequencing [10].

Multi-population GAs are the most popular parallelization method with numerous publications. A detailed discussion of parallel GAs, including multi-population (or

multiple-deme) GAs, can be found in [1]. Basically, multi-population GA work with more than one population called subpopulations, and facilitates some sort of communication between the subpopulations with a communication operator based on a topology of connections. In our previous study [7], a multi-population structure with a new genetic representation, called k^{th} nearest neighbor representation, was used to generate a number of local optimum solutions quickly by using a greedy GA over each subpopulation. However, in the MPGA approach we present here, GAs are designed to maintain different levels of greediness over different subpopulations. The level of greediness supports the robustness and efficiency of GAs. We also developed (i) a new communication operator which works on a fully connected topology, and controls the knowledge transfer and the communication frequency by incorporating certain tabu search features, and (ii) a greedy mutation operator which preserves feasibility after mutation.

2 Continuous Galvanizing Line Scheduling

The considered problem is the scheduling of a continuous galvanizing line (CGL) in a flat steelmaking plant with the one-million-tonne annual production capacity. CGL scheduling is an extremely complex problem in steelmaking industries due to the following challenges. (i) Coil changeover requirements: Succeeding steel coils must comply with the changeover requirements of each equipment of CGL with respect to the preceding coils, (ii) Planning requirements: Further constraints such as the order due dates, and the priorities of the planning department, and (iii) Multi-objectivity: Schedules are subject to the optimization of objectives related to the product quality and the line productivity.

Production in the CGL is carried out in production campaigns constructed based on a selected campaign type for a given campaign tonnage. A campaign type is a cluster of coils of a particular type, with respect to quality and thickness. A production campaign is a production run with specific start and end times in which coils are processed continuously. A production campaign is constructed from the primary and secondary WIP inventories. The primary WIP inventory is the first N coils that fill the campaign tonnage when the coils of a campaign type are sorted according to the due dates and the predefined priorities. The remaining coils from the selected campaign type and the coils of different campaign types that can be used to improve the quality of the schedule are called the secondary WIP inventory. First, the coils of the primary WIP inventory are taken into the production campaign and scheduled. If the schedule violates any constraint(s), then the coils of the secondary WIP inventory are taken into the production campaign to fix the transition violations with a minimum increase in campaign tonnage. The relationship between the production types, primary WIP inventory, secondary WIP inventory, and the production campaign is shown in Figure 1. In Figure 1, the white coils are the coils of selected campaign type that fills the campaign tonnage. These white coils form the primary WIP inventory. All the remaining coils in the selected campaign type are colored gray. Also the coils which belong to different campaign types but can be produced with the selected campaign

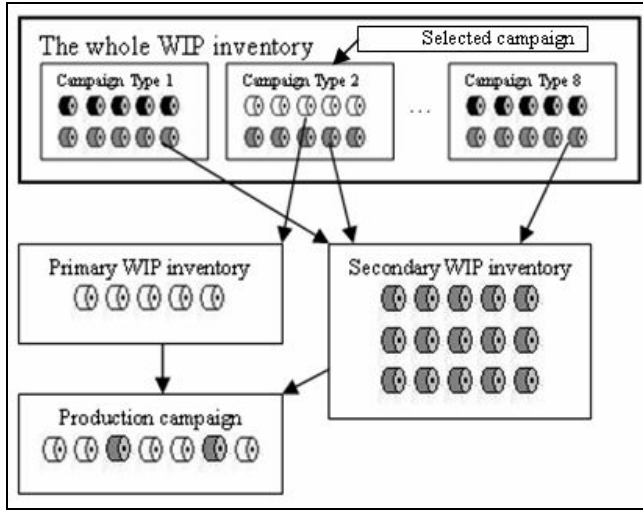


Fig. 1. The relationship between the campaign type, primary WIP inventory, secondary WIP inventory and the production campaign

type are colored gray. These gray coils form the secondary WIP inventory. Firstly, a production campaign is constructed with primary WIP inventory, i.e., with white coils. If there are constraint violations in production campaign, then these violations are fixed by using the secondary WIP inventory.

To make a schedule compatible with the last coil of the previous schedule and the first coil of the next schedule, it is needed to define two coils as an input: starting coil and ending coil. The starting coil is the last coil of the previous schedule. Since the next schedule is not known beforehand, the ending coil is an artificial (dummy) coil that can be compatible with the schedule of the next campaign type. Now the scheduling process can be summarized in the following steps:

Step 1. Determine primary and secondary WIP inventories.

Step 2. Construct a production campaign compatible with the starting and ending coils by scheduling the primary WIP inventory.

Step 3. If the schedule obtained is feasible, go to Step-5.

Step 4. Repair the schedule by adding minimum number of coils from the secondary WIP inventory

Step 5. Stop and report the final schedule of the current production campaign to the decision maker.

The CGL scheduling, as stated above, is an interesting problem since it includes many kinds of theoretical problems from the literature. In Step 2, if the starting and the ending coils are same, the problem is the shortest Hamiltonian cycle problem (i.e., well-known Traveling Salesman Problem). However, if the starting and the

ending coils are not same, then the problem corresponds to the shortest Hamiltonian path problem. In Step 4, a repair algorithm runs for repairing the constraint violations. A constraint violation can occur between two adjacent coils in the schedule. Therefore, for the repair algorithm, the first coil of a violation is defined as the starting coil and the second coil as the ending coil. In this case, the problem is to find a feasible schedule between the starting and the ending coils by using minimum number of coils from the secondary WIP inventory. If the starting and the ending coils are same, problem turns out to be a special case of Prize-Collecting Traveling Salesman Problem in which a traveler must visit minimum number of external nodes to find a legal tour. In Step 4, if the starting and the ending coils are different, then the problem can be defined as a "prize-collecting" Hamiltonian path problem in which a traveler must visit minimum number of nodes to find a legal path from initial node to the ending node.

The constraints of the CGL scheduling problem are presented below:

- i. *The thickness difference between the two consecutive coils can be $P_{thickness1}\%$ of the thickness of thinner coil while getting thinner and $P_{thickness2}\%$ of the thinner coil while getting thicker at maximum.*

For example, assuming that $P_{thickness1} = 10$ and $P_{thickness2} = 20$, and two adjacent coils in a schedule have thicknesses t_i and t_j . If $t_i > t_j$, then it means that the thickness is getting thinner and $(t_i - t_j \leq 0.10 \times t_j)$ must hold. If $t_i < t_j$, then it means that the thickness is getting thicker and $(t_j - t_i \leq 0.20 \times t_i)$ must hold. Note that the case $t_i = t_j$ is the ideal case and always holds for any value of $P_{thickness1}$ and $P_{thickness2}$. Also it is important to notice that although a sequence of coils in the schedule (j_i, j_j) is feasible, the sequence (j_j, j_i) may not be feasible.

- ii. *The width difference between the two consecutive coils can be P_{width1} mm while getting narrower and P_{width2} mm while getting wider at maximum.*

Assuming that $P_{width1} = 200$ and $P_{width2} = 150$ mm, and two adjacent coils in a schedule have widths $width_i$ and $width_j$. If $width_i > width_j$, then it means that the thickness is getting narrower and $(width_i - width_j \leq 200)$ must hold. If $width_i < width_j$, then it means that the width is getting wider and $(width_j - width_i \leq 150)$ must hold. The case $width_i = width_j$ is the ideal case and always holds for any value of P_{width1} and P_{width2} .

- iii. *Coating thickness difference between the two consecutive coils can be $P_{coating}$ gr per square meter at maximum.*

Assuming that two adjacent coils have coating thicknesses $coating_i$ and $coating_j$, $(|coating_i - coating_j| \leq P_{coating})$ must hold.

- iv. *The annealing cycle type of the two consecutive coils must correspond to a permitted transition in the annealing cycle transition matrix.*

The annealing cycle transition matrix shows the allowed and restricted transitions among annealing cycle types. A smaller size typical annealing cycle transition matrix is given in Table 1, where "X" represents a restricted transition and "A" represents an allowed transition among two annealing cycles.

Table 1. An example annealing transition matrix

Cycles	10	20	30	40	41	50	51	60	61
10	A	A	A	X	X	X	X	X	X
20	A	A	A	X	X	X	X	X	X
30	A	A	A	A	A	A	A	A	A
40	X	X	A	A	A	A	A	X	X
41	X	X	A	A	A	A	A	X	X
50	X	X	A	A	A	A	A	A	A
51	X	X	A	A	A	A	A	X	X
60	X	X	A	X	X	A	X	A	A
61	X	X	A	X	X	A	X	A	A

- v. *The width enlargement in the schedule can only be made with the special coils, those with lower quality specifications or those that are non-skin-passed.*

CGL includes a process in which work rolls operate on coils to achieve some customized skin-pass requirements. Since this process is accomplished by applying a high pressure with work rolls to the surface of a coil, every coil causes wear on the work rolls. Therefore, if the width is enlarged with some coils that have to be skin-passed, the wear on a work roll yields traces on the surface of the enlarged coils. Hence, the width enlargement can only be made with the coils that are non skin-passed or the coils with lower quality specification.

- vi. *There must be at least one non-skin-passed coil between two coils requiring different skin-pass mills to operate.*

Since a setup is required to adjust CGL for different skin-pass mills, at least one non-skin-passed coil should be placed between two coils requiring different kinds of skin-pass mills.

- vii. *The exiting inner diameter of a coil that will be side trimmed must be the same with the preceding and succeeding coils.*

Since a side trimmed coil and a change in the exiting diameter require setups that can not be performed simultaneously, a change in the exiting diameter is not permitted with the coils that will be side trimmed.

The objectives of the CGL scheduling problem are stated as follows:

- Minimize the total number of the width-increase chains* where a width-increase chain is defined as a sub-schedule in which the widths of the coils continuously increase as illustrated in Figure 2.
- Minimize the total length of the width-increase chains* where the length of a width-increase chain is the number of coils in a chain.
- Minimize the total number of the exiting inner diameter changes.*
- Minimize the total number of the passivation type change.*
- Minimize the total deviations of thickness.*
- Minimize the total number of oil type changes.*
- Minimize the total number of coils used from secondary WIP inventory.*

All the constraints and the objectives are unified into a single objective function by adding up the penalized constraint violations and the weighted objective values for

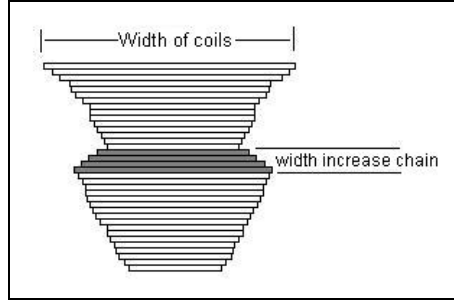


Fig. 2. Width increase chain

minimization. The objective function with user-supplied penalties and weights can be represented as follows:

$$\min z = \sum_{i=1}^7 p_i C_i + \sum_{j=1}^7 w_j O_j \quad (1)$$

where C_i and p_i represent the violation amount of the constraint i , and its associated penalty; and O_j and w_j represent the value of the objective j and its assigned weight respectively. This approach allows constraint violations proportional to their relative penalties. In CGL scheduling problem, with guidance of scheduling experts, all the constraint violations and the objectives are sorted according to their importance levels in which a tradeoff is not permitted among any high important and low important components, i.e., the preemptive case. If preemptive priorities are preferred for constraint violations and objectives, then penalties and the weights must be chosen accordingly to prevent any possible tradeoff among them. These penalties and the weights are determined by applying a series of *preferable* – *not preferable* comparisons to the scheduling experts of the plant. The developed software also includes an interface allowing manual changes in weights and penalties to reflect any possible changes in the system dynamics.

3 Developed MPGA

We have developed a multi-population parallel genetic algorithm using the k^{th} nearest neighbor representation [7], for preparing schedules for the CGL. This representation has previously been presented in [7] for the Euclidean traveling salesman problem with high performance. In this paper, we introduce a new approach with some extensions and modifications for a highly constrained multi-objective real CGL scheduling problem.

The multi-population GAs are the most popular parallel method among the parallel GAs with numerous publications [1]. The two important characteristics of multi-population parallel GAs are a number of subpopulations and a communication operator with a topology that defines the connections between the subpopulations. Probably the most important part of the multi-population parallel GAs is the communication operator. If there is no communication among subpopulations, then the multi-population GA exhibits an equivalent performance to running a number of

individual GAs in parallel, or running a GA in multiple times sequentially. In our MPGA we have developed a new communication operator which:

- uses a fully connected topology for communication,
- exchanges some parts of two individuals selected from different subpopulations,
- utilizes a communication length parameter (*comlength*) which controls the amount of knowledge transfer among individuals,
- utilizes a *tabu list* to control and restrict the communication among the subpopulations which have recently communicated,
- preserves the transferred knowledge against the evolution process for a number of generations.

The main framework of MPGA is given in Figure 3. A fully connected topology is used for the communication operator and communication takes place among randomly selected two subpopulations.

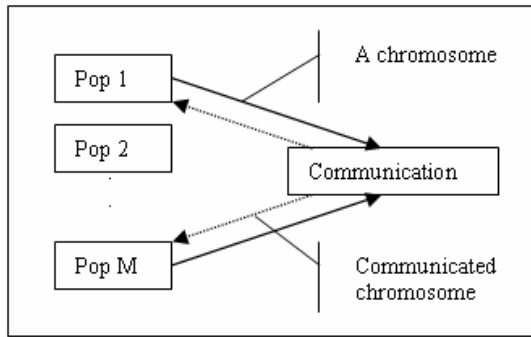


Fig. 3. The framework of MPGA

In [7], multi-population structure is used to generate a number of local optimum solutions quickly by using a greedy GA over each subpopulation. In our MPGA approach, we have used GAs that have different levels of greediness over different subpopulations. Different levels of greediness for GAs are obtained by using a different probability distribution for each subpopulation which is utilized by the k^{th} nearest neighbor representation. By this way, a GA with a lower level of greediness becomes more robust than a GA with a higher level of greediness. As the level of greediness decreases, the exploration capability of GA increases. While the level of greediness increases, the exploitation capability of the GA increases. Therefore, the communication among subpopulations enables GAs to operate on the imported knowledge with different levels of greediness. We also developed a greedy mutation operator which performs an improvement heuristic to preserve feasibility after mutation. In the following subsections we have described the genetic representation and the operators used in MPGA.

3.1 k^{th} Nearest Neighbor Representation

The k^{th} nearest neighbor representation extends the nearest neighbor heuristic to the k^{th} nearest neighbor for TSPs. A fully-connected Hamiltonian graph can be constructed where the coils are nodes, and the distances are the transition costs for scheduling CGL. The transition costs from one coil to another are computed based on penalized violations of the constraints and the weighted objectives. Therefore, each GA searches for a schedule of maximum fitness in which the next coil that will be taken into the schedule can be selected from out of the unscheduled nearest k coils. The parameter k can be considered as the maximum adjacency degree due to its restrictiveness over the neighborhood of a coil. Since a gene represents the g^{th} unvisited nearest neighbor of the current coil to schedule next where $1 \leq g \leq k$, the maximum values of genes must be determined depending on the total number of coils, the value of k , and the position of the gene as follows;

$$\max g_i = \begin{cases} k & \text{if } N - i + 1 \geq k \\ N - i + 1 & \text{if } N - i + 1 < k \end{cases} \quad (2)$$

where g_i represents the value of a gene in the i^{th} position of a chromosome, and N represents the total number of coils that will be scheduled. By using the k^{th} nearest neighbor representation for $k = 3$, a chromosome for a 5 coil problem, with the maximum value limits for each gene, is illustrated as follows.

Maximum value limits:	3	3	3	2	1
Chromosome:	1	3	2	1	1

The starting coil is the last coil of the previous schedule. Since the last coil of the previous schedule (i.e., our starting coil) that is currently in production line, and the first coil of the current schedule that will be prepared must be compatible with each other, we will decide which coil to take the schedule first according to our starting coil. Since the allele of first gene is 1, we select the first nearest neighbor of the starting coil. Suppose that the first nearest neighbor of the starting coil is coil 5, now the current schedule is {5}. Since the allele of the second gene is 3, we select the third unvisited nearest neighbor of coil 5. Suppose that this coil is coil 2, now the current schedule is {5, 2}. Since the allele of the third gene is 2, we select the second unvisited nearest neighbor of coil 2, namely coil 3. Therefore, the current schedule is {5, 2, 3}. As the subsequent chromosome is decoded in the same manner, the schedule {5, 2, 3, 1, 4} is obtained.

Since the frequency of visiting the nearest neighbor, and the k^{th} nearest neighbor in the optimal solution can not be the same, the k^{th} nearest neighbor representation utilizes a probability distribution for the degree of neighborhood. This probability distribution is used in the initialization of the subpopulation phase, and in the mutation operator to determine the new allele of a gene. Therefore, these probability distributions, each one for a subpopulation, also describe the greediness of GAs. We have defined these probability distributions according to the ratio of probabilities of

Table 2. The ratios used to generate probability distributions for k^{th} nearest neighbor representation

Subpopulations	Ratio
1	1.5
2	1.4
3	1.3
4	1.2
5	1.1

adjacent closeness degrees. For example, a ratio of R indicates that the probability of visiting the first nearest neighbor is R times more than the probability of visiting the second nearest neighbor and R^2 times more than visiting the third nearest neighbor and so on. Therefore, the greediness of a subpopulation increases as the value of R increases. These ratios are given in Table 2 for each subpopulation.

Since we have many constraints related to the adjacency of coils in CGL scheduling, a feasible transition from one coil to another is often limited to a degree of closeness. By restricting the available connections from a coil to at most its k^{th} nearest neighbors, we reduce the size of the search space, and eliminate most of the infeasible transitions among coils.

3.2 The Communication Operator

Communication operator is probably the most important operator in parallel GAs which has a significant effect on the performance of parallel GAs as mentioned in [11]. In general, a migration policy is preferred as a communication operator ([2], [6], [11], [3], [4]). Although the communication rate and the topology of connections between subpopulations are important to address the communication level of an algorithm, migration can be considered as a high level of communication since a migration from quickly converged subpopulation can easily capture the slowly converging subpopulations. As observed in [6], the performances of the parallel GAs with such a high level of communication are similar to the performance of a GA with a single large population and while designing a multi-population parallel GA, the topology and the communication level can not be distinguished from each other completely [2]. Therefore, the prevention feature of a premature convergence must be embedded in either the topology or the communication operator, or both. We preferred to use a fully connected topology with a new communication operator which allows the control of the communication level by controlling the communication probability and the communication amount (number of genes). The developed communication operator also restricts the re-communication of recently communicated subpopulations via a *tabu list*. Therefore, the topology used in this paper can be considered as a fully connected topology with dynamically changing communication restrictions that embeds the prevention of premature convergence in both, the topology and the operator.

We have proposed a new communication operator which acts like a crossover operator to perform knowledge exchange among individuals. The proposed operator also utilizes a *tabu list* and a *tabutenure* parameter to store the recently communicated

subpopulations and to restrict them to not to re-communicate for a number of generations, respectively. By this way, subpopulations are prevented against the imported knowledge influx. Also the communicated individuals are preserved against the evolution process if their subpopulations are still in the *tabu list* (i.e., for *tabutenure* generations). The waiting times in the *tabu list* are updated by GAs. Each GA checks the index of its subpopulation at the *tabu list* after every generation and updates its waiting time if its subpopulation index exists in the list. If the index of its subpopulation doesn't exist in the *tabu list*, it sends a communication request to the communication operator with a probability of communication (*pcom*). The communication operator is activated when a request is received from a subpopulation. The operator randomly selects another subpopulation that is not in the *tabu list*, then performs communication among two individuals randomly selected from corresponding subpopulations, and finally updates the *tabu list*.

The communication operation works on the phenotypes of the selected chromosomes. To illustrate the communication operation, assume that the solutions decoded from the selected chromosomes are as follows:

$$\begin{aligned} A &= (1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9) \\ B &= (2 \ 5 \ 4 \ 6 \ 9 \ 1 \ 3 \ 7 \ 8) \end{aligned}$$

It randomly selects a communication site (*comsite*) from interval $[0, N_{comamount}]$ where, *comamount* represents the communication amount (i.e., the length of substring that will be exchanged). Assume that *comsite* = 2 and *comamount*=4. Then, the substrings that will be exchanged are $s_1 = (3, 4, 5, 6)$ and $s_2 = (4, 6, 9, 1)$. Now we will produce the second substring in parent A, and the first substring in parent B. While completing this operation, our purpose is to protect the relative positions of the sub-strings in the parents in which they will be produced. To accomplish this goal, all the elements, except the first one, in the second substring are deleted from parent A, and similarly all the elements, except the first one, in the first substring are deleted from parent B. After this operation, current schedules are reduced to (2 3 4 5 7 8) and (2 9 1 3 7 8). Adding the remaining elements of the substrings after their first element in the corresponding offspring yields the following two new schedules:

$$\begin{aligned} a &= (2 \ 3 \ 4 \ 6 \ 9 \ 1 \ 5 \ 7 \ 8) \\ b &= (2 \ 9 \ 1 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8) \end{aligned}$$

The obtained offspring are re-encoded into corresponding chromosomes. Since the offspring might not be representable with the k^{th} nearest neighbor representation, if needed, it is allowed to exceed “*k*” in re-encoding process. Preserving the relative starting positions of the exchanged substrings is the main advantage of this communication operator.

3.3 Genetic Operators and Parameters

The operators and the parameters of the designed GA are described below.

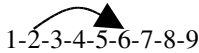
- **The maximum nearest neighborhood degree allowed:** $k = 10$.
- **Number of populations:** 5

- **Subpopulation sizes:** $popsiz = 20$. Therefore, there are 100 chromosomes in total.
- **Initial population:** Initial subpopulations are generated randomly according to the predefined probability distributions for each subpopulation, computed based on Table 2.
- **Selection operator:** We used the *tournament selection operator* with tournament size $tsiz = 3$. The tournament selection operator simply selects $tsiz$ chromosomes from the current population and places the fittest one to the new population until the new population is filled.
- **Crossover operator:** We have used the single point crossover operator with probability $pcross = 0.1$.
- **Mutation operator:** We have proposed a new mutation operator. The mutation simply selects a gene with probability $pmutate$ ($pmutate=0.001$), and mutates its current value according to the probability distribution generated according to the ratios given in Table 2. The mutation operation is performed on the genotype. Since mutating a gene may result in a complete change in the phenotype succeeding the mutated position, we only consider the jump effect of the mutation on the phenotype. To illustrate this case, consider the following phenotype and assume that the gene at the third position is to be mutated.

1-2-3-4-5-6-7-8-9

Mutating the gene at the third position corresponds to a jump from the second position in the phenotype.

1-2-3-4-5-6-7-8-9



In this case the coils 3, 4 and 5 will be excluded from the schedule. To restore these coils to the schedule, we perform a cheapest insertion operation which inserts these coils to the cheapest available location while preserving the newly produced connection, i.e., the connection from coil 2 to 6. In cheapest insertion operation, we do not care about the “ k ” restriction over the neighborhood. Therefore, if needed, it is allowed to exceed “ k ”. The final schedule is then re-encoded into the chromosome.

- **Elitism:** Elitist strategy is used to preserve the best solution obtained against the selection, crossover and mutation operators. Elitism simply saves the best-so-far chromosome throughout generations, and replaces the worst chromosome with the elite after each generation within each subpopulation.

4 Repairing Algorithm

The MPGA described in Section 3 schedules the primary WIP inventory. If the found best solution of MPGA has some constraint violations, then the repairing algorithm (RA) is performed to repair or reduce the amount of violations in found best solution

of the MPGA one by one by using minimum number of coils from secondary WIP inventory. The sequence of coils taken from secondary WIP inventory to repair a violation is called a sub-schedule. An illustrative example is given in Figure 4 for two constraint violations between the coils A-B and C-D. Figure 4 shows two sub-schedules. The first sub-schedule consists of four coils taken from secondary WIP inventory to repair the violation between A and B, whereas the second one consists of four coils repairing the violation between C and D. The RA runs for each constraint violation consecutively, starting from the violation which has the largest penalty value.

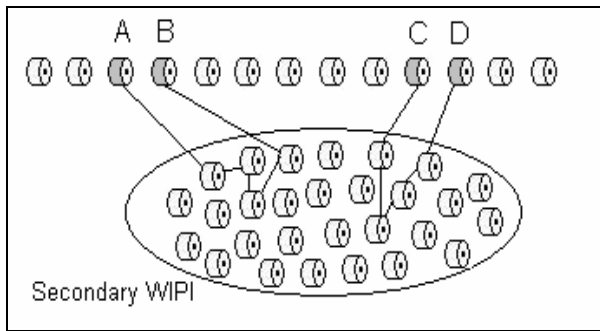


Fig. 4. Repairing the constraint violations

The RA utilizes the cheapest insertion heuristic [12] for repairing the constraint violations. In the considered problem, the number of coils used from secondary WIP inventory to repair a violation depends on the violation type. On the average, a violation can be repaired by using 1-5 coils from secondary WIP inventory. Therefore cheapest insertion is a fast and effective tool for violation repairing. It starts with the constraint violation that has the largest penalty, and sets the first coil of constraint violation as the initial coil and the following as the ending coil. Letting that the initial coil be A and the ending coil be B as given in Figure 4, RA tries to repair the violation between coils A and B by iteratively inserting best available coil of the secondary WIP inventory at the best available position in the sub-schedule until either no further improvement in the objective value can be achieved or the violation is fully repaired. Then, this procedure continues to repair the next constraint violation with the largest penalty until all violations are considered.

5 Results

The developed MPGA has already been put to practice at a major steelmaking plant in Turkey. Although no detailed comparison has been completed yet to prove the contribution of the MPGA to the scheduling of CGL experimentally, the preliminary results are encouraging. High quality schedules have already been generated by using MPGA within very reasonable computational times (2-3 minutes for scheduling

150-200 coils in roughly 25-30 generations) when compared to those of the human scheduling experts.

Our intention to perform a comparison of our approach faces two drawbacks: (i) No prior research is found on CGL scheduling as highlighted in the first section. Therefore, a literature-based comparison of the performances of our MPGA and any other technique including standard GA is not available. (ii) The human scheduling experts have psychological reactions against the early successful results of MPGA which caused them to avoid many attempts in comparing their performances with MPGA's. Although this is an on-going evaluation process that can take more than a year, a typical performance of MPGA versus human experts is presented in Table 3 for a smaller size sample case. For instance, MPGA was able to schedule all 66 coils of the primary inventory while only Expert#2 could schedule the same number of coils. Since the reason behind missing coils is to avoid some important violations, number of missing coils can also be counted as violations. Table 3 does not include all the constraints and the objectives as addressed in Section 2 due to the incapability of the human experts to evaluate more than nine criteria concurrently. For this case, the schedule obtained by using MPGA achieved 10 violations while the best expert resulted in 16 violations.

Table 3. A sample case of 66 coils: MPGA vs. human scheduling experts

Evaluation Criteria	MPGA	SE* #1	SE* #2	SE* #3
Number of coils	66	63	66	65
No. of violations on width differences	0	1	0	0
No. of violations on thickness differences	0	1	1	0
No. of violations on widening	0	2	0	0
No. of violations on annealing cycle type	1	0	1	1
No. of violations on skin-pass mill	0	0	0	0
No. of violations on inner diameter changes	8	8	10	10
No. of violations on inner diameter changes with side-trimmed coils	1	5	5	5
No. of violations on coating thickness	0	1	1	0
Total number of violations	10+0	18+3	18+0	16+1

*SE: Scheduling Expert

According to our preliminary results, 60% to 75% improvement in the number of constraint violations, and 5-25% improvement in the objective values can be expected in a realistic realm.

6 Conclusions

CGL scheduling in steelmaking is a challenging real-world problem incorporating multiple objectives, and multiple constraints into various types of TSP and Hamiltonian path problems. In this study, a multi-population parallel genetic algorithm with a new genetic representation and new operators is developed for this challenging problem. The developed approach produces a schedule in two phases: (i) schedule construction phase, and (ii) schedule improvement phase. Phase one schedules the primary WIP inventory

which includes N coils selected according to the campaign type, campaign tonnage, priorities of the planning department and the due dates. Phase two is designed to repair violations by using a secondary WIP inventory for improving the quality of the schedules. Secondary WIP inventory includes the remaining coils from the selected campaign type, and some of the coils of other campaign types that are compatible with the ones in primary WIP inventory. Although the performance evaluation of the designed algorithm is an ongoing process, preliminary results indicates that the algorithm outperforms the schedules of human scheduling experts and a 60% to 75% improvement in the number of constraint violations, and 5-25% improvement in the objective values can be expected.

References

1. Cantú-Paz, E.: A survey of parallel genetic algorithms. IlliGAL Report 97003, Illinois Genetic Algorithms Lab., University of Illinois (1997).
2. Cantú-Paz, E., Goldberg, D.E.: Efficient parallel genetic algorithms: theory and practice. *Computer Methods in Applied Mechanics and Engineering*, 186 (2000) 221–238.
3. Cohoon, J.P., Martin, W.N., Richards, D.S.: Genetic Algorithms and punctuated Equilibria in VLSI. In: Schwefel H.-P., Männer, R. (eds.): *Parallel Problem Solving from Nature*, Springer-Verlag (Berlin), (1991) 134–144.
4. Cohoon, J.P., Martin, W.N., Richards, D.S.: A multi-population genetic algorithm for solving the K-partition problem on hyper-cubes. In: Belew, R.K., Booker, L.B. (eds.): *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann (San Mateo, CA), (1991) 244–248.
5. Fang, H.-L., Tsai, C.-H.: A Genetic Algorithm Approach to Hot Strip Mill Rolling Scheduling Problems. In: *Proceedings of the International Conference on Tools with Artificial Intelligence*, IEEE, Piscataway, NJ (1998) 264–271.
6. Grosso, P.B.: Computer simulations of genetic adaptation: parallel subcomponent interaction in a multilocus model, Ph.D. Thesis, The University of Michigan, (1985).
7. Kapanoglu, M., Koc, I.O., Kara, İ., Aktürk, M.S.: Multi-population genetic algorithm using a new genetic representation for the Euclidean traveling salesman problem. In: Durmusoglu M.B., Kahraman, C. (eds.): *Proceedings of the 35th International Conference on Computers & Industrial Engineering*, Turkey, Vol. 1 (2005) 1047–1052.
8. Lee, H.-S., Murthy, S.S., W. Haider, S., Morse, D. V.: Primary Production Scheduling at Steelmaking Industries. *IBM Journal of Research and Development*, 40 (1996) 231–252.
9. Lopez, L., Carter, M.W., Gendreau, M.: The Hot Strip Mill Production Scheduling Problem: A Tabu Search Approach, *European Journal of Operational Research*, 106 (1998) 317–335.
10. Okano, H., Davenport, A.J., Trumbo, M., Reddy, C., Yoda, K., Amano M.: Finishing line scheduling in steel industry. *IBM Journal of Research and Development*, 48, 5/6 (2004) 811–830.
11. Petty, C.B., Leuze, M.R., Grefenstette, J.J.: A parallel genetic algorithm. In: Grefenstette, J.J. (ed.), *Proceedings of the Second International Conference on Genetic Algorithms*. Lawrence Erlbaum Associates (Hillsdale, NJ), (1987) 155–161.
12. Reinelt, G.: The traveling salesman problem: Computational Solutions for TSP Applications. *Springer Lecture Notes in Computer Science* 840, Springer-Verlag, Berlin (1994).
13. Yasuda, H., Tokuyama, H., Tarui, K., Tanimoto, Y., Nagano, M.: Two-Stage Algorithm for Production Scheduling of Hot Strip Mill. *Operations Research*, 32 (1984) 695–707.

Improvement in the Performance of Island Based Genetic Algorithms Through Path Relinking

Luis delaOssa, José A. Gámez, and José M. Puerta

Intelligent Systems and Data Mining Group
Computer Systems Department - University of Castilla-La Mancha
Campus Universitario s/n, 02071, Albacete Spain
{ldelaossa, jgamez, jpuerta}@info-ab.uclm.es

Abstract. In island based genetic algorithms, the population is splitted into sub-populations which evolve independently and ocasionaly communicate by sending some individuals. This way, several zones of the landscape are explored in parallel and solutions with different features can be discovered. The interchange of information is a key point for the performance of these algorithms, since the combination of those solutions usually produces better ones. In this work, it is proposed a method based in path relinking which makes the combination process more effective.

1 Introduction

Genetic algorithms (GAs) have been succesfully applied to solve complex problems in areas such as engineering, science, etc [11]. Although they can reach good solutions with not too much effort, the increasing complexity of those problems have lead the community to research for improvements.

One of these research lines, due to the parallel nature of this kind of algorithms, consisted of taking advantage of the power of parallel machines [13]. These attempts gave rise to new kinds of approaches[4] that spread from those which make simple parallelization of the evaluation function, to models that make a huge use of communication and behave in a different manner than their sequential counterparts. These last are called fine grained parallel GAs or cellular GAs [1].

Between these two tendencies, there can be placed the coarse grained parallel GAs or island models[4]. They basically consist of several subpopulations (also called demes or islands) that evolve independently and, ocasionaly, send copies of their individuals to the other islands.

The difference in performance between island models and sequential GAs is due to the fact that each subpopulation follows its own evolution process, and that makes it possible the exploration of different regions of the search space. Moreover, the quality of the solutions in each island improves when mixing them with the received ones.

Since this communication process has been shown as a key point in the functioning of the algorithms[14] it becomes important doing it as effective as possible.

In [18] the authors proposed an island-based parallel evolutionary algorithm in which a simple estimation of distribution algorithm (UMDA) evolves inside each island. In

that paper, the novelty lies in the way in which the information is incorporated into the resident island. Thus, each island migrates a (univariate) probabilistic model, and the incoming model is combined with the resident one by using an informed search: *path relinking*. The results obtained were very good, specially for deceptive problems. Encouraged by these results, in this work we extend our analysis to the case in which genetic algorithms evolve inside each island. Thus, we propose a method that only requires each island to send an individual. Then, the receiving island generates a new individual by doing a path relinking based search between it and the best one on its population and incorporates it. This way, it is tried to take as much advantage as possible of the usefull information but, at the same time, trying to not alter the population significantly.

We also compare the incorporation of information through path relinking in island based genetic algorithms with the traditional migration of individuals. In order to do that, we test the behaviour of both approaches on the solution of five different functions under several configurations. Afterwards, we study the behaviour of our proposal by analysing the effects of each one of the parameters.

This paper is divided into 5 sections. In Sect. 2 we make a brief overview of island based genetic algorithms. Section 3 describes the motivations of our work as well as the schemes we propose for our study. In Sect. 4 we carry out an empirical comparison of the proposed methods and the traditional migration of individuals by testing their behaviour when solving different problems. Finally, in Sect. 5 we summarize our work and present some future lines of research.

2 Island Based Genetic Algorithms

As mentioned before, island models split the population into several subpopulations or demes which evolve independently according to the original algorithm and, ocasionaly, interchange information in the shape of a set of individuals.

Despite their simplicity and the large amount of related bibliography [1,7,5], their functioning is not still fully understood. This lack of knowledge is mainly due to the fact that there are a lot of parameters which need to be set and have influence on the performance of the algorithms. Besides those required to set up the evolutionary algorithm which evolves in each island, such as population size, probability of crossover/mutation, etc. , it becomes necessary to specify the interaction among them. These are the main parameters used in order to do it:

- Number of islands: When real parallelism is used, this parameter is constrained by the available hardware, but when parallelism is simulated, this is a tunable parameter. It acquires more importance when the total population is fixed and must be divided into all islands.
- Percentage of migrating individuals: Determines the number of individuals that each island sends to the others.
- Gap between migrations: Is the number of generations elapsed between migration processes. The most common option consist of fixing it to a constant but some works use asynchronous communication [2].

- Topology: Defines how is the interconnection among islands (star, ring, grid, etc). One important fact in topologies is the degree of conectivity, i.e., the number of islands connected to a given one.
- Replacement policies: It is necessary to define how population inside an island is formed after a migration is carried out. The most common option lies on using elitism, that is, received individuals are added to current population and, afterwards, individuals with worst fitness are discarded.

There are studies which show that, for some problems, island models perform better than sequential genetic algorithms [21]. One of the causes for that is related with the division of the search in several independent threads which explore different regions of the space. However, this fact can not explain the performance of these algorithms by itself. Cantú-Paz et. al. [3] show, both teoretical and empirically, that the results obtained by a run of a genetic algorithm with a population P are better in average than those obtained with n runs of this algorithm with population P/n . This result is related with those showed in [12,15], where it is demostrated that the effectiveness of a genetic algorithm is determined by the size of its population. The bigger it is, the better chance of finding a better solution. However, the effort of the algorithm also increases and sometimes it is necessary a tradeoff. In the same line, [14], shows that favorable changes spread faster when size of the demes is small, however this rapid rise stops sooner than the rise in the sequential algorithm.

Besides these conclusions there must be also considered that, if the interaction among islands is too intense, the lost of diversity can lead algorithms to behave as single population GAs. Relative to this point, there is a work by Skolicky and De Jong [20], which studies the effects of varying the percentage of individuals migrated and the gap between migrations. Concerning with the percentage of individuas migrated, results suggest that fixing it up to 10%, as it is done in standard settings, is unnecessary, but not damaging. However, gap between migrations is a more determining parameter when trying to keep the diversity. Thus, if migration gap is very small, diversity is lost faster, and results become poor. In the other hand, using a big gap makes results improve in almost every case.

From all these studies, there can be concluded that communication is a key point in the performance of island based genetic algorithms and, although it has to exist, it also has to be fixed in such way that diversity is preserved.

3 Path Relinking Based Incorporation of Information

Considering what has been pointed above, communication among islands must satisfy two criteria:

1. It is important to interchange of information among islands so that none of them converge prematurely.
2. This interchange must not be too intense. Otherwise, the lost of diversity would make the algorithm behave like a single population genetic algorithm.

The migration of individuals is the traditional way to make such communication. When a subset of individuals arrives to an island, they replace some of the individuals present

on it according to the *replacement policy*. This way, and according with the principles of the GAs, features on this incoming individuals will soon spread among all the population.

Since there is important to keep diversity, too frequent and numerous migrations must be avoided. Otherwise, features of the incoming individuals would take over the population making it very similar to the one from which they come. In the other hand, given that features propagate according to the evolutionary principles, and due to the statistical nature of them, too sporadic and small migrations can make some of the features lost.

The aim of this work is designing a schema that preserves diversity. In order to do that, instead of mixing a subset of individuals with the subpopulation, and this way trying to merge features from incoming and resident solutions, we generate only an individual which takes as much advantage as possible of the incoming information. This way, the improvement resulting of incorporating the information is done from the first moment, whereas the population does not change significantly.

3.1 Path Relinking

Path Relinking [10] arises as a complement for other metaheuristics such as Tabu Search [9], Scatter Search [8] or GRASP [17]. Its main basis is that good solutions can be generated by merging features of two given solutions. In order to do that, it iteratively makes changes to one solution *starting solution* until it is transformed into another, called *guiding solution*. This sequence of changes is seen as a path across the search space which goes from one solution to the other and, along this path, some better solutions with features of both can be found. These techniques have been successfully applied in many problems as assignment [16], phylogenetic inference [6], etc.

3.2 Application to Island Models of GAs

Path Relinking is grounded in creating a new solution from two previously given. This solution should improve the quality of both parents and would be composed by parts of them.

In this work, a path relinking based approach is used to create a new solution from the best individual in each island and the individual received. Thus, instead of performing a *blind* crossover, it is tried to get a new individual which gathers the features of both parents in an optimal way. Afterwards, this individual replaces the worst one in the island.

One of the advantages of this approach is that, since only one individual is introduced to the island, the population inside is not affected the same way that it would be if mixing all the individuals with the incoming set. Thus, this *precise* way of communication helps to preserve diversity.

Moreover, the fact that a local search based algorithm is used to combine the individuals helps to improve the quality of solutions because it allows to a more efficient exploration of the search space.

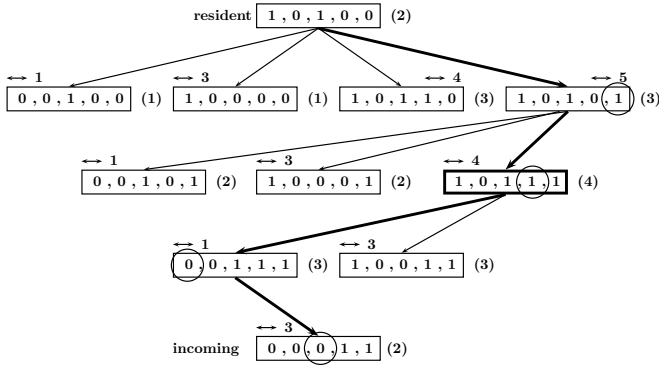


Fig. 1. Example of Path Relinking combination for OneMax problem (size=5)

```

function PathRelinkingCombination( $\mathcal{I}_i, \mathcal{I}_r$ ) {
     $best.fitness = -\infty$ 
     $\mathcal{I}_{aux} = \mathcal{I}_i.copy$ 
     $changes = \{j \mid \mathcal{I}_i(X_j) \neq \mathcal{I}_r(X_j)\}$ 
     $stage = 0$ 
    while ( $changes \neq \emptyset$ ) do
    {
         $fitness = new\ array[changes.length]$ 
        for  $i = 1$  to  $changes.length$  do
             $\mathcal{I}_{aux}(X_{changes[i]}) = \mathcal{I}_r(X_{changes[i]})$ 
             $fitness[i] = evaluate(\mathcal{I}_{aux})$ 
             $\mathcal{I}_{aux}(X_{changes[i]}) = \mathcal{I}_i(X_{changes[i]})$ 
         $i^* = \arg\ max_i\ fitness[i]$ 
         $\mathcal{I}_{aux}(X_{changes[i^*]}) = \mathcal{I}_r(X_{changes[i^*]})$ 
        if  $fitness[i^*] > best.fitness$  then
             $best.fitness = fitness[i^*]$ 
             $best.individual = \mathcal{I}_{aux}.copy$ 
         $changes = changes \setminus \{i^*\}$ 
    }
    return  $best.individual$ 
}

```

Fig. 2. Combination of models through path relinking

Figure 1, shows an example of path relinking generation of a new solution for the OneMax problem with size $n = 5$. On it, we take the best resident individual as the *starting solution*. As it can be seen, in the first steps there can be done 4 different changes because there are 4 positions which value is different. Each change leads to a new individual whose fitness is specified between parenthesis. In this case, the change of positions 4 and 5 result in an individual with the same fitness (3), so we can decide

one of them randomly. In this case, position 5 is changed. In the next step, only 3 changes can be done. The change of position 4 leads to an individual with fitness 4, so it is the choosen change. After that, there are only two possible changes and, although none of them leads to an individual better than the original one, one must be choosen. Finally, we do the only possible change. The path followed to go from the first solution to the *guiding solution* is marked in bold, and so it is the best individual found. The pseudocode of this process is described in Fig. 2.

Concerning to this process, there are two remarks which are worth to be done:

- During the search we can accept moves that lead to worse configurations.
- The number of evaluations required by PR is $\frac{(k+1)k}{2}$, i.e., $O(k^2)$, k being the number of coordinates with different value between $\mathcal{I}_{resident}$ and $\mathcal{I}_{incoming}$. Thus, in the worst case $k = n$, but in practice and when the evolution converges $k \ll n$. In the example of Fig. 1, $n = 5$ and $k = 4$.

4 Experimental Study

In this section we perform an experimental study of our proposal. In order to do that, we have choosen a set of test functions and have solved them with both traditional and path relinking based island models under several parameter configurations. We show the results of the comparison between both schemes as well as a study of the influence of parameters in the proposed approach.

4.1 Test Problems

In order to get a broader vision over the behaviour of the models, we have tried to cover a wide range of problems by choosen eight functions frequently used in literature relative to combinatorial optimization:

• *Massively Multimodal Deceptive Function*

This problem can be defined matematically as:

$$f_{mmdp}(\mathbf{x}) = \sum_{j=1}^m f_b^6(\mathbf{s}_j)$$

with $\mathbf{s}_j = (x_{6j-5}, x_{6j-4}, x_{6j-3}, x_{6j-2}, x_{6j-1}, x_{6j})$, $n = 6m$ and

$$f_b^6(\mathbf{s}_j) = \begin{cases} 1.000000 & \text{if } \#ones(s_j) = 0 \\ 0.000000 & \text{if } \#ones(s_j) = 1 \\ 0.360684 & \text{if } \#ones(s_j) = 2 \\ 0.640576 & \text{if } \#ones(s_j) = 3 \\ 0.360684 & \text{if } \#ones(s_j) = 4 \\ 0.000000 & \text{if } \#ones(s_j) = 5 \\ 1.000000 & \text{if } \#ones(s_j) = 6 \end{cases}$$

In this work, we have used $m = 15$ so the size is $n = 90$ and the optimum is also 15.

- *CheckerBoard* function

In this problem, a $s \times s$ grid is given. Each point of the grid can take a value 0 or 1. The goal of the problem is to create a checkerboard pattern of 0's and 1's on the grid. The evaluation counts, for each position except the borders, how many of the bits in the four basic directions has the opposite value to it. If we consider the grid as a matrix $x = [x_{ij}]_{i,j=1,\dots,s}$ and interpret $\delta(a, b)$ as the Kronecker's delta function, the *CheckerBoard* function can be written as:

$$F_{cb}(x) = 4(s-2)^2 - \sum_{i=2}^{s-1} \sum_{j=2}^{s-1} \{\delta(x_{ij}, x_{i-1j}) + \delta(x_{ij}, x_{i+1j}) + \delta(x_{ij}, x_{ij-1}) + \delta(x_{ij}, x_{ij+1})\}$$

The maximum value is $4(s-2)^2$. We use $s = 10$, so dimension (n) is 100 and the optimum is 256.

- *SixPeaks* function

This problem can be defined mathematically as:

$$F_{sp}(x, p) = \max\{t(0, x), h(1, x), t(1, x), h(0, x)\} + R(x, p)$$

$$\text{with } \begin{cases} t(b, x) = \text{number of trailing b's in } x. \\ h(b, x) = \text{number of leading b's in } x \\ R(x, p) = \begin{cases} n & \text{if } t(0, x) > p \text{ and } h(1, x) > p \text{ or} \\ & t(1, x) > p \text{ and } h(0, x) > p \\ 0 & \text{otherwise} \end{cases} \end{cases}$$

We have taken dimension (n) to be 100 and p to be 30. This problem is characterized because it has 4 global and 2 local optima. In our setting the optimum value is 169.

- *Colville* function

Consists of the optimization of the following function:

$$F_c(x_1, x_2, x_3, x_4) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2 + 10.1((x_2 - 1)^2 + (x_4 - 1)^2) + 19.8(x_2 - 1)(x_4 - 1)$$

where $-10 \leq x_i \leq 10$. In our setting, each x_i is represented by 25 binary variables ($n = 100$) using the method described in [19]. We have also tested the function using a gray encoding. The minimum value for F_c is 0.

- *EqualProducts* function

Given a set of n random real numbers a_1, a_2, \dots, a_n from an interval $[0, k]$, a subset of them is selected. The aim of the problem is to minimize the difference between the products of the selected and unselected numbers. It can be written as follows:

$$F_{ep}(x) = abs(\prod_{i=1}^n h(x_i, a_i) - \prod_{i=1}^n h(1 - x_i, a_i))$$

$$\text{with } h(x, a) = \begin{cases} a & \text{if } x = 1 \\ 1 & \text{if } x = 0 \end{cases}$$

We have taken dimension to be 100. The numbers have been generated randomly from a uniform distribution in $[0, 4]$, therefore, we don't know the optimum for this problem. However, values close to zero are better.

- *Rastrigin* function

Consists of the optimization of the following function:

$$F_r(x) = 10k + \sum_{i=1}^k (x_i^2 - 10\cos(2\pi x_i))$$

where $-5.12 \leq x_i \leq 5.11$. In our setting, we have taken $k = 10$ and each x_i is represented by 10 binary variables ($n = 100$) also using the method described in [19]. As it happened with the Colville function, we have also tested the models with a *gray* version of this problem. The minimum value for F_r is 0.

In order to use our software, we have converted the minimization problems in maximization ones by multiplying the results of the evaluation functions by -1 .

4.2 Experiments

In order to compare, we have run the algorithms over the problems described. As it was mentioned in Sect. 2 the amount of parameters to take into account is huge. In all our experiments, the basis is a genetic algorithm that uses roulette wheel selection with rank assignation of probabilities. The uniform crossover is used with probability 0.75, whereas the probability of mutation has been fixed to 0.05. Concerning with the parameters relatives to the island model, they are part of the study, so we have tested the algorithms over several configurations. In fact, we have made combinations with populations 256 and 2048, with 8, 16 and 32 islands, and with 5 and 20 generations between migrations. In the case of the individual migration, 10% of the population is migrated and, in both cases, a unidirectional ring is used as topology.

Each configuration has been evaluated by running the algorithm 50 times. In order to show the results obtained, we have splitted them into two tables, Tab. 1 and Tab. 2, depending on the population size, since it is, a priori, the parameter which affects the performance in a more notable way.

For both algorithms, each table shows the results, *mean fitness* \pm *deviation* and *average number of evaluations* \pm *deviation* for 50 runs of each configuration *number of islands / generations between migrations*. In order to determine if one of the schemes outperforms the other we have marked with \bullet , also for each configuration *number of islands / generations between migrations* the algorithm with the best average fitness. Then, we have performed a Mann-Whitney unpaired test with the results of the other algorithm. If they do not present a significant difference ($p\text{-value} > 0.05$) we have also marked it. We have proceeded the same way with the number of evaluations marking the outstanding algorithm (or both in case there is no difference) with \circ .

4.3 Analysis of the Results

Comparison between the two approaches

As it can be seen in both tables, if considering the fitness, in most cases path relinking based island models either outperform or have no difference with the conventional

Table 1. Comparison between migration of individuals and path relinking based communication for population 256

Islands/Generations	Migration of Individuals		Path Relinking Based Migration	
MMDP (90)				
8/5	14.5830±0.2846	348600.32±88047.82	14.7053±0.2483	324955.56±115761.72
8/20	14.6981±0.2553	397998.08±80033.12	14.9137±0.1551	234908.14±118693.00
16/5	14.5543±0.2469	370933.76±101981.47	14.7915±0.2418	227134.50±147511.51
16/20	14.5646±0.2492	434017.28±62168.07	14.9137±0.1712	136473.36±63394.86
32/5	12.9335±0.3379	380948.48±98495.66	14.9497±0.1625	186613.26±47819.77
32/20	12.9207±0.3589	385418.24±110377.44	14.9856±0.0711	196161.3±40480.75
EqualProducts (100)				
8/5	• -2.7990±3.0027	270202.88±158904.72	• -2.6304±2.2877	222647.32±123096.80
8/20	• -3.2392±2.7974	234782.72±144861.33	• -2.1236±1.7621	260450.72±162505.45
16/5	• -2.7768±2.7331	231956.48±147734.82	• -2.6615±2.2461	279685.16±130416.60
16/20	• -2.7604±2.5018	263383.04±133865.55	• -2.6596±2.2927	264771.76±149645.56
32/5	• -1.9804±2.1356	259635.20±156869.79	• -6.0034±5.5914	317913.34±136652.68
32/20	• -1.9431±1.7147	239943.68±143171.51	• -2.8625±3.1519	240761.24±140890.08
Rastrigin Binary (100)				
8/5	• -1.1516±1.3371	288942.08±131166.18	• -0.3897±0.5408	203520.06±116273.58
8/20	• -0.8048±0.8871	328002.56±115835.23	• -0.3940±0.6102	232652.12±83704.11
16/5	• -0.9208±1.0198	285255.68±108747.01	• -0.2681±0.5034	224010.52±55749.64
16/20	• -0.6571±0.7925	338539.52±84873.48	• -0.1131±0.1762	302775.06±70792.57
32/5	• -2.5162±0.7731	489113.60±31475.57	• -0.1739±0.3048	404879.44±60740.21
32/20	• -2.1851±0.8295	488099.84±26986.43	• -0.1050±0.1174	418450.18±64635.92
Rastrigin Gray (100)				
8/5	• 0±0	156907.52±15333.14	• 0±0	115808.00±25271.43
8/20	• 0±0	227578.88±27431.47	• 0±0	152449.66±27670.64
16/5	• 0±0	190330.88±17441.02	• 0±0	137076.46±21859.86
16/20	• 0±0	270694.40±26021.49	• 0±0	180153.50±19944.87
32/5	• -0.1943±0.2410	497285.12±14294.35	• 0±0	211106.00±20351.73
32/20	• -0.1373±0.1720	496056.32±18636.94	• 0±0	251844.50±22303.36
Colville Binary (100)				
8/5	• -1.4577±1.5350	195614.72±131767.99	• -1.2042±1.1947	235457.64±148028.89
8/20	• -0.9285±0.9011	249620.48±150818.61	• -1.1774±0.9601	223974.54±134692.02
16/5	• -1.3319±1.3460	238192.64±136589.15	• -0.9929±0.8764	249262.96±107367.58
16/20	• -0.7792±0.9868	272844.80±173142.70	• -0.6362±0.7028	272017.42±163028.87
32/5	• -0.2109±0.2181	342579.20±194829.34	• -0.6517±0.5603	417338.88±93352.00
32/20	• -0.2387±0.2643	339399.68±193159.83	• -0.4946±0.5144	408454.00±140235.69
Colville Gray (100)				
8/5	• -0.0857±0.0237	107786.24±129853.74	• -0.0741±0.0300	100765.96±141414.30
8/20	• -0.0701±0.0311	88770.56±133544.40	• -0.0638±0.0355	93860.72±143183.60
16/5	• -0.0741±0.0284	63795.20±107848.75	• -0.0721±0.0269	104084.12±89245.62
16/20	• -0.0627±0.0301	67696.64±118544.77	• -0.0601±0.0270	61652.66±41015.53
32/5	• -0.0818±0.0174	41093.12±26587.71	• -0.0695±0.0305	162622.58±98758.38
32/20	• -0.0644±0.0239	31569.92±17574.95	• -0.0545±0.0227	97996.28±49382.38
SixPeaks (100)				
8/5	100.00±0.00	195322.88±26814.02	127.60±34.14	156526.70±60088.21
8/20	100.00±0.00	290316.80±45491.10	160.72±22.64	137461.20±58310.94
16/5	100.00±0.00	230727.68±30393.83	155.20±27.88	218604.44±68474.03
16/20	100.00±0.00	336442.88±50446.71	167.62±9.75	176442.66±45867.25
32/5	88.70±2.80	480051.20±24383.73	167.40±11.31	380687.86±52336.71
32/20	90.64±2.56	483153.92±32143.16	169.00±0.00	288755.66±30062.80
CheckerBoard (10)				
8/5	252.60±6.58	114636.80±95164.73	255.36±3.16	79326.22±52984.13
8/20	254.80±3.89	147153.92±90076.21	256.00±0.00	90463.68±52554.50
16/5	253.26±5.98	125419.52±94032.60	256.00±0.00	93462.96±35789.29
16/20	• 255.72±1.97	157153.28±62760.59	256.00±0.00	105927.24±30647.58
32/5	• 256.00±0.00	288972.80±62746.02	256.00±0.00	147299.50±35853.45
32/20	• 256.00±0.00	263275.52±59086.01	256.00±0.00	155980.32±35559.56

Table 2. Comparison between migration of individuals and path relinking based communication for population 2048

Islands/Generations	Migration of Individuals		Path Relinking Based Migration	
MMDP (90)				
8/5	13.2394±0.3127	483819.52±33233.74	14.5543±0.3671	351040.90±113684.99
8/20	12.8072±0.3097	470179.84±48758.74	14.6365±0.3887	353733.74±107999.50
16/5	13.2561±0.3371	473006.08±36705.67	14.8490±0.2306	258519.72±123910.30
16/20	12.8182±0.3371	483082.24±38896.90	14.9353±0.1394	384272.78±63986.01
32/5	13.2193±0.2764	473989.12±31676.61	14.9784±0.0862	286278.52±46301.43
32/20	12.8782±0.3040	476200.96±45680.96	14.9784±0.0862	431824.44±65290.12
EqualProducts (100)				
8/5	• -2.6050±2.6289	○ 244695.04±133108.25	• -2.8395±3.0815	○ 261563.10±153509.57
8/20	• -2.1472±2.0913	○ 277504.00±155735.55	• -2.4363±2.4389	○ 270838.18±147109.92
16/5	• -2.5523±2.0771	○ 245432.32±142048.99	• -3.6125±2.9426	○ 278107.60±157970.75
16/20	• -2.6721±2.2210	○ 284385.28±139188.38	• -2.7731±3.4226	○ 278578.44±169746.73
32/5	• -2.0967±2.1436	○ 263372.80±140346.48	• -3.1402±3.1902	○ 287492.38±158421.99
32/20	• -2.0584±1.9206	○ 257720.32±151984.23	• -2.5525±2.8202	○ 263272.56±128966.65
Rastrigin Binary (100)				
8/5	-3.6454±1.2535	504463.36±14537.23	• -1.2101±0.9535	○ 454775.32±69957.68
8/20	-5.3319±1.1518	506306.56±20726.32	• -2.6805±1.4841	○ 513674.20±31202.25
16/5	-3.6326±1.3799	○ 504709.12±11256.66	• -0.9483±0.9502	○ 482099.92±49544.14
16/20	-5.1607±1.3001	○ 505446.40±19429.03	• -2.5594±1.1254	○ 513418.92±32092.35
32/5	-3.4138±1.2269	○ 504217.60±13368.94	• -1.4274±0.8422	○ 495956.36±32053.61
32/20	-4.6350±1.2230	○ 502865.92±19056.30	• -2.5840±1.0567	○ 537505.54±26168.70
Rastrigin Gray (100)				
8/5	-1.5139±0.6285	○ 508395.52±9358.26	• -0.3761±0.5237	○ 473077.80±64477.56
8/20	-3.5943±1.0168	○ 508887.04±14117.83	• -1.1284±0.8773	○ 532812.60±20764.47
16/5	-1.3489±0.6852	507412.48±9022.97	• -0.0590±0.2188	○ 431781.50±58804.97
16/20	-3.4713±0.9553	○ 504586.24±19984.53	• -1.1759±0.5910	○ 513251.70±28243.97
32/5	-1.9471±0.6323	507412.48±9681.94	• -0.0839±0.2296	○ 464009.64±48043.48
32/20	-3.2424±0.7233	○ 503726.08±18554.47	• -1.2977±0.6209	○ 529135.88±16129.17
Colville Binary (100)				
8/5	• -0.6272±0.7975	○ 393871.36±191563.67	• -0.6499±0.6550	○ 375117.80±189723.37
8/20	• -0.5242±0.6440	○ 351477.76±202340.11	• -0.3663±0.4549	○ 324988.58±209438.71
16/5	• -0.4830±0.7335	○ 326287.36±209242.96	• -0.3623±0.4525	○ 309570.06±195951.66
16/20	• -0.2749±0.3503	○ 307609.60±203216.92	• -0.1846±0.2887	○ 295473.22±207314.68
32/5	• -0.3642±0.6001	○ 302694.49±217743.91	• -0.2470±0.3109	○ 374955.70±167322.59
32/20	• -0.2009±0.2947	○ 290406.40±207445.31	• -0.1226±0.1258	○ 306726.14±199542.50
Colville Gray (100)				
8/5	• -0.0561±0.0350	○ 79790.08±119152.35	• -0.0507±0.0385	○ 72542.64±99302.76
8/20	• -0.0448±0.0312	○ 52142.08±22674.60	• -0.0386±0.0312	○ 74577.64±75386.88
16/5	• -0.0565±0.0295	○ 54968.32±67430.39	• -0.0557±0.0273	○ 71360.30±62760.78
16/20	• -0.0357±0.0319	○ 65044.48±67951.64	• -0.0341±0.0269	○ 64487.80±31340.53
32/5	-0.0602±0.0261	○ 42926.08±16013.89	• -0.0452±0.0286	○ 107757.52±51523.22
32/20	• -0.0388±0.0300	○ 63324.16±24941.91	• -0.0294±0.0255	○ 86864.14±37876.30
SixPeaks (100)				
8/5	77.62±3.31	494141.44±19509.72	144.18±32.67	○ 410826.72±100043.69
8/20	69.58±2.74	492544.00±30868.58	165.82±4.88	○ 391367.18±78143.93
16/5	76.86±3.09	491683.84±20095.08	157.70±26.01	○ 377348.98±79327.95
16/20	69.49±2.47	496844.80±30564.45	168.78±1.07	○ 439887.14±60998.31
32/5	75.36±2.87	○ 496107.52±21529.59	167.38±3.43	○ 479291.80±41000.15
32/20	68.42±2.30	○ 495984.64±27600.17	167.34±3.69	○ 511246.88±45512.21
CheckerBoard (10)				
8/5	• 254.76±3.99	○ 418201.60±52310.59	• 255.22±3.16	○ 231343.44±114141.48
8/20	255.26±0.59	476200.96±42246.42	• 255.90±0.70	○ 322663.20±75464.78
16/5	• 255.98±0.14	○ 411074.56±44816.06	• 255.98±0.14	○ 193736.98±72189.86
16/20	254.82±1.62	473620.48±39254.34	• 256.00±0.00	○ 320660.16±74119.55
32/5	255.60±1.56	434667.52±45991.55	• 255.98±0.14	○ 236553.98±72595.27
32/20	254.62±0.75	485662.72±36462.45	• 256.00±0.00	○ 337724.56±66154.77

island model. In fact, for a population of 256 individuals and the problems *MMDP*, *Rastrigin* (Binary) and *SixPeaks*, the improvement happens regardless of the configuration, whereas there is only three cases (*EqualProducts* with 32 islands and a gap of 5 generations and binary *Colville* with 32 islands) where the situation is the opposite. In the rest of cases, there is no difference. The tendency is the same when using 2048 individuals since the path relinking based approach outperforms the migration of individuals, regardless of the configuration, in 4 out of 8 problems. Here, the three only cases when individuals outperform the path relinking approach take place when solving the *EqualProducts* function.

Concerning with the number of evaluations, tendency is the same. Thus, for a population of 256, the path relinking approach uses less evaluations in every case when solving the *MMDP*, *SixPeaks*, *CheckerBoard* and both *Rastrigin* functions. In the case of *Colville* (gray) the migration of individuals outperforms the path relinking based communication 4 times. In the rest of cases, there are no significant difference. For population 2048 things are slightly different. As it happened before, it depends on the problem. For *MMDP* and *Checkerboard*, the path relinking algorithm uses less evaluations in every case. In *Colville* (gray) it seems that individual migration performs better, and in the rest of problems, things become similar for both algorithms.

In general, it can be said that path relinking approach outperforms the traditional schema of migration since, for population 256 results are better (if not in fitness when considering the evaluations as well) in 5 out of 8 problems where there are similar results for the other three. This tendency is also found when using a population of 2048 individuals. In this case, the algorithm outperforms the other in 5 out of 8 problems, although for the *EqualProducts* and the *Colville* function things seem different.

Analysis of the parameters

In order to get more information about the behaviour of the proposed model, we have studied the impact of each one of the three parameters considered on the fitness achieved by the algorithm.

- Population size

To study the influence of the populations size on the results we have compared, for each configuration (*problem-number of islands-generations*), the results obtained for the populations 256 and 2048. Results can be seen in Table 3. Mann-Whitney has been also used to compare the two results for each configuration. For each problem, either the best result in each problem or both, if there is no difference ($p - value > 0.05$), have been marked with a ●.

As it can be seen, results are fairly dependent on the problem. Whereas for both *Rastrigin* versions and *SixPeaks* the smaller population outperforms in every case, in the *Colville* function a big population works better. In *CheckerBoard* and *Equalproducts* it seems not to be difference whereas for the problem *MMDP*, it seems that a big population is better with more islands are used.

- Number of islands

In the case of the number of islands we have proceeded the same way. For each configuration (*problem-size of population-generations*) we have compared the results

Table 3. Comparison among populations for each configuration (*Number of islands-generations-problem*)

Population	5 generations			20 generations		
	8 Islands	16 Islands	32 Islands	8 Islands	16 Islands	32 Islands
MMDP (90)						
256	•			•		
2048		•	•		•	•
EqualProducts (100)						
256	•	•	•	•		•
2048	•	•	•	•	•	•
Rastrigin Binary (100)						
256	•	•	•	•	•	•
2048						
Rastrigin Gray (100)						
256	•	•		•	•	•
2048						
Colville Binary (100)						
256						
2048	•	•	•	•	•	•
Colville Gray (100)						
256						
2048	•	•	•	•	•	•
SixPeaks (100)						
256	•	•	•	•	•	•
2048	•		•	•		
CheckerBoard (100)						
256	•	•	•	•	•	•
2048	•	•	•	•	•	•

Table 4. Comparison among number of islands used for each configuration (*problem-population-generations*)

Generations	Population 256			Population 2048		
	8 Islands	16 Islands	32 Islands	8 Islands	16 Islands	32 Islands
MMDP (90)						
5			•			•
20			•		•	•
EqualProducts (100)						
5	•	•		•	•	•
20	•	•	•	•	•	•
Rastrigin Binary (100)						
5	•	•	•	•	•	
20		•	•	•	•	•
Rastrigin Gray (100)						
5	•	•	•		•	•
20	•	•	•	•	•	•
Colville Binary (100)						
5		•	•		•	•
20		•	•		•	•
Colville Gray (100)						
5	•	•	•	•	•	•
20	•	•	•	•	•	•
SixPeaks (100)						
5			•		•	•
20		•	•		•	
CheckerBoard (100)						
5		•	•		•	•
20	•	•	•	•	•	•

when varying the number of islands. The comparison has been carried out the same way as above. Results can be seen in the Table 4. The best result is marked with a • as well as those which not present significative difference.

Results are clearer for this parameter since for almost all the configurations, using 32 islands is either the outstanding or among the outstanding configurations. Although the difference with using 16 islands is minimum, it is more remarkable when comparing with 8 islands.

• Gap between migrations

Results of the comparisons for the gap between migrations can be seen in Table 4.3. We have compared the two intervals for each configuration (*problem-size of population-number of islands*). In this case, the behaviour of the parameter slightly depends on the

Table 5. Comparison of the number of generations for each configuration (*problem-population-number of islands*)

Generations	Population 256			Population 2048		
	8 Islands	16 Islands	32 Islands	8 Islands	16 Islands	32 Islands
MMDP (90)						
5			•	•		•
20	•	•	•	•	•	•
EqualProducts (100)						
5	•	•		•		
20	•	•	•	•	•	•
Rastrigin Binary (100)						
5	•	•	•	•	•	•
20	•	•	•			
Rastrigin Gray (100)						
5	•	•	•	•	•	•
20	•	•	•			
Colville Binary (100)						
5	•	•				
20	•	•	•	•	•	•
Colville Gray (100)						
5	•			•		
20	•	•	•	•	•	•
SixPeaks (100)						
5			•			•
20	•	•	•	•	•	•
CheckerBoard (100)						
5		•	•	•	•	•
20	•	•	•	•	•	•

population size. For 256 and each configuration, using 20 generations seems to be the best option, since it is always either the best or similar to the best option. When working with 2048 individuals, results are almost the same except for the *Rastrigin* function. In this case, it seems better to use 5 generations.

In general, results confirm the benefits of preserving diversity because the best option seems to be choose 32 islands in every case even preserving the gap of 20 generations between migrations.

5 Conclusions and Future Work

In this work, we have presented a method to improve the performance of island based models based in path relinking. Results show that this method improves the results of the traditional scheme of migration in almost all cases, and regardless of the parameter setting.

Although the communication is not so intense as it is when migrating individuals in terms of producing changes in the population inside the islands, it keeps on being necessary to preserve diversity by using a big number of islands. Moreover, if using smaller populations, it can be beneficial using big gaps between migrations.

As future work, we plan to use a disperse initialization of the islands, as well as trying other migration schemes but synchronous migrations, as for instance, doing the migrations only when detecting the convergence.

References

1. E. Alba, C. Cotta, and J. M. Troya. Numerical and real time analysis of parallel distributed gas with structured and panmictic populations. In *Proceedings of the IEEE Conference on Evolutionary Computing (CEC)*, volume 2, pages 1019–1026, 1999.
2. Enrique Alba and Jose M. Troya. An analysis of synchronous and asynchronous parallel distributed genetic algorithms with structured and panmictic islands. In *IPPS/SPDP Workshops*, pages 248–256, 1999.
3. E. Cantú-Paz and D.E. Goldberg. Are multiple runs of genetic algorithms better than one? In *Proceedings of the Genetic and Evolutionary Computation Conference 2003*, 2003.
4. E. Cantú-Paz. A survey of parallel genetic algorithms. Technical Report IlliGAL-97003, Illinois Genetic Algorithms Laboratory. University of Illinois at Urbana-Champaign, 1997.
5. E. Cantú-Paz. *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer Academic Publishers, 2001.
6. C. Cotta. Scatter search with path relinking for phylogenetic inference. *European Journal of Operational Research*, 169(2):520–532, 2006.
7. D. Whitley, S. Rana, and R.B. Heckendorn. The island model genetic algorithm: On separability, population size and convergence. *Journal of Computing and Information Technology*, 1(7):33–47, 1999.
8. M. Laguna F. Glover and R. Martí. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 3(29):653–684, 2000.
9. F. Glover. Tabu search - part i. *ORSA Journal of Computing*, 1:190–206, 1989.
10. F. Glover. Scatter search and path relinking. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 297–316. McGraw-Hill, 1999.
11. D.E. Goldberg. *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley, New York, 1989.
12. D.E. Goldberg, K. Deb, and J.H. Clark. Genetic algorithms, noise, and the sizing of populations. *Complex Systems*, 6:333–362, 1992.
13. J.J. Grefenstette. Parallel adaptive algorithms for function optimization. Technical Report CS-81-19, Computer Science Department, Vanderbilt University, Nashville, TN., 1981.
14. P.B. Grosso. *Competent simulations of genetic adaptation: Parallel subcomponent interaction in a multilocus model*. PhD thesis, University of Michigan, 1985.
15. G. Harik, E. Cantú-Paz, D.E. Goldberg, and B. Miller. The gamblers ruin problem, genetic algorithms, and the sizing of populations. In *Proceedings of the Fourth International Conference on Evolutionary Computation*, pages 7–12. IEEE Press, 1997.

16. P. Tolla L. Alfandari, A. Plateau. *Metaheuristics: Computer Decision-Making*, chapter A path-relinking algorithm for the generalized assignment problem, pages 1–18. Kluwer Academic Publishers, 2004.
17. M. Laguna and R. Martí. Grasp and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11(1):44–52, 1999.
18. José A. Gámez Luis de la Ossa and José M. Puerta. Improving model combination through local search in parallel univariate edas. In *IEEE Congress on Evolutionary Computation , CEC2005*, volume 2, pages 1426–1433, Edinburgh, Scotland, September 2005. IEEE Press.
19. Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 1996.
20. Z. Skolicki and K. De Jong. The influence of migration sizes and intervals on island models. In *Proceedings of Genetic and Evolutionary Computation Conference, GECCO 2005*, pages 1295–1302. ACM Press.
21. D. Whitle, S.Rana, and R.B. Heckendorn. Island model genetic algorithms and linearly separable problems. In *Selected Papers from AISB Workshop on Evolutionary Computing*, volume 1305 of *LNCS*, pages 109–125. Springer Verlag, 1997.

Using Datamining Techniques to Help Metaheuristics: A Short Survey

Laetitia Jourdan¹, Clarisse Dhaenens², and El-Ghazali Talbi^{1,2}

¹ INRIA Futurs, Bât M3, Cité Scientifique 59655 Villeneuve d'Ascq
France

² LIFL, CNRS, Université de Lille I, 59655 Villeneuve d'Ascq
France

{jourdan, dhaenens, talbi}@lifl.fr

Abstract. Hybridizing metaheuristic approaches becomes a common way to improve the efficiency of optimization methods. Many hybridizations deal with the combination of several optimization methods. In this paper we are interested in another type of hybridization, where datamining approaches are combined within an optimization process. Hence, we propose to study the interest of combining metaheuristics and datamining through a short survey that enumerates the different opportunities of such combinations based on literature examples.

1 Introduction

Hybrid metaheuristics are more and more studied and a first taxonomy has been proposed in [44]. Many works propose to combine two or more metaheuristics, but other works present also hybridizations between exact optimization methods and metaheuristics. Another promising approach to hybridization is to use datamining techniques to improve metaheuristics. Datamining (DM), also known as Knowledge-Discovery in Databases (KDD), is the process of automatically exploring large volumes of data e.g., instances described according to several attributes, to discover patterns. In order to achieve this, datamining uses computational techniques from statistics, machine learning, pattern recognition or combinatorial optimization.

Datamining tasks can be organized into a taxonomy, based on the desired outcome of the algorithm. Usually a distinction is made between supervised and unsupervised learning. Classical tasks of supervised learning are:

- Classification: examining the attributes of a given instance to assign it to a predefined category or class.
- Classification rule learners: discovering a set of rules in the database which forms an accurate classifier.

The most common tasks of unsupervised learning are:

- Clustering: partitioning a data set into subsets (clusters), so that data in each subset share some common aspects. Partitioning is often indicated by a proximity evaluated using a distance measure.

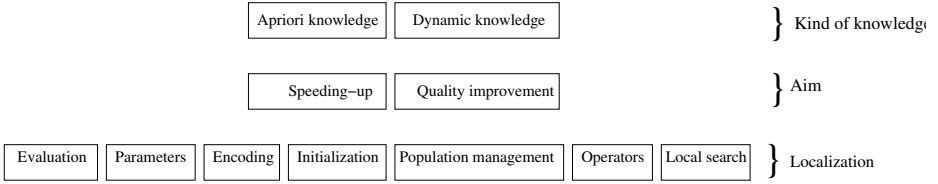


Fig. 1. The proposed taxonomy

- Association rule learners: discovering elements that occur in common within a given data set.

Using metaheuristics for knowledge extraction has become common while the other way which consists in using knowledge discovery to improve metaheuristic is less studied. This research way can be refereed as knowledge incorporation in metaheuristics and may be performed by using informed operators, approximation of fitness, etc.

To illustrate the different ways to integrate knowledge into metaheuristics, a small taxonomy that summarizes compositions found in several articles, is proposed in Figure 1.

- Two kinds of knowledge can be distinguished: a previously acquired knowledge which is called *Apriori Knowledge* and a dynamically acquired knowledge which is extracted or discovered during the search.
- Another useful information to classify algorithms is to distinguish the aim of the cooperation. Either the cooperation is used to reduce the computational time *i.e.*, speed up techniques, by simplification of the fitness *i.e.*, fitness approximation, or by significantly reducing the search space *e.g.*, leading the metaheuristic in promising area; or the cooperation is used to improve the quality of the search by introducing knowledge in operators or in other parts of the metaheuristic. In fact, the insertion of datamining techniques often leads to both speeding up the metaheuristic and improving the quality.
- The last point used to distinguish the hybridizations is to determine which part of the metaheuristic is concerned by the knowledge incorporation. Hybridization can occur in each part of the metaheuristic: parameters, encoding, evaluation, initialization, operators, etc.

This paper aims to provide a quick comprehensive picture of the interest of combining datamining techniques and metaheuristics. We do not consider here the vast topic of incorporating knowledge but the use of knowledge algorithms also called datamining algorithms. In order to present this literature review, we have chosen to classify references with respect to the localization of the knowledge integration.

The remainder of this paper is set out as follows. Section 2 highlights the potential of datamining to speed-up metaheuristics by using datamining during

the evaluation. Section 3 discusses how datamining can help to set the parameters of the metaheuristic. Section 4 presents the use of datamining techniques for the initialization of metaheuristics. Section 5 is devoted to population management. Section 6 details the benefit of datamining in crossover operators. Section 7 shows the local search datamining applications in evolutionary computation. Section 8 exhibits that some metaheuristics are based on datamining incorporation. Finally, conclusions and perspectives are drawn in the last section.

2 Using Datamining During the Evaluation

In some real cases, the fitness function can be very expensive to compute. Thus decreasing the number of complete evaluations would be beneficial. To achieve this, some approximations of the fitness functions could be used, and datamining techniques may be interesting to obtain good approximations. A very complete and comprehensive survey on fitness approximation has been proposed in [15]. It shows that fitness approximation can be used either for expensive fitness functions or multi-modal fitness functions and may be realized by several approaches exposed below.

2.1 Replace the Evaluation Function by a Datamining Algorithm

Datamining techniques can be used to build approximate models of the fitness function. In this context, previously calculated fitnesses are learned by a datamining algorithm to approximate the fitness of new individuals. Many works use neural networks (both multi-layer perceptrons and radial-basis-function networks) to realize an approximation of the function to optimize. For example in [6], the authors use an artificial neural network (ANN) with a multiple objective genetic algorithm. They evaluate a large part of the population with an ANN and a small part is still simultaneously evaluated with the original function. Rasheed et al. propose to cluster data and to construct separate approximation models for the different clusters [30,32,29]. The approximation model can be used each time or alternatively with the real objective function.

2.2 Using Datamining Techniques to Avoid Evaluations

When the fitness function can be approximated, some authors use the approximation only within operators, such as initialization, mutation, crossover and selection. This approach avoids the calculation of time consuming fitness for individuals that may be of very bad quality and that will not be kept in the population. For example, in [31], the authors use approximations to improve operators. They generate several possible new individuals and then choose the best according to a reduced model. To compute the model, they maintain a large sample of points encountered during the course of the optimization and divide it into dynamic clusters. To compute the approximate model of an individual, they use the weighted k-nearest-neighbor approach which is a classification technique.

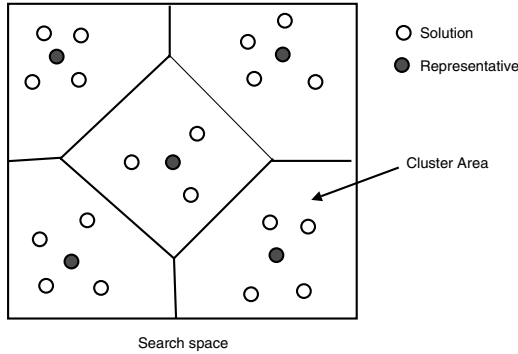


Fig. 2. Using representatives for the evaluation process through clustering

2.3 Using a Datamining Algorithm to Estimate a Representative

Approximating the fitness could be not satisfying because of the quality of the approximation for example. Another way to speed up the metaheuristic is to reduce the number of calls to the fitness function. This may be realized using fitness imitation. In this kind of approach, a set of individuals is considered as similar to another one and their fitness is fixed as equal to the reference individual which is called the representative. To determine the different sets, clustering techniques are often used.

Hence, in [16,18,47], the author proposes to maintain a large population size by using clustering algorithms. For example, in [47] Yoo et al. propose to use a fuzzy clustering approach to divide the population and to elect a representative of each cluster. Only the representatives are evaluated which allows the reduction of the evaluation costs. The fitness value of an individual of a cluster is estimated in respect with its associated representative (Figure 2).

3 How Datamining May Help to Set Parameters

A very difficult part in designing metaheuristics deals with the setting of the parameters of such methods. How can we fix in advance parameters such as the probability of application of a given operator, the size of the population or the number of iterations, for example? Two approaches may be used in this context:

- A first approach which is empirical consists in both executing several times the method with different parameter values and trying to select the best values. If the number of executions or the number of parameters are high, determining the best set of parameters may require statistical analyses. This may be seen as a datamining help.
- To set the probability of application of an operator, another approach may be used. It consists in analyzing the performance of the operators during the algorithm execution. In particular, this approach may be used when several

operators are available for the same operation (crossover or mutation, for example). In [12], the author proposes to compute the rate of appliance of a mutation operator by calculating the progress of the last applications of this operator. Hence, it becomes possible to determine the probabilities of appliance of a given operator in an adaptively way where the more efficient an operator is, the more it will be used. Another approach could be to analyse in details the new individuals generated by operators (in term of quality, diversity) using clustering algorithms for example. This would give valuable information that can help to set the new application probabilities.

These two approaches give examples on the way the datamining techniques may help to set parameters.

4 Using Datamining for Initialization

Generally, metaheuristics generate their initial solution(s) randomly. In continuous optimization, this generation may also be done using a grid initialization. It could be also interesting to cleverly generate the initial population in order, for example, to reduce the search space by leading the metaheuristic to promising area.

For example, in [28], Ramsey et al. propose to initialize a genetic algorithm with case-based reasoning in a tracker/target simulation with a periodically changing environment. Case-based initialization allows the system to automatically bias the search of the genetic algorithm toward relevant areas of the search space.

5 Datamining and Population Management

Datamining techniques are often used to manage the population. Several works deal with introducing new individuals in the population. Some common methods try to inject new individuals into the population. This could be realized to diversify the population like in the random immigrant strategy. In order to lead the search to promising search spaces it could be also interesting to regularly introduce individuals that are built based on information of the past encountered solutions.

In [20,21], Louis presents CIGAR (Case Injected Genetic AlgoRithm). The aim of CIGAR is to provide periodically to the genetic algorithm solutions that suit to similar problems. CIGAR has been successfully applied to several problems such as jobshop, circuit modelling, etc.

In [4,39], the authors propose to hybridize a genetic algorithm and the Apriori algorithm (Apriori is a classical algorithm to generate association rules [1]) to discover interesting subroutines for the oil collecting vehicle routing problem. They insert the found subroutines into the new individuals.

In the work of Ribeiro et al. [37,38,40], the authors present a GRASP hybridized with several frequent item set mining algorithms: the Direct Count and

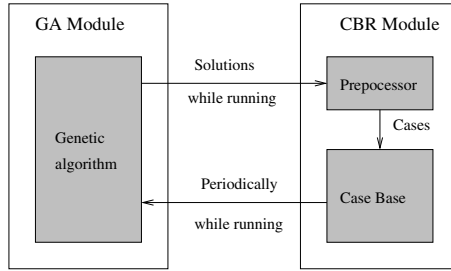


Fig. 3. Case Injected Genetic AlgoRithm (CIGAR)

the Intersect algorithms in [38] and the FPM Max^* in [40], which are Apriori-like approaches. These algorithms are used to extract patterns that are promising only on elite solutions. The hybridization is realized after a fixed number of seconds or iterations and new starting solutions for the GRASP are computed thanks to the found patterns. The authors apply their approach to the Set Packing Problem and the Maximum Diversity Problem. The method allows for the speed up of the convergence of the algorithm and for the improvement of the robustness of the GRASP.

Another example of the use of datamining techniques in the population management is the use of clustering algorithms in Multi-objective population metaheuristics where the result to produce is a set of solutions of best compromise (Pareto solutions). An archive is often used to store these solutions and the clustering is used to avoid a bias towards a certain region of the search space. Such a bias would lead to an unbalanced distribution of the Pareto solutions. Authors often use the average linkage method as this clustering algorithm performs well for Pareto optimization [48].

6 Using Datamining Within Operators

Incorporating knowledge in operators could be useful if, for example, it allows to cleverly exploit the search space. In the following section, some examples using machine learning approaches in crossover to explore the search space are presented.

Handa et al. propose a co-evolutionary genetic algorithm, which uses an hybridization between a GA and C4.5 (a classification algorithm) in order to discover the schemata to use in the crossover [10,9]. In their early work [8], Handa et al. have proposed a co-evolutionary algorithm in order to discover the good schemata to use that have not been discovered by the GA. The method works well but was just presented for bit representation.

LEM [22,23] integrates a symbolic learning component to evolutionary computation; it seeks out rules explaining the differences between the better and worse performers in the population, and generates new individuals based on the templates specified in these rules. An example of behaviour of LEM in the search

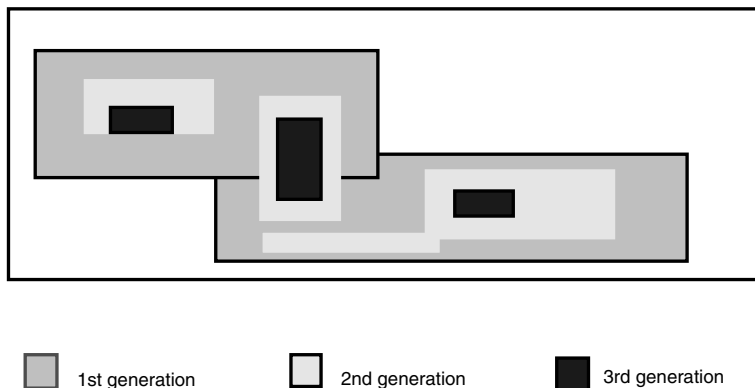


Fig. 4. LEM: Example of search region reductions defined by description of the 1st, 2nd and 3rd generations

space is shown on Figure 4 where the search regions associated to each generation are illustrated. This figure shows how the search space is reduced. The LEM methodology has proved able to improve the efficiency of the evolutionary process. LEM uses AQ learning algorithm (a general decision rules learning algorithm) in order to produce rules. Let us remark that, existing implementations, such as AQ11 [24], AQ15 [25] handle noise with pre and post-processing techniques. The basic AQ algorithm however, heavily depends on specific training examples during the search (the algorithm actually employs a beam search). This approach has been used with the C4.5 algorithm for mono-objective jobshop problems in the work of Huyet [14,13].

In the work of Jourdan et al. [17,45], the authors propose to extend LEM to LEMMO for the multi-objective case in order to seek for rules that explain why some individuals dominate others in a multi-objective point of view and why some individuals are dominated by other. They generate new individuals thanks to the rules by creating solutions that match to positive rules and do not match to negative rules. This approach has shown good results on a water system application both in speeding up the multi-objective algorithm and in improving the quality of the solutions.

We can remark that usually the authors use classification methods (C4.5, AQ, etc) to identify the genes that induce the good quality of the individuals. Some authors propose to also determine the genes that characterize bad quality solutions and to use them to repair the constructed solutions [17,45].

7 Datamining in Local Search

Metaheuristics are often hybridized with local search methods to improve the intensification part. Some datamining algorithms are themselves local searches and could be used as part of the metaheuristics.

In [6], the authors propose to use inverse Artificial Neural Networks (ANN) as local search to exploit specific region for candidate solutions. They note that ANNs can be adequate as they construct a smooth mapping. The ANN is trained in a reverse way as the input layer presents the criteria and the output the parameters to be optimised.

Moreover, datamining problems can be often modeled as optimization problems and in this case, the hybridization of the metaheuristic with a machine learning algorithm could be realized such that the machine learning algorithm treats a subproblem. For example, when clustering or grouping problems are solved using a metaheuristic, the metaheuristic searches for the optimal subset of genes that act as initial cluster centers. At the lower level, a local learning method performs local clustering from these initial centers. The objective is to combine the strength of EAs and clustering methods to produce a global efficient clustering algorithm. Kmeans is often used as a local search [5,11,46] for initialization of the solutions or to realize a local search during the search. Another approach, presented in [7], uses fuzzy c-means and hard c-means as an objective function. This article shows the importance of the initialization of the solution(s). In [3], the authors also use Expectation Maximization (EM) as a local search to analyze gene trajectory.

8 Datamining Based Metaheuristics

Some metaheuristics are designed to directly care of dynamic knowledge. We decide to create a specific part for them as they are now considered as new metaheuristic and not as improvement of previous ones. A lot of these algorithms are classified as Non-Darwinian evolutionary computation as they replace Darwinian operators by other operators. As identified in the taxonomy (Figure 1), the cooperation can appear in different localizations but we observe that in the proposed metaheuristics, the integration is often localised in the operator part. For example, the *Population-based Incremental Learning* (PBIL) creates a real-valued probability vector characterizing high fitness solutions [2] (Figure 5). This vector is then used to build new solutions. Generally, PBIL does not use mutation and crossover. PBIL can be considered as both encoding and initialisation localization hybridizations.

Specifically, Muhlenbein and Paass have estimated the probability density functions of binary variables in their chromosomes by the product of the individual probability density functions in the UMDA (Univariate Marginal Distribution Algorithm) [26]. Hence, UMDA is a special class of the PBIL algorithm.

Pelikan and Goldberg developed an algorithm "BOA" (Bayesian Optimization Algorithm) that extends the above ideas by using Bayesian Networks to model the chromosomes of superior fitness [27] (Figure 6). BOA can be classified as localization operator algorithm with dynamic knowledge.

A similar approach has also been proposed by Larranaga and Lozano, who have given the term "EDA" (Estimation of Distribution Algorithms) to the statistical estimation approach to EC [19].

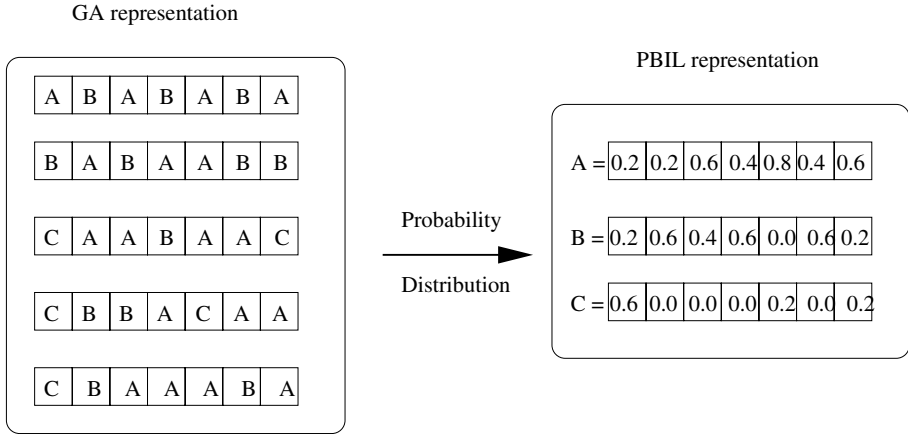


Fig. 5. The PBIL probability vector

```

begin
  t=0;
  Initialise randomly Population POP(0);
  Evaluate(POP(0));
  repeat
    Select a set of promising strings S(t) from POP(t);
    Construct the network B using a given metric and constraints;
    Generate a set of new strings O(t) according to the joint distribution
      encoded by B;
    Create a new population POP(t+1) by replacing some strings from P(t)
      with O(t);
    Evaluate(POP(t));
    t=t+1;
  until (termination condition)
end

```

Fig. 6. Overview of the Bayesian Optimization Algorithm

Similarly, cultural algorithms use high performing individuals to develop beliefs constraining the way in which individuals are modified by genetic operators [35,36] (Figure 7). In cultural algorithm, beliefs are formed based on each entity's individual experiences. The reasoning behind this, as outlined by [35], is that cultural evolution allows populations to learn and adapt at a rate faster than pure biological evolution alone. Importantly, the learning which takes place individually by each entity is passed on to the remainder of the group, allowing learning to take place at a much faster rate. Cultural algorithm can be classified as operator localization algorithm in the taxonomy.

Ravise and Sebag [43,34,41,33,42] worked on civilized genetic algorithms that differ from Darwinian's ones as they keep information of the population in order to avoid doing the same errors. The knowledge is dynamically updated during

```

begin
  t=0;
  Initialise Population POP(0);
  Initialise Belief Network BLF(0);
  Initialise Communication Channel CHL(0);
  Evaluate(POP(0));
  t=1;
  repeat
    Communicate(POP(0), BLF(t));
    Adjust(BLF(t));
    Communicate(BLF(t), POP(t));
    Modulate Fitness (BLF(t), POP(t));
    t=t+1;
    Select POP(t) from POP(t-1);
    Evolve(POP(t));
    Evaluate(POP(t));
  until (termination condition)
end

```

Fig. 7. Overview of the cultural evolution algorithm (Reynolds 1994)

generations. The datamining hybridization accelerates and improves the convergence of the algorithm. But it has been tested only on bit representation. The authors have observed that the GA must be run first with Darwinian operator in order to have diversity in its population. After a fixed number of generations, the civilized operator is used. They keep history of the past results in order to not reproduce the same error (and produce bad individuals). Civilized genetic algorithms can be classified as operator based dynamic knowledge.

9 Discussion and Conclusion

We have seen that there are multiple reasons to integrate datamining methods within metaheuristics. It could be to approximate the fitness function, to improve the convergence of the metaheuristics or to create an operator that is adapted to the problem.

In a research point of view, the actual major interest is to use datamining to extract useful information from the history of the metaheuristic in order to move the search in interesting space areas. Moreover, the hybridization between metaheuristics and datamining techniques have not been studied a lot in multi-objective optimization.

The major drawback of hybridization is the setting of parameters. When applying the datamining method, how many solutions have to be stored in dynamic knowledge, etc ? Many articles realize experimentally the parameter settings and many authors remark that clearly the performances are correlated with the parameters. A very promising search investigation is to automatically determine during the search all these parameters for designing adaptive efficient metaheuristics.

References

1. Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *Proceeding 20th International Conference Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 12–15 1994.
2. S. Baluja. Population based incremental learning. Technical Report CMU-CS-94-163, Carnegie Mellon University, 1994. <http://www.cs.cmu.edu/afs/cs/user/baluja/www/techreps.html>.
3. Z.S.H. Chan and N. Kasabov. Gene trajectory clustering with a hybrid genetic algorithm and expectation maximization method. In *IEEE International Joint Conference on Neural Networks*, pages 1669–1674, 2004.
4. F.L. Dalboni, L.S. Ochi, and L.M.A. Drummond. On improving evolutionary algorithms by using data mining for the oil collector vehicle routing problem. International Network Optimization Conference, 2003.
5. Emanuel Falkenauer. A new representation and operators for genetic algorithms applied to grouping problems. *Evolutionary Computation*, 2(2):123–144, 1994.
6. A. Gaspar-Cunha and A.S. Vieira. A hybrid multi-objective evolutionary algorithm using an inverse neural network. In *Hybrid Metaheuristic*, pages 25–30, 2004.
7. L. O. Hall, I. B. Özyurt, and J. C. Bezdek. Clustering with a genetically optimized approach. *IEEE Trans. on Evolutionary Computation*, 3(2):103–112, 1999.
8. H. Handa, N. Baba, O. Katai, and T. Sawaragi. Coevolutionary genetic algorithm with effective exploration and exploitation of useful schemata. In *Proceedings of the International Conference on Neural Information Systems*, volume 1, pages 424–427, 1997.
9. H. Handa, T. Horiuchi, O. Katai, and M. Baba. A novel hybrid framework of coevolutionary GA and machine learning. *International Journal of Computational Intelligence and Applications*, 2002.
10. H. Handa, T. Horiuchi, O. Katai, T. Kaneko, T. Konishi, and M. Baba. Fusion of coevolutionary ga and machine learning techniques through effective schema extraction. 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)*, page 764, San Francisco, California, USA, 7–11 July 2001. Morgan Kaufmann.
11. J. Handl and J. Knowles. Improvements to the scalability of multiobjective clustering. In IEEE, editor, *IEEE Congress on Evolutionary Computation*, pages 438–445, 2005.
12. T.P. Hong, H. Wang, and W. Chen. Simultaneously applying multiple mutation operators in genetic algorithms. *Journal of heuristics*, 6:439–455, 2000.
13. A.-L. Huyet. *Extraction de connaissances pertinentes sur le comportement des systèmes de production : une approche conjointe par optimisation évolutionniste via simulation et apprentissage*. PhD thesis, Université Blaise Pascal Clermont II, October 2004.
14. A.-L. Huyet and J.-L. Paris. Configuration and analysis of a multiproduct kanban system using evolutionary optimisation coupled to machine learning. In *Proceedings of CESA 2003, the IMACS Multiconference Computational Engineering in Systems Applications*, July 2003. ISBN 2-9512309-5-8, CDROM.
15. Y. Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing Journal*, 9(1):3–12, 2005.

16. Y. Jin and B. Sendhoff. Reducing fitness evaluations using clustering techniques and neural networks ensembles. In *Genetic and Evolutionary Computation Conference*, volume 3102 of *LNCS*, pages 688–699. Springer, 2004.
17. L. Jourdan, D. Corne, D.A. Savic, and G.A. Walters. Preliminary investigation of the learnable evolution model for faster/better multiobjective water systems design. In *LNCS 3410*, editor, *Third International Conference on Evolutionary Multi-Criterion Optimization (EMO'05)*, pages 841–855, 2005.
18. H.-S. Kim and S.-B. Cho. An efficient genetic algorithms with less fitness evaluation by clustering. In *Proceedings of IEEE Congress on Evolutionary Computation*, pages 887–894. IEEE, 2001.
19. P. Larranaga and J.A. Lozano. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, 2002.
20. S. J. Louis. Genetic learning from experience. In IEEE, editor, *Congress on Evolutionary Computation (CEC'03)*, pages 2118 – 2125, Australia, Dec 2003. IEEE.
21. S. J. Louis. Learning for evolutionary design. In *Proceedings of the 2003 Nasa/DoD Conference on Evolutionary Hardware*, pages 17–23, July 2003.
22. R.S. Michalski. Learnable evolution model: Evolutionary processes guided by machine learning. *Machine Learning*, 38(1–2):9–40, 2000.
23. R.S. Michalski, G. Cervon, and K.A. Kaufman. Speeding up evolution through learning; Lem. In *Intelligent Information Systems 2000*, pages 243–256, 2000.
24. R.S. Michalski and J.B. Larson. Selection of most representative training examples and incremental generation of v11 hypothesis: The underlying methodology and the descriptions of programs esel and aq11. Technical Report Report No. 867, Urbana, Illinois: Department of Computer Science, University of Illinois, 1978.
25. R.S. Michalski, I. Mozetic, J. Hong, and N. N. Lavrac. The multipurpose incremental learning system aq15 and its testing application to three medical domains. In *Proc. of the Fifth National Conference on Artificial Intelligence*, pages 1041–1045. PA: Morgan Kaufmann, 1986.
26. H. Muhlenbein and G. Paass. From recombination of genes to the estimation of distributions: I. binary parameters. *Lecture Notes in Computer Science*, 1141:178–187, 1996.
27. M. Pelikan, D. E. Goldberg, and E. Cantú-Paz. BOA: The Bayesian optimization algorithm. 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 GECCO-99*, volume I, pages 525–532, Orlando, FL, 13–17 1999. Morgan Kaufmann Publishers, San Fransisco, CA.
28. C. Ramsey and J. Grefenstette. Case-based initialization of genetic algorithms. In *Fifth International Conference on Genetic Algorithms*, pages 84–91, 1993.
29. K. Rasheed. An incremental-approximate-clustering approach for developing dynamic reduced models for design optimization. In *Proceedings of the 2000 Congress on Evolutionary Computation (CEC'00)*, pages 986–993, California, USA, 6–9 2000. IEEE Press.
30. K. Rasheed and H. Hirsh. Using case based learning to improve genetic algorithm based design optimization. In Thomas Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA97)*, San Francisco, CA, 1997. Morgan Kaufmann.
31. K. Rasheed and H. Hirsh. Informed operators: Speeding up genetic-algorithm-based design optimization using reduced models. In L. Darrell Whitley, David E. Goldberg, Erick Cantú-Paz, Lee Spector, Ian C. Parmee, and Hans-Georg Beyer, editors, *GECCO*, pages 628–635. Morgan Kaufmann, 2000.

32. K. Rasheed, S. Vattam, and X. Ni. Comparison of methods for developing dynamic reduced models for design optimization. In *Proceedings of the Congress on Evolutionary Computation (CEC'2002)*, pages 390–395, 2002.
33. C. Ravise and M. Sebag. An advanced evolution should not repeat its past errors. In *International Conference on Machine Learning*, pages 400–408, 1996.
34. C. Ravise, M. Sebag, and M. Schoenauer. A genetic algorithm led by induction. url: citeseer.ist.psu.edu/126354.html.
35. R. G. Reynolds, Z. Michalewicz, and Michael J. Cavaretta. Using cultural algorithms for constraint handling in genocop. In *Evolutionary Programming*, pages 289–305, 1995.
36. R. G. Reynolds and B. Peng. Cultural algorithms: computational modeling of how cultures learn to solve problems: an engineering example. *Cybernetics and Systems*, 36(8):753–771, 2005.
37. M. Ribeiro, A. Plastino, and S. Martins. Hybridization of grasp metaheuristic with data mining techniques. *Special Issue on Hybrid Metaheuristic of the Journal of Mathematical Modelling and Algorithms*, 5(1):23–41, April 2006.
38. M. Ribeiro, V. Trindade, A. Lastino, and S. Martins. Hybridization of GRASP metaheuristic with data mining techniques. In *Workshop on Hybrid Metaheuristics 16th European Conference on Artificial Intelligence (ECAI)*, pages 69–78, 2004.
39. H.G. Santos, L.S. Ochi, E.H. Marinho, and L.M.A. Drummond. Combining an evolutionary algorithm with data mining to solve a vehicle routing problem. *NEURO-COMPUTING*, 2006. (to appear).
40. L. Santos, M. Ribeiro, A. Plastino, and S. Martins. A hybrid GRASP with data mining for the maximum diversity problem. In LNCS 3636, editor, *Hybrid Metaheuristic*, pages 116–128, 2005.
41. M. Sebag, C. Ravise, and M. Schoenauer. Controlling evolution by means of machine learning. In *Evolutionary Programming*, pages 57–66, 1996.
42. M. Sebag and M. Schoenauer. Controlling crossover through inductive learning. In Yuval Davidor, Hans-Paul Schwefel, and Reinhard Männer, editors, *Parallel Problem Solving from Nature – PPSN III*, pages 209–218, Berlin, 1994. Springer.
43. M. Sebag, M. Schoenauer, and C. Ravise. Toward civilized evolution: Developing inhibitions. In Thomas Bäck, editor, *Proceeding of the Seventh Int. Conf. on Genetic Algorithms*, pages 291–298, San Francisco, CA, 1997. Morgan Kaufmann.
44. E-G. Talbi. A taxonomy of hybrid metaheuristics. *Journal of heuristics*, 8(2):541–564, Sept. 2002.
45. L. Vermeulen-Jourdan, D. Corne, D.A. Savic, and G.A. Walters. Hybridising rule induction and multi-objective evolutionary search for optimising water distribution systems. In *Proceeding of Fourth International Conference on Hybrid Intelligent Systems (HIS'04)*, pages 434–439, 2004.
46. L. Vermeulen-Jourdan, C. Dhaenens, and E-G. Talbi. Clustering nominal and numerical data: A new distance concept for a hybrid genetic algorithm. In Jens Gottlieb and Günther R. Raidl, editors, *Evolutionary Computation in Combinatorial Optimization – EvoCOP 2004*, volume 3004 of LNCS, pages 220–229, Coimbra, Portugal, 5-7 April 2004. Springer Verlag.
47. S-H. Yoo and S-B. Cho. Partially evaluated genetic algorithm based on fuzzy c-means algorithm. In LNCS 3242, editor, *Parallel Problem Solving From Nature (PPSN)*, pages 440–449, 2004.
48. 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.

An Iterated Local Search Heuristic for a Capacitated Hub Location Problem

Inmaculada Rodríguez-Martín and Juan-José Salazar-González

DEIOC, Universidad de La Laguna,
38271 La Laguna, Tenerife, Spain
{irguez, jjsalaza}@ull.es

Abstract. This paper addresses a capacitated hub problem consisting of choosing the routes and the hubs to use in order to send a set of commodities from sources to destinations in a given capacitated network with minimum cost. The capacities and costs of the arcs and hubs are given, and the graph connecting the hubs is not assumed to be complete. For solving this problem we propose a heuristic approach that makes use of a linear programming relaxation in an Iterated Local Search scheme. The heuristic turns out to be very effective and the results of the computational experiments show that near-optimal solutions can be derived rapidly for instances of large size.

1 Introduction

This paper presents a heuristic method for solving a *Capacitated Hub Problem* (CHP) that arises in the design of telecommunications networks. The CHP is defined as follows. Let us consider a set I of computers (*source terminals*) that must send information to a set J of computers (*destination terminals*) through a network. The network consists of cables, possibly going through some computers (*hubs*) of another set H . The whole set of computers is called *node set* and it is represented by $V := I \cup J \cup H$, and the set of cables is called *arc set* and it is represented by A . Hence, we have a directed graph $G = (V, A)$. A given computer can be at the same time source terminal, destination terminal and hub. When this happens, for notational convenience, the computer is represented by several nodes in V , thus we will assume that subsets I , J and H are disjoint. When a cable allows communication in both directions it is considered as two opposite arcs in A . We will assume there are not arcs in A from a destination terminal to a source one, i.e., there are no arcs from nodes in J to nodes in I .

Associated to each arc $a \in A$ there is a *capacity* q_a , representing the maximum amount of information units that can go through it, and a value c_a representing the cost of sending a unit of information through a and called *routing cost*. Equally, associated to each hub $h \in H$ there is a *capacity* q_h , and a value c_h representing the fix cost of using hub h and called *maintenance cost*. Finally, for each pair $(i, j) \in I \times J$ of terminals we are given an amount d_{ij} of information to be sent from i to j . Since most of these values can be zero, the d_{ij} information units going from i to j will be called *commodity* if $d_{ij} > 0$, and for simplicity

they will be referred with one index $k \in K := \{(i, j) \in I \times J : d_{ij} > 0\}$. Also for brevity of notation, we will write d_k instead of d_{ij} if $k = (i, j)$.

The Capacitated Hub Problem consists of deciding the way the required flow of information between terminals must traverse the capacitated network in order to minimize the sum of the routing and maintenance costs.

A particular case of this problem, referred to as the *Capacitated Multiple Allocation Hub Location Problem*, has been recently considered in Ebery et al. [6], motivated by a postal delivery application. In this particular combinatorial problem, where the name “terminal” is replaced by “client”, the capacity requirements only concern the hubs and apply only to mail letters coming directly from a client and going directly to a client. Moreover, the maintenance cost of a hub is paid when it receives or delivers letters to a client, and it is not paid if the letter is moved from one hub to another hub. The subgraph connecting the hubs is assumed to be complete (see, e.g., Carello et al. [5] for a heuristic approach to solve a hub location problem in telecommunications where the hubs are supposed to be fully connected). This assumption allows a reduction in the number of candidate paths for the mail, as a letter will never go through more than two hubs. Ebery et al. [6] present mathematical models exploiting these advantages by using variables with three and four indices. Other papers on capacitated versions of hub location problems are Ernst and Krishnamoorthy [8], Campbell [4], Aykin [1], and Boland et al. [3]. Some of these papers study the single-allocation version of the problem and others are devoted to the multi-allocation version. Another distinction can be made on whether the demand flow of a commodity can be split or not on different paths. Uncapacitated versions of the CHP have been extensively studied in the literature (see, e.g., Campbell [4], Ernst and Krishnamoorthy [7], Mayer and Wagner [13], Skorin-Kapov, Skorin-Kapov and O’Kelly [16]). In this context, the CHP is a capacitated multiple-allocation hub location problem with splittable demands, without the assumption that the network connecting the hubs is a complete graph.

The CHP is also related to network design (see, e.g., Barahona [2], O’Kelly, Bryan, Skorin-Kapov and Skorin-Kapov [14]). In particular, a problem closely related to CHP has been addressed in Holmberg and Yuan [10]. Indeed, it is possible to convert a CHP instance into an instance of the problem in [10] by duplicating each hub and adding a dummy arc connecting the new pair of nodes with appropriately defined cost and capacity. Arcs in the CHP network have routing costs and no fix costs, while dummy arcs only have fix costs. This special structure of the resulting network design instances justifies the research on an ad-hoc algorithm for the CHP.

Rodríguez-Martín and Salazar-González [15] propose a mixed integer linear programming formulation for CHP and describe three exact methods to solve it based on decomposition techniques. These exact methods are able to deal only with small CHP instances. In this paper we present a heuristic method that provides good feasible solutions for large CHP instances in a reasonable time. The heuristic is based on the linear programming relaxation of a mixed-integer mathematical model.

The remainder of the paper is organized as follows. Section 2 presents a mathematical formulation based on 2-index variables. In Section 3 we outline the heuristic algorithm. Section 4 shows the computational results, and finally Section 5 is devoted to conclusions.

2 Problem Formulation

In order to simplify the notation it is convenient to extend the arc set of G with the dummy arcs from destinations to origins, i.e.,

$$A := A \cup \overleftarrow{K},$$

where \overleftarrow{K} is the set of arcs from $j \in J$ to $i \in I$ if $d_{ij} > 0$. For each subset $S \subset V$, we will denote

$$\delta^+(S) := \{(u, v) \in A : u \in S, v \in V \setminus S\}$$

and

$$\delta^-(S) := \{(u, v) \in A : u \in V \setminus S, v \in S\}.$$

Let us consider a decision variable y_h associated to each $h \in H$ that takes value 1 when the hub is used, and a continuous variable x_a associated to each $a \in A$ representing the amount of communication traversing arc a .

To present a mathematical model we also make use of an additional set of continuous variables, $[f_a^k : a \in A]$, for each commodity $k \in K$. If $k = (i, j)$ is the commodity going from i to j , then variable f_a^k represents the amount of communication from source i to destination j traversing arc a . Then the mathematical model for CHP is:

$$\min \sum_{a \in A} c_a x_a + \sum_{h \in H} c_h y_h \quad (1)$$

subject to:

$$x_a \leq q_a \quad \text{for all } a \in A \quad (2)$$

$$\sum_{a \in \delta^+(\{h\})} x_a \leq q_h y_h \quad \text{for all } h \in H \quad (3)$$

$$y_h \in \{0, 1\} \quad \text{for all } h \in H, \quad (4)$$

and there exist f_a^k such that

$$x_a = \sum_{k \in K} f_a^k \quad \text{for all } a \in A \quad (5)$$

and for each commodity $k = (i, j)$:

$$\sum_{a \in \delta^+(\{v\})} f_a^k = \sum_{a \in \delta^-(\{v\})} f_a^k \quad \text{for all } v \in V \quad (6)$$

$$f_a^k \geq 0 \quad \text{for all } a \in A \quad (7)$$

$$f_{(j,i)}^k = d_k \quad (8)$$

$$f_{(u,v)}^k = 0 \quad \text{for all } (u, v) \in \overleftarrow{K} \setminus \{(j, i)\}. \quad (9)$$

Constraints (6)–(9) require the existence of a flow circulation in G moving exactly d_k units of commodity k . Equalities (5) gather the individual flows into a common one that must satisfy the arc-capacity requirements in (2) and the node-capacity requirements in (3). Finally, (4) impose that a 0-1 decision on the hubs must be taken. Then, vector x represents the total communication units circulating through the network, and is a vector of continuous variables, while vector y is a 0-1 vector indicating the hubs to be installed. The main difference with a multi-commodity flow formulation problem is that the x_a variables are not required to be integer variables. The problem remains difficult to solve due to the integrability requirement on the y_h variables. In fact, the CHP is \mathcal{NP} -hard since it generalizes the *Uncapacitated Multiple Allocation Hub Location Problem*, which is known to be \mathcal{NP} -hard (see Hamacher et al. [9]).

Magnanti and Wong [12] observed that it is computationally useful to add the upper bound inequalities $f_a^k \leq q_h y_h$ for all $k \in K$, for all $h \in H$ and for all $a \in \delta^+(\{h\}) \cup \delta^-(\{h\})$, even if they are redundant when inequalities (3) are also present in the model. This was also confirmed by our experiments. No theoretical explanation is known (to our knowledge) for this behavior. A potential argument could be based on the fact that the linear program is highly degenerated due to the underlying multi-commodity structure, and the redundant inequalities could help the LP-solver. In addition, we also got better computational results by extending the model with the equations

$$\sum_{a \in \delta^+(\{v\})} x_a = \sum_{a \in \delta^-(\{v\})} x_a \quad \text{for all } v \in V. \quad (10)$$

The following section describes a heuristic method that provides good feasible solutions for the CHP.

3 A Heuristic Approach

At first glance, a simple way to tackle the CHP is to solve the mixed integer programming model (1)–(10) with a general purpose MIP solver. However, the model may have a large number of continuous variables and constraints, and therefore commercial solvers are unable to solve even medium-size instances. This drawback motivates our research on near-optimal approaches.

The heuristic we propose is based on the fact that, for a given subset of hubs $H' \subset H$, the LP-relaxation of model (1)–(10) resulting from setting $y_h = 1$ for all $h \in H'$ and $y_h = 0$ for all $h \in H \setminus H'$, is relatively easy to solve using a LP-solver. The LP-relaxation associated with a subset $H' \subset H$ gives the optimal way of routing the demands if only those hubs are open in the network, or shows that no feasible routing exists with that y_h setting. Therefore, if H' is the optimal set of hubs for a CHP instance, by solving the associated LP-relaxation we will get the optimal CHP values of the flow variables f_a^k .

Overall, the heuristic consists of a construction phase and a combined local search and shaking phase which is repeated until a stopping condition is met.

The best solution found during this iterative process is the final heuristic solution. In the construction phase a set of hubs is selected. A shaking procedure is applied after each local search to randomly perturb the current solution. If feasibility is lost, another procedure tries to recover it before starting the following local search. See Algorithm 1. for the whole scheme. It follows the scheme of an Iterative Local Search (ILS) (see e.g. [11]) where the aim is to create a random walk in the space of local optima defined by the output of a local search procedure. In our case, this procedure makes use of information given by a linear programming relaxation.

Algorithm 1. LP-based algorithm for the CHP

```

GenerateInitialHubSet( $H'$ ) { construction phase }
BestSolutionValue  $\leftarrow$  SolveLP( $H'$ )
while stopping criterium is not satisfied do
  { to obtain feasibility }
  while  $H'$  is not feasible do
    Select a hub  $h \in H \setminus H'$ 
     $H' \leftarrow H' \cup \{h\}$ 
  end while
  CurrentSolutionValue  $\leftarrow$  SolveLP( $H'$ )
  { local search }
  repeat
    Select a hub  $h \in H'$ 
    NewSolutionValue  $\leftarrow$  SolveLP( $H' \setminus \{h\}$ )
    if  $H' \setminus \{h\}$  is feasible and NewSolutionValue  $<$  CurrentSolutionValue then
       $H' \leftarrow H' \setminus \{h\}$ 
      CurrentSolutionValue  $\leftarrow$  NewSolutionValue
    end if
  until no better feasible solution is found
  if CurrentSolutionValue  $<$  BestSolutionValue then
    BestSolutionValue  $\leftarrow$  CurrentSolutionValue
  end if
  Shake( $H'$ )
end while

```

Next we describe with more detail each of the heuristic components.

3.1 Construction Phase: Generation of an Initial Set of Hub

To generate an initial set of hubs H' , we proceed as follows. Initially, H' is the empty set. For each commodity $k = (i, j)$ we consider hubs h such that arc $a = (i, h)$ exists. These candidate hubs are sorted in increasing order of $c_a q_a + c_h$. The amount $c_a q_a + c_h$ intends to be a measure of the cost of shipping demand to hub h through arc a . We add to H' as many hubs of this sorted list as necessary to make it possible to supply the demand d_k from source i , taking into account the capacities of the arcs and hubs. In a similar way, hubs h such

that arc $a = (h, j)$ exists, are sorted according to the same criterium, and the ones necessary to make it possible that the demand d_k reaches destination j are added to H' . During this process we keep a report of the spare capacity of each hub and each arc. When a hub or an arc becomes saturated, it is not further considered.

3.2 Getting a Feasible Solution

A given subset of hubs $H' \subseteq H$ is said to be feasible if it is possible to send all the demands d_k from their sources to their destinations when restricting the hub set to H' . In other words, H' is feasible when the LP-model resulting from setting $y_h = 1$ for all $h \in H'$ and $y_h = 0$ for all $h \in H \setminus H'$ in formulation (1)–(10) has a feasible solution. If that is not the case, the feasibility is sought by iteratively adding new hubs to H' . In each iteration the hub in $H \setminus H'$ with minimum cost is chosen. The procedure stops when feasibility is reached.

Intuitively, the set H' represents the set of a priori *open* hubs, i.e. those through which the demand flow can go, while all hubs in $H \setminus H'$ are *closed*. If in fact no flow traverses a hub in H' , this hub is removed from H' . Hubs in $H \setminus H'$ are said to be *closed*.

3.3 Local Search

The local search procedure starts from a feasible solution H' and tries to improve it by closing hubs, that is, by removing hubs from H' . In each iteration the open hub that maximizes the ratio c_h/f_h , being f_h the total amount of demand traversing h , is the candidate to be closed. Thus, the aim is to close a hub with a high cost and a low utility. The movement is done only if it produces a better solution. The local search procedure continues until no further improvement is obtained or the feasibility is lost.

Notice that the evaluation of a solution H' (referred as $\text{SolveLP}(H')$ in Algorithm 1.) implies solving the corresponding LP-relaxation of model (1)–(10), as explained before, using a commercial LP-solver. This is the most time consuming part of the heuristic. To prevent the same LP-relaxation to be solved several times, we keep a report of already tested solutions H' . That is, a closing movement is performed only if it does not lead to an already tested solution H' .

3.4 Shaking

The shake procedure allows to escape from a local minimum without completely destroying its good properties. In order to do so, half of the hubs in the current solution H' are closed (i.e., are removed from H'), and $2|H'|$ hubs in $H \setminus H'$ are opened (i.e., are added to H'). The hubs to be opened and to be closed are selected randomly among all possible candidates. The new initial solution obtained in this way shares at least half of its hubs with the former local minimum. However, the new solution includes a large enough number of new hubs as to make it likely that the following local search leads to a different local minimum.

3.5 Stopping Criterium

As Algorithm 1. shows, the feasibility obtaining, local search, and the shaking procedures, are embedded in a loop. The stopping criterium determines the number of times this loop is executed. We found a good compromise between solution quality and computing time by letting the algorithm run until ten iterations have been performed without obtaining any improvement in the best solution.

4 Computational Experiments

This section shows the results of experiments conducted for solving some instances with the heuristic method. Standard hub location instances available in the literature (e.g., CAB and AP in [6]) do not fit into the definition of CHP since, for example, they are based on complete graphs. Hence we have generated random instances. These experiments were carried out on a Pentium IV 1500 Mhz. using CPLEX 8.1 as LP and MIP solver.

The instance generator is next described. Given the sets I, J, H , the arc density of the graph induced by H was fixed to a parameter taking values of 30%, 50% and 85% of the total amount of possible arcs, which gave instances with low, medium and high density. The percentage of arcs from $(I \times H) \cup (H \times J)$ was fixed to 80% of the total amount of possible arcs. The amount of information d_{ij} from $i \in I$ to $j \in J$ was generated in $[1, 5]$ for all commodities. The capacities q_h and q_a were generated in $[1, \lambda|I||J|]$, being $\lambda = 2.5$. The costs c_h and c_a were generated in $[50, 150]$ and $[1, 50]$ respectively. All these settings were chosen in order to increase the probability of producing instances with feasible solutions. We considered $(|I|, |J|, |H|)$ in $\{(2, 2, 4), (3, 3, 5), (5, 5, 5), (5, 5, 10), (5, 5, 30), (5, 5, 40), (5, 5, 50), (10, 10, 50)\}$, and for each triplet we generated five random instances with the above features.

To evaluate the quality of the heuristic solutions, they are compared to the solutions obtained by CPLEX running with a time limit of three hours of CPU. CPLEX works on the mixed integer programming model (1)–(10) and gives the optimal solution, if it stops before the time limit, or a feasible solution otherwise. CPLEX also reports the lower bound for the optimal objective value obtained at the end of the root node in the branch-and-cut tree, or the best lower bound found if the time limit is reached before the root node is completed. This lower bound from CPLEX is useful as an alternative measure of the heuristic quality when the optimal solution is not available.

Tables 1 to 3 show average results of applying the heuristic algorithm on feasible instances. Column headings display:

- $(|I|, |J|, |H|)$: Number of source, destinations and hubs nodes.
- d : Density of the generated graphs: low (l), medium (m) or high (h).
- *open-hubs*: Number of hubs open in the optimal solution.
- *heur/opt*: Percentage deviation between the heuristic value and the optimal (or best) CHP solution value.

- *heur/lb*: Percentage deviation between the heuristic value and the CPLEX lower bound at the root node.
- *time-heur*: Heuristic CPU time (in seconds).
- *time-opt*: CPU time (in seconds) that CPLEX takes to find the optimal solution.
- *time-lb*: CPU time (in seconds) that CPLEX takes to find the lower bound at the root node.

Table 1 shows the results obtained for small instances, with a number of commodities going from 4 to 25 and a number of hubs ranging from 4 and 10. For these instances, the average deviation of the heuristic solution from the optimal solution generated by CPLEX never exceeds 1.25%, and it is 0.0% in 50 of the 60 instances solved. Therefore, the heuristic produces solutions of high quality. However, for these small instances the times of the heuristic and exact methods are comparable.

Table 2 shows the results for medium size instances, with 25 commodities and a number of hubs ranging from 30 to 50. We appreciate that the gap between the heuristic and the optimal solution values increases with the size of the problems,

Table 1. Heuristic results for small instances

(I , J , H)	d	$open-hubs$	$heur/opt$	$heur/lb$	$time-heur$	$time-opt$	$time-lb$
(2, 2, 4)	l	2.4	0.00	0.00	0.03	0.03	0.03
	m	2.4	0.00	1.16	0.02	0.02	0.02
	h	2.4	1.24	1.30	0.02	0.02	0.02
(3, 3, 5)	l	3.2	0.57	0.57	0.03	0.06	0.06
	m	3.2	0.00	0.12	0.03	0.03	0.03
	h	3.6	1.07	1.28	0.03	0.04	0.04
(5, 5, 5)	l	4.4	0.00	0.00	0.06	0.04	0.04
	m	4.6	0.00	0.00	0.07	0.05	0.05
	h	4.4	0.19	0.27	0.12	0.08	0.08
(5, 5, 10)	l	5.0	0.55	0.55	0.21	0.17	0.17
	m	5.6	1.24	1.24	0.24	0.45	0.45
	h	6.2	0.65	0.92	0.44	1.00	0.98

Table 2. Heuristic results for medium-size instances

(I , J , H)	d	$open-hubs$	$heur/opt$	$heur/lb$	$time-heur$	$time-opt$	$time-lb$
(5, 5, 30)	l	6.4	1.52	2.34	2.47	48.59	43.87
	m	6.4	3.34	4.32	4.31	109.49	97.30
	h	6.2	3.70	5.46	5.46	277.37	226.81
(5, 5, 40)	l	6.4	1.47	2.28	3.81	171.34	156.33
	m	6.6	5.14	5.91	5.25	356.63	329.16
	h	6.0	2.78	4.40	10.33	1198.46	848.65
(5, 5, 50)	l	6.0	6.97	7.49	6.45	438.35	416.08
	m	6.8	9.23	9.83	8.89	845.33	765.87
	h	5.8	6.43	8.05	12.86	2696.73	2103.81

Table 3. Heuristic results for large instances

(I , J , H)	d	$heur/opt$	$heur/lb$	$time-heur$	$time-opt$	$time-lb$
	l	5.21	5.91	28.87	5225.56	3963.08
(10, 10, 50)	m	3.43	5.55	51.73	t.l.	t.l.
	h	-20.32	24.67	89.87	t.l.	t.l.

Table 4. Comparison of the heuristic performance with different stopping criterium settings

			50 iter		100 iter		200 iter	
(I , J , H)	d	$time-opt$	$heur/opt$	$time-heur$	$heur/opt$	$time-heur$	$heur/opt$	$time-heur$
(5, 5, 30)	l	48.59	0.58	11.22	0.58	18.30	0.08	50.56
	m	109.49	0.23	26.90	0.02	41.99	0.02	65.26
	h	277.37	1.10	26.88	0.59	59.33	0.10	141.75
(5, 5, 40)	l	171.34	0.21	26.78	0.11	46.61	0.11	76.37
	m	356.63	1.98	36.18	1.59	63.89	0.41	146.68
	h	1198.46	0.28	60.27	0.05	121.53	0.05	191.36
(5, 5, 50)	l	438.35	3.43	31.10	3.12	61.08	1.35	126.10
	m	845.33	6.81	67.51	4.94	131.67	2.67	303.02
	h	2696.73	1.74	93.69	0.77	178.63	0.77	289.16

although it was always smaller than 10%. For CPLEX these instances are more difficult to solve, as the column *time-opt* shows. The exact method takes almost 45 minutes on average to solve the largest instances, while the heuristic takes approximately 13 seconds on average.

CPLEX is able to solve all instances in Tables 1 and 2 within the time limit of three hours. However it can not cope with bigger instances in that time, as Table 3 shows. When the time limit is exceeded we report 't.l.' in the corresponding columns, and compute the deviation gaps for the heuristic with respect to the best feasible solution found so far and the best lower bound. The negative value in column *heur/opt* indicates that the solution given by the heuristic in 90 seconds for dense instances is 20% better than the best feasible solution found by CPLEX in 3 hours of computation. In fact, for these instances CPLEX is unable to get good lower bounds within the time limit, and this explains the high value 24.67 of *heur/lb*. The reason for the high computation time at the root node is not only the size of the linear program but mainly the high degeneration of the solutions. This inconvenient is reduced by using the dual simplex algorithm, but still the convergency is slowly achieved even on medium-size instances. We conducted experiments where the linear programs were solved by the interior-point method available in CPLEX, but no advantage was observed using this option.

We have also solved larger instances with the heuristic. However CPLEX was not able to find a feasible solution within the time limit. For this reason we do not report more results in Table 3.

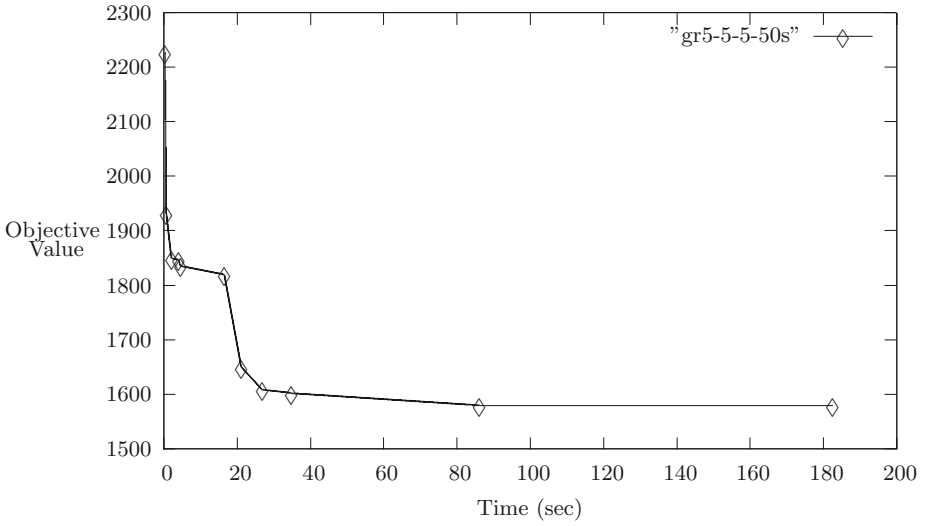


Fig. 1. Evolution of the objective value versus computation time for an instance with $|I| = |J| = 5$, $|H| = 50$ and low arc density

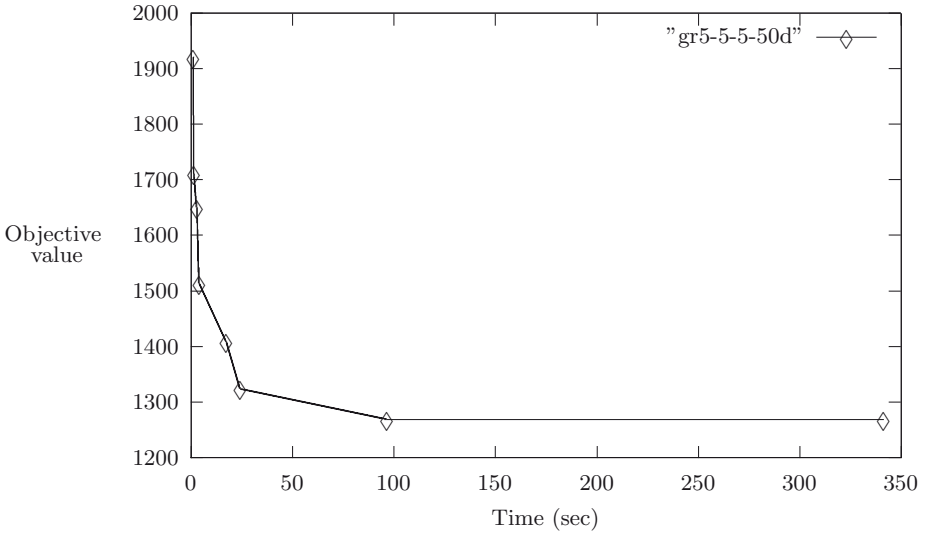


Fig. 2. Evolution of the objective value versus computation time for an instance with $|I| = |J| = 5$, $|H| = 50$ and high arc density

Table 4 shows the development of the solution quality when changing the stopping criterium. We tried three different settings: stop after k iterations without any improvement on the objective value for $k \in \{50, 100, 200\}$. The experiments were conducted on medium-size instances. The results show that the quality of

the heuristic solution improves when k increases, though the computation time increases too. For this reason we think that $k = 10$ is a good option for obtaining high quality solution in reasonable time. The reported results are obtained computing the average over five instances. Observing the whole set of 45 instances, instead, the results show more clearly the consistency of the heuristic in finding optimal solutions when enough time is given. More precisely, four instances were solved to optimality with $k = 10$, 15 with $k = 50$, 20 with $k = 100$, and 26 with $k = 200$.

Finally, Figures 1 and 2 illustrate the development of the objective value respect to the computation time on two CHP instances. Instance $gr5-5-5-50s$ corresponds to an sparse network with 5 origins, 5 destinations and 50 potential hub locations. Instance $gr5-5-5-50d$ is a dense network with also 5 origins, 5 destinations and 50 potential hub locations. Both have been solved with $k = 200$ to display the evolution of the solution on a large time interval. The optimal objective value for $gr5-5-5-50s$ is 1527, and it was not achieved by the heuristic in three minutes of computational time. The optimal value for $gr5-5-5-50d$ is 1268, and it was found in 97 seconds.

5 Conclusions

We have studied the combinatorial optimization problem of deciding on the hubs to be opened in a telecommunications network to send given demands of information from source terminals to destination terminals at minimum cost. We are not assuming that all the hubs are connected by an arc, as happens in papers related to hub location found in the literature. Capacities on the arcs and hubs are considered, and therefore the problem is referred to as the *Capacitated Hub Problem*.

The complexity of our problem is due to the cost for opening hubs, thus managing a zero-one variable for each node. The large number of continuous variables and constraints creates difficulties when applying a general purpose solver. For that reason we have presented a heuristic approach based on solving linear programming relaxations. The approach follows the so-called Iterated Local Search scheme.

The computational experiments on random instances show that the proposed method is highly effective finding near-optimal solutions for this \mathcal{NP} -hard problem. As for the computing times, the results are very good, specially considering the difficulties of CPLEX not only to solve the model, but even to find a feasible solution for medium-size and large instances.

Acknowledgements

This work has been partially supported by “Ministerio de Educación y Ciencia”, Spain (research project TIC2003-05982-C05-02).

References

1. T. Aykin, "Lagrangian relaxation based approaches to capacitated hub-and-spoke network design problems", *European Journal of Operational Research* 79 (1994) 501–523.
2. F. Barahona, "Network design using cut inequalities", *SIAM Journal on Optimization* 6 (1996) 823–837.
3. N. Boland, M. Krishnamoorthy, A.T. Ernst, J. Ebery, "Preprocessing and cutting for multiple allocation hub location problems", *European Journal of Operational Research* 155 (2004) 638–653.
4. J.F. Campbell, "Integer programming formulations of discrete hub location problems", *European Journal of Operational Research* 72 (1994) 387–405.
5. G. Carello, F. Della Croce, M. Ghirardi, R. Tadei, "Solving the hub location problem in telecommunication network design: a local search approach", *Networks* 44 (2004) 94–105.
6. J. Ebery, M. Krishnamoorthy, A. Ernst, N. Boland, "The capacitated multiple allocation hub location problem: Formulations and algorithms", *European Journal of Operational Research* 120 (2000) 614–631.
7. E. Ernst, M. Krishnamoorthy, "Exact and heuristic algorithms for the uncapacitated multiple allocation p -hub problem", *European Journal of Operational Research* 104 (1998) 100–112.
8. E. Ernst, M. Krishnamoorthy, "Solution algorithms for the capacitated single allocation hub location problem", *Annals of Operations Research* 86 (1999) 141–159.
9. H.W. Hamacher, M. Labbé, S. Nickel, T. Sonneborn, "Adapting polyhedral properties from facility to hub location problems", *Discrete Applied Mathematics* 145 (2004) 104–116.
10. K. Holmberg, D. Yuan, "A Lagrangian heuristic based branch-and-bound approach for the capacitated network design problem", *Operations Research* 48 (2000) 461–481.
11. H.R. Lourenço, O.C. Martin, T. Stützle, "Iterated local search". In F. Glover and G.A. Kochenberger, editors, *Handbook of Metaheuristics*, Kluwer's International Series in Operations Research & Management Science, Norwell, 2002.
12. T.L. Magnanti, R.T. Wong, "Network design and transportation planning: models and algorithms", *Transportation Science* 18 (1984) 1–55.
13. G. Mayer, B. Wagner, "HubLocator: an exact solution method for the multiple allocation hub location problem", *Computer & Operations Research* 29 (2002) 715–739.
14. M. O'Kelly, D. Bryan, D. Skorin-Kapov, J. Skorin-Kapov, "Hub network design with single and multiple allocation: A computational study", *Location Science* 4 (1996) 125–138.
15. I. Rodríguez-Martín, J.J. Salazar-González, "Decomposition approaches for a Capacitated Hub Problem", Proceedings of the IBERAMIA 2004 conference, *Lecture Notes in Artificial Intelligence* 3315 (2004) 154–164.
16. D. Skorin-Kapov, J. Skorin-Kapov, M. O'Kelly, "Tight linear programming relaxations of uncapacitated p -hub median problems", *European Journal of Operational Research* 94 (1996) 582–593.

Using Memory to Improve the VNS Metaheuristic for the Design of SDH/WDM Networks

Belén Melián*

Dpto. E.I.O. y Computación
Universidad de La Laguna, Spain
mbmelian@ull.es

Abstract. Variable neighborhood search is among the well studied local search based metaheuristics. It has provided good results for many combinatorial optimization problems throughout the last decade. Based on previous successful applications of this metaheuristic on various network design problems in telecommunications, we further enhance this approach by incorporating adaptive memory mechanisms from the scatter search and tabu search metaheuristics. The heuristics are compared among each other as well as against objective function values obtained from a mathematical programming formulation based on a commercial solver. The problem instances cover a large variety of networks and demand patterns. The analysis carried out in this paper corroborates that there are significant differences between the variable neighborhood search and the hybrid approach.

1 Introduction

The ever-rising data volume demanded by the market makes network planning in order to minimize the necessary investment while meeting customer demands an important task for the network providers. Synchronous Digital Hierarchy (SDH) and Wavelength Division Multiplex (WDM) form the core of many current backbone networks. Many of these networks, especially in Europe, have a general mesh topology. Therefore, we consider an arbitrary mesh of fiber lines (links) connecting the locations of the network providers (nodes) where the traffic demands arise. WDM systems are only used point-to-point, as in most current commercial networks. Common line-speeds for SDH/WDM networks range from 622Mbit/s up to 40Gbit/s per channel. Thus, with state-of-the-art multiplexers that provide 80 or even 160 channels, WDM is currently the fastest commercially available transmission technology for long-range networks. A good overview of the SDH and WDM technology can be found in [7].

SDH requires a dedicated pair of fibers for each transmission, whereas WDM multiplexes several optical signals on a single pair of fibers. The costs for several

* This research has been partially supported by the projects TIN2005-08404-C04-03 (70% of which are FEDER funds) and P.I. 042004/088.

discrete fibers for SDH compared to the multiplexer costs and only a single fiber pair for WDM make WDM suitable for longer distances and high bandwidth demands while discrete SDH lines are cost effective for short lines with limited bandwidth demand. However, the overall optimization problem includes additional equipment like cross-connects, port-cards, amplifiers and regenerators and is much more versatile. The resulting optimization problem is to find a minimum cost combination of the equipment and the routing for a given static demands matrix.

In this paper, we present a Hybrid Variable Neighborhood Search metaheuristic that uses ideas of the scatter search and tabu search metaheuristics and an integer programming formulation as a reference for the described network-planning problem. Section 2 explains the general problem and the integer model. Section 3 contains the detailed description of the hybrid variable neighborhood search. Computational results for various problem instances are given in Section 4. Section 5 provides an outlook on future research.

2 Problem Description and Integer Model

In order to make this paper self contained, this section describes the problem and summarizes the basic model, which is derived from [8].

The optimization problem at hand deals with a set of demands to be routed through an optical network. Associated with each demand is an origin node, a destination node and a size expressed in 2.5Gbit/s units. Each demand can be routed either entirely on one or more discrete fiber pairs or over one or more channels of a WDM system. The demands can be switched from one system to another at the intermediate locations through digital cross-connects (DXC). Optical fibers joining pairs of nodes used to carry the demands through the network are called links, edges or segments. The costs of an edge depend on its length, the required bandwidth and the transmission technology. SDH requires a pair of fibers and maybe additional amplifiers or regenerators for long ranges. WDM links are basically composed of a pair of multiplexer terminals, the fiber pair and possibly also amplifiers. Depending on the number of channels that are actually used, transponder pairs are needed. The number of transponders can be any number between one and the maximum capacity of the multiplexer. The costs of the fiber and the amplification do not depend on the number of channels that are actually used, but they are always per pair of terminals. An SDH line and a single channel of a WDM system each alike occupy one port in the DXC and thus each need one port-card at both ends. The goal of the network planner is to minimize the total cost of the additional fibers and the SDH/WDM equipment.

The network design problem at hand considers that the entire demand between a pair of nodes has to be routed on the same path. All cross-connects have the same number of ports. The given infrastructure is composed of a set of nodes N that represents the switching locations of the provider. A set of undirected edges E connecting these nodes represents the fiber links. Finally, a

set of demands D is given which contains the number of units for each single demand d_{st} from the origin node s to the destination node t .

The cost input consists of the following data:

C_e^{FS}	costs of an SDH line on edge e (fiber, amplifiers, regenerators)
C_e^{FW}	costs of a WDM line on edge e (fiber, amplifiers)
C_e^W	costs of the WDM mux-terminals on edge e
C^O	cost of a basic DXC system
C^C	cost of a WDM channel (a pair of transponders)
C^P	cost of a DXC-port (port-card)

The capacity of the systems is defined as follows:

M^W	capacity of a WDM system (number of wavelengths)
M^O	capacity of a DXC (number of ports)

The spare capacity of previous designs is given by:

g_e	spare WDM channels on WDM systems on segment e
h_n	spare OXC ports on OXC systems at node n

Decision variables:

f_e	number of SDH-systems on edge e
w_e	number of WDM-systems on edge e
v_e	number of channels used in the WDM-systems on edge e
y_n	number of DXCs used in node n
z_{ij}^{st}	1 if demand (s, t) is routed along edge (i, j) ; 0 otherwise

Objective value:

$$\text{Minimize } \sum_{e \in E} ((C_e^{FS} + 2C^P)f_e + (C_e^{FW} + C^W)w_e + (C^C + 2C^P)v_e) + \sum_{n \in N} C^O y_n$$

s.t.:

$$\sum_{j \in N} z_{ji}^{st} - \sum_{j \in N} z_{ij}^{st} = \begin{cases} -1 & i = s \\ 0 & \forall i \neq s, t \\ +1 & i = t \end{cases} \quad \forall (s, t) \in D \quad (1)$$

$$\sum_{(s,t) \in D} d_{st}(z_{ij}^{st} + z_{ji}^{st}) \leq v_e + f_e \quad \forall e \in E \text{ with } i \text{ and } j \text{ adjacent to } e \quad (2)$$

$$v_e \leq M^W w_e + g_e \quad \forall e \in E \quad (3)$$

$$\sum_{e \text{ adjacent to } n} (v_e + f_e) \leq M^O y_n + h_n \quad \forall n \in N \quad (4)$$

$$y_n \geq 0 \text{ and integer} \quad \forall n \in N \quad (5)$$

$$f_e, w_e, v_e \geq 0 \text{ and integer } \forall e \in E \quad (6)$$

$$z_{ij}^{st} \in \{0, 1\} \quad \forall (i, j) \in E, (s, t) \in D \quad (7)$$

Constraints (1) guarantee the flow-conservation. The origin and the destination nodes of each demand both have one adjacent edge that is used; all other nodes have either none or two. Constraints (2) ensure that the demands that each edge carries are less or equal than the total capacity of the installed SDH and WDM systems. Constraints (3) match the number of available WDM-channels with the maximum capacity of the installed WDM-multiplexers. Constraints (4) adjust the capacity of the cross connects of each node to the capacity of its adjacent edges. Constraints (5) and (6) ensure the integrality and non-negativity of the decision variables.

3 Variable Neighborhood Search

Variable Neighborhood Search (VNS) [4] is a metaheuristic for solving combinatorial and global optimization problems based on a simple principle: systematic changes of neighborhoods within the search. Many extensions have been made, mainly to be able to solve large problem instances. However, since the main idea behind variable neighborhood search is to keep the simplicity of the basic scheme, one of the promising areas of research in VNS is the use of memory.

Let \mathcal{N}_k , ($k = 1, \dots, k_{max}$) be a finite set of neighborhood structures, and $\mathcal{N}_k(s)$ the set of solutions in the k^{th} neighborhood of a solution s . Usually, a series of nested neighborhoods is obtained from a single neighborhood by taking $\mathcal{N}_1(s) = \mathcal{N}(s)$ and $\mathcal{N}_{k+1}(s) = \mathcal{N}(\mathcal{N}_k(s))$, for every solution s . This means that a move to the k -th neighborhood is performed by repeating k times a move into the original neighborhood. A solution $s' \in S$ is a *local minimum* with respect to \mathcal{N}_k if there is no solution $s \in \mathcal{N}_k(s') \subseteq S$ better than s' (i.e., such that $f(s) < f(s')$ where f is the objective function of the problem).

In order to solve the problem of designing WDM networks, we propose a Basic Variable Neighborhood Search (BVNS) metaheuristic, whose pseudocode is shown in Figure 1. As indicated in Figure 1, when the value k reaches k_{max} , which is set to the number of demands of the instance, we reset $k = 1$ and a new solution is generated to restart the search.

VNS in its present form relies only on the best solutions currently known to center the search. Knowledge of previous good solutions is forgotten, but might be useful to indicate promising regions not much explored yet. Also characteristics common to many or most good solutions, such as the same path assigned to the same demand in several good solutions reached during the search, could be used to better focus the search procedure. In this paper, we introduce adaptive memory mechanisms provided by both the scatter search [6] and tabu search [3] metaheuristics.

Initialization.

- Select the set of neighborhood structures \mathcal{N}_k , for $k = 1, \dots, k_{max}$.
- Find an initial solution s .
- Choose a stopping condition.

Iterations.

Repeat the following sequence until the stopping condition is met:

- (1) Set $k \leftarrow 1$. Find an initial solution s .
 - (2) Repeat the following steps until $k = k_{max}$:
 - (a) Shaking.
Generate a point s' at random from the k^{th} neighborhood of s ($s' \in \mathcal{N}_k(s)$).
 - (b) Local search.
Apply some local search method with s' as initial solution; denote the so obtained local optimum with s'' .
 - (c) Move or not.
If this local optimum is better than the incumbent, move there ($s \leftarrow s''$), and continue the search with \mathcal{N}_1 ($k \leftarrow 1$); otherwise, set $k \leftarrow k + 1$.
-

Fig. 1. Basic Variable Neighborhood Search

3.1 Application to the Design of WDM Networks

In order to solve the problem of designing WDM networks, in this paper we propose the Hybrid Basic Variable Neighborhood Search (HBVNS) metaheuristic summarized in Figure 2. The original basic variable neighborhood search is hybridized by making use of two adaptive memory mechanisms. We use a reference set of solutions to record both good objective function value solutions and disperse solutions. We keep a reference set consisting of five solutions, being the first three of them the solutions with the least objective function values and the last two solutions disperse ones. The action way in which both the scatter search and tabu search adaptive memories are used are explained below.

A characteristic to take into account when designing a strategy to solve the network design problem at hand is the fact that there is spare capacity installed on the given network topology from previous designs. Then, the notion of base network proposed in [8] is considered. A base, which initially consists of the given network design, is an incomplete network design that does not satisfy the set of demand requirements that a complete design should be capable of handling. As the process iterates, the base network evolves by including additional equipment on segments, which is tentatively added to the original base. With the aim of minimizing the provisioning costs, the routes assigned to each demand to be routed from its origin to its destination, should make a cost-effective use of the spare capacity available on the base network.

The evolution of the base network is linked to an adaptive memory mechanism that keeps track of where new equipment is added in the best solutions recorded during the search. This adaptive memory mechanism makes use of a reference set of solutions, whose notion is the same as the one used in the scatter search

methodology, where it is used as a repository of solutions that are submitted to a combination method. In some basic designs, the reference set contains the best solutions (according to the objective function value) found during the search. However, in more advanced designs, the reference set strategically mixes high quality solutions and diverse solutions, as explained below.

Since the real-world problem at hand does not allow any number of intermediate nodes between the origin and destination of a demand, the described solution approach builds a list of paths for each demand by making use of an efficient implementation of the k -shortest path algorithm [2]. Then, the construction of a solution starts with the selection of a path for each demand requirement. Once each demand is assigned to a path, the cost of the resulting design is calculated. The cost is associated with the equipment that is required to satisfy the demands using the chosen paths.

The MIP formulation described in section 2 allows any path between an origin and a destination of a demand and does not limit the number of intermediate nodes. Based on this fact, the model can be used to find optimal solutions or at least to provide lower bounds of the problem. On the other hand, the hybrid variable neighborhood search developed in this work restricts the number of paths available to route each demand through the network to a certain value reaching an upper bound of the problem.

We now describe in detail all the procedures involved in the hybrid variable neighborhood search summarized in Figure 2.

- *Initialization.* In order to perform the initialization step of the basic variable neighborhood search metaheuristic, the procedure that generates the initial solution and the neighborhood structures must be defined. In the hybrid basic variable neighborhood search proposed in this paper, instead of building a single initial solution, we build a reference set consisting of five solutions. The construction of both the single solution used to run the basic variable neighborhood search and the reference set of solutions for the hybrid method is performed by making use of the constructive procedure proposed in [8].
 - *Initial reference set.* As said above, the initial reference set is constructed by using the constructive procedure proposed in [8], which attempts to assign demands to paths in order to efficiently utilize the spare capacity in the original base network. The initial reference set consists of solutions that are dispersed between them.
 - *Neighborhood structures.* The k^{th} neighborhood of a solution s , $\mathcal{N}_k(s)$, consists of all the solutions that can be reached from s by changing the paths assigned to k different demands. In order to carry out the computational experience, k_{max} is set to the number of demands each instance consists of.
- *Shaking.* This procedure generates a solution s' at random from the k^{th} neighborhood of s ($s' \in \mathcal{N}_k(s)$).
- *Improvement method and Move Decision.* The improvement method is the local search procedure proposed in [8]. In order to run the improvement method from a given solution, the demands are ordered according to their unit cost.

Initialization. Select the set of neighborhood structures \mathcal{N}_k , for $k = 1, \dots, k_{max}$.

(Use of memory) Generate the reference set of solutions, and select the best solution s .

We use as a stopping condition a maximum number of local searches.

Iterations.

Repeat the following sequence until the stopping condition is met:

- (1) Set $k \leftarrow 1$.
 - (2) Repeat the following steps until $k = k_{max}$:
 - (a) *Shaking*. Generate a point s' at random from the k^{th} neighborhood of s ($s' \in \mathcal{N}_k(s)$).
 - (b) *Local search*. Apply some local search method with s' as initial solution; denote the so obtained local optimum with s'' . Update the reference set adding the solution s'' if it is better than the worst solution in the set.
 - (c) *Move or not*. If this local optimum is better than the best solution in the reference set, move there ($s \leftarrow s''$), and continue the search with \mathcal{N}_1 ($k \leftarrow 1$); otherwise, set $k \leftarrow k + 1$.
 - (3) **Use of memory**. ($k = k_{max}$)
 - Rebuild the reference set by using both the objective function value and the diversity criteria.
 - Update the base network by using a tabu list.
 - Rebuild the set of paths for each demand making a cost-effective use of the spare capacity installed on the network.
 - Combine the solutions in the reference set to construct a new current solution s .
 - Set $k \leftarrow 1$.
-

Fig. 2. Hybrid Basic Variable Neighborhood Search

The demand ordering is important because the improvement method, which is based on changing one demand from its current path to another, starts with the demand that has the largest unit cost. The first candidate move is then to reassign the demand that is at the top of the unit cost list. If reassigning this demand leads to an improving move, the move is executed to change the current solution. If an improving move that involves reassigning the first demand in the list cannot be found, then the second demand is considered. The process continues until a demand is found for which a reassignment of path leads to an improving move. If all the demands are examined and no improving move is found, the local search is abandoned. Each time an improvement is achieved while executing the local search procedure, the solution is inserted in the reference set following the quality criterion. Therefore, the improved solution is included in the reference set if it is more cost effective than the worst solution in the set.

- *Use of memory*. If the value of k , which determines the neighborhood structure to be used, reaches the maximum value k_{max} set to the number of demands, then in order to guide the search over promising areas, we make use of two ways of adaptive memory. First of all, we rebuild the reference set by keeping the best three solutions according to the design cost and

replacing the other two solutions in the set by two disperse solutions. These two disperse solutions are built by means of a constructive method that attempts to best utilize the resources in the network while avoiding the use of the paths used for the three best solutions in the reference set. Then, the base network is evolved employing the information embedded in the reference set as proposed in [8]. The main criteria used to evolve the base network relates to the number of times a segment has appeared in the paths assigned to the demands in the reference set solutions. The evolution procedure uses global (referred to the whole search process) and local (referred to the current reference set) information in the form of counters that keep track of the number of channels used in each segment in order to decide where to add equipment to the current base. In this process, tabu search contributes with a short term memory component since we keep a tabu list that records on which segments additional capacity was installed in the last three evolution processes. Since the amount of spare capacity on the current base network has been updated, the paths for each demand are recalculated. The rationale behind rebuilding the reference set of solutions before updating the base network is to keep both goodness and diversity at the time to decide where additional capacity has to be installed.

After rebuilding the reference set, updating the base network and obtaining the k -shortest paths for each demand, a new current solution is obtained by combining the solutions in the reference set as the scatter search metaheuristic proposes. The new current solution is constructed by combining the three solutions in the reference set with the best design costs. If the paths assigned to a given demand are equal in these three solutions, the new solution uses the same path for that demand. Otherwise, the path assigned to the demand is chosen to best utilize the additional capacity installed on the current base network.

4 Computational Results

We have used several problem instances to test the performance of the proposed Hybrid Basic Variable Neighborhood Search metaheuristic that makes use of adaptive memory mechanisms. The results given by this hybrid method are then compared to the results given by the VNS without the use of any memory and to the results provided by CPLEX. However, the aim of this paper is to emphasize how the use of memory to guide the search in a variable neighborhood search algorithm leads to significant improvements in the final design costs. The results obtained by using CPLEX are reported as a reference to test the quality of the solutions found during the search.

The instances used for testing consists of both real (shared by Dr. Leonard Lu of AT&T Labs) and randomly generated instances, which are available under request. The random instances are based on the networks corresponding to the real instances, with the demands randomly generated. The motivation for generating random instances is to study the performance of our methods on instances with

Table 1. Test problem characteristics

Set Name	$ N $	$ E $	$ D $
MetroD	11	16	10, 20, 30, 54
		27	10, 20, 30, 54
		42	10, 20, 30, 48, 54
Extant0D	12	17	15, 19, 21, 44, 66
		33	15, 21, 44, 66
		46	15, 19, 21, 44, 66
Example2D	17	26	27, 36, 79, 81, 135
		68	27, 36, 81, 135
NationalD	50	63	45, 65, 91, 112

various characteristics. In this case, differently sized networks are considered with several densities according to the number of links. Then, several sets of uniform and clustered demands are randomly created. Uniform demands are generated by randomly selecting an origin and a destination, where each pair has the same probability of being selected. Clustered demands are generated selecting a subset of nodes as “high traffic” locations and then generating a demand pattern that has a higher density around those nodes. The costs for SDH and WDM systems depend on the length of the edges in all these instances. Demands are in 2.5 Gbit/s units for all the instances described in this section. The (mixed) integer flow formulation mentioned above was solved by CPLEX in order to obtain optimal solutions as a reference for the metaheuristics. For the larger problem instances, where CPLEX is not able to find optimal solutions, it can at least provide good bounds. All the experiments were carried out on a Pentium 4 with 2.4 Ghz and the CPLEX calculations were performed with CPLEX version 8.1. The size of the branch&bound tree was limited to 400 MB, which was also the termination criterion for the computation. The CPLEX results were given in [5].

The problem instances are summarized in Table 1. For each set, Table 1 shows the name, the number of nodes N , links L , and the different numbers of demands D . Table 2 reports the data regarding the equipment cost used in the solution of the problem instances listed in Table 1.

Table 3 summarizes the comparison between the standard basic variable neighborhood search (BVNS) and the hybrid basic variable neighborhood search (HBVNS), where both procedures were run 10 times for 5000 local searches using up to 50 paths for each demand. Since in real-world applications, there must be a limit in the number of nodes between the origin and destination nodes of a demand, our method builds a list of paths for each demand as said before. In order to carry out our experiments, we have built up to 50 paths for each demand when possible. The design costs reported in this table are the best costs out of the 10 obtained for each instance. The computational times show the time at which the procedures found the best design cost reported. The so performed experiments provide as with a fair way to compare both metaheuristics and to corroborate the effectiveness of using memory in a basic variable neighborhood search procedure, which is the goal of this paper.

Table 2. Description of costs

Constant	Cost	Description
C_e^F	$\$1,400 * length(e)$	Cost of a fiber on a segment e
C_e^W	\$95,000	cost of a WDM unit
C^O	\$120,000	cost of an OXC unit
C^c	\$18,000	channel cost of a WDM unit
C^p	\$10,000	port cost of an OXC unit

The advantage of using memory as explained above to develop an HBVNS is significant. For the smaller instances both metaheuristics reach the same solution. However, as the network size increases, the HBVNS gets better results. The objective function values have improved in 17 out of 38 instances as reported in this table. Although the computational times have increased in the HBVNS compared to the standard BVNS, cost reduction is the main goal in a telecommunications network design problem. Moreover, HBVNS is able to reach five optimal solutions that the former metaheuristic did not find. Note that the results given by CPLEX are lower bounds to the real WDM network design problem, since a node-arc model has all paths implicitly available. In other words, the solution to the model presented above provides a lower bound because the number of intermediate nodes for paths between origin and destination pairs is not bounded. Since the metaheuristics get an upper bound of the problem, for the 15 instances in which HBVNS and CPLEX find the same solution, it is the optimal solution to the problem.

In order to test if we may conclude from sample evidence that the use of memory given by the reference set in the basic variable neighborhood search makes a significant difference in the performance of the overall hybrid approach, we use Wilcoxon Signed Ranks Test [1]. Wilcoxon suggested a T statistic, which has the approximate quantiles given by the normal distribution, under the null hypothesis that there are no significant differences between the two compared procedures. The critical region of approximate size $\alpha = 0.05$ corresponds to all values of T less than -1.6449 . Since in our case $T = -2.8977$, the null hypothesis is rejected and we may conclude that there is a significant difference between the BVNS and our hybrid approach that makes use of adaptive memory mechanisms.

Summarizing the results we may consider the following concerns. The first refers to the use of memory for enhancing the quality of the results of the variable neighborhood search metaheuristic. As an overall observation, we may deduce that the concept of a simple use of memory derived from ideas of the scatter search and tabu search metaheuristics is able to considerably enhance the quality of variable neighborhood search. A final observation refers to the overall necessity of the heuristic concepts. Though CPLEX is capable to produce optimal or at least competitive results for all problem instances examined in this paper, the run-times are almost prohibitively long for the larger instances, and CPLEX is not able to solve considerably larger instances beyond the dimension of those

Table 3. Overview of the computational results

			BVNS		HBVNS		CPLEX		
[N]	[E]	[D]	Cost	Time[s]	Cost	Time [s]	Cost	Time [s]	Bound
11	16	10	4.09	0.02	4.09	0.01	4.09	0.20	opt
		20	4.38	0.03	4.38	0.02	4.38	0.75	opt
		30	8.42	3.70	8.42	0.09	8.42	0.39	opt
		54	14.12	3.51	14.11	0.54	14.03	6.53	opt
	27	10	2.75	0.01	2.75	0.01	2.75	0.29	opt
		20	4.26	0.01	4.26	0.03	4.03	1.09	opt
		30	6.46	0.04	6.46	0.06	6.40	1.76	opt
		54	11.22	1.48	11.22	1.32	10.84	17.34	opt
	42	10	1.96	0.01	1.96	0.03	1.94	0.38	opt
		20	3.08	0.03	3.08	0.03	3.06	1.21	opt
		30	5.38	0.01	5.38	0.02	5.38	7.21	opt
		48	7.11	0.05	7.11	0.06	6.99	10.89	opt
54	8.42	1.31	8.42	0.12	8.35	39.44	opt		
12	17	15	3.69	0.01	3.69	0.01	3.69	0.75	opt
		19	6.26	0.05	6.26	0.29	6.26	8.04	opt
		21	6.21	0.55	6.21	0.53	6.21	3.45	opt
		44	15.05	0.50	14.55	17.02	14.36	96.11	opt
		66	12.59	0.06	12.59	0.61	11.83	81.81	opt
		33	15	3.69	0.5	3.69	0.14	3.69	7
	46	21	6.50	9.31	6.03	23.42	6.03	67	opt
		44	16.06	5.84	14.11	25.09	13.66	4228	opt
		66	15.65	0.19	13.68	45.06	11.77	9024	opt
		15	3.69	0.3	3.69	0.21	3.69	12	opt
		21	6.62	2.10	6.03	2.26	6.03	165	opt
		44	15.53	0.74	14.08	55.84	13.16	6148	opt
66	17.69	0.19	11.77	37.27	11.77	11194	opt		
17	26	27	23.59	41.01	22.47	22.85	22.47	3	opt
		36	82.23	126.90	81.84	187.89	81.84	31	opt
		81	98.34	74.94	97.20	253.12	96.65	470	opt
		135	173.95	140.89	172.62	546.29	170.15	29255	opt
	68	27	20.63	67.52	19.69	116.26	19.27	110	opt
		36	64.20	2.90	64.20	56.81	63.75	28841	opt
		81	79.90	298.45	81.52	401.87	76.66	216806	75.21
		135	145.67	807.56	144.73	661.21	138.40	81098	130.98
50	63	45	37.72	281.58	35.07	298.31	34.07	854	opt
		65	50.38	32.73	49.86	820.22	48.65	77493	45.56
		91	60.52	1112.69	56.95	463.77	55.64	95755	53.19
		112	43.56	77.06	43.97	672.33	42.88	39544	38.67

considered in this paper. These networks clearly show the limits of the CPLEX approach while the heuristics are able to scale much better with the network size and thus are easily applicable to even larger problem instances occurring in various real-world planning problems. Experience from the past shows that CPLEX often cannot even provide a useful bound if networks with more than 100

nodes are considered, implying that it is out of question whether the heuristics are useful for solving the considered class of telecommunications network design problems.

5 Conclusions and Future Research

We have presented a variable neighborhood search metaheuristic for the combined SDH and WDM equipment planning and routing problem. The incorporation of adaptive memory mechanisms from the scatter search and tabu search metaheuristics leads to a further improvement of the results, obtaining significant differences in the design costs of the problem at hand. For smaller problem instances, CPLEX remains the best choice but for large problem instances the hybrid variable neighborhood search is very competitive with respect to the objective values while being much faster than CPLEX.

As future research, we can consider bigger instances with more than 50 nodes, for which in some cases it is not even possible to get feasible solutions using CPLEX. Other possible extension is to analyze which of the adaptive memory mechanisms used in this paper is contributing the most to the effectiveness of our hybrid basic variable neighborhood search.

References

1. W.J. Conover. *Practical Nonparametric Statistics*. John Wiley and Sons, 1999.
2. F. Glover and M. Laguna. Bandwidth packing: A tabu search approach. *Management Science*, 39(4):492–500, 1993.
3. F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.
4. P. Hansen and N. Mladenovic. Variable neighborhood search. In F. Glover and G. A. Kochenberger, editors, *Handbook of Metaheuristics*. 2003.
5. H. Höller, B. Melián, and S. Voß. Applying the pilot method to improve vns and grasp metaheuristics for the design of sdh/wdm networks. apr 2006. Submitted for publication.
6. M. Laguna and R. Martí. *Scatter Search: Methodology and Implementations in C*. Kluwer Academic Publishers, Boston, 2003.
7. B. Lee and W. Kim. *Integrated Broadband Networks*. Artech House, Boston, 2002.
8. B. Melian, M. Laguna, and J.A. Moreno-Perez. Minimizing the cost of placing and sizing wavelength division multiplexing and optical cross-connect equipment in a telecommunications network. *Networks*, 45(4):199–209, 2005.

Multi-level Ant Colony Optimization for DNA Sequencing by Hybridization*

Christian Blum** and Mateu Yábar Vallès

ALBCOM, Dept. Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya, E-08034 Barcelona, Spain
cblum@lsi.upc.edu

Abstract. Deoxyribonucleic acid (DNA) sequencing is an important task in computational biology. In recent years the specific problem of DNA sequencing by hybridization has attracted quite a lot of interest in the optimization community. This led to the development of several metaheuristic approaches such as tabu search and evolutionary algorithms. In this work we propose an ant colony algorithm to resolve this problem. In addition, we apply our algorithm within a multi-level framework which helps in significantly reducing the computation time. The results show that our algorithm is currently among the state-of-the-art methods for this problem.

1 Introduction

Deoxyribonucleic acid (DNA) is a molecule that contains the genetic instructions for the biological development of all cellular forms of life. Each DNA molecule consists of two (complementary) sequences of four different nucleotide bases, namely adenine (A), cytosine (C), guanine (G), and thymine (T). In mathematical terms each of these sequences can be represented as a word from the alphabet $\{A, C, G, T\}$. One of the most important problems in computational biology consists in determining the exact structure of a DNA molecule, called *DNA sequencing*. This is not an easy task, because the nucleotide base sequences of a DNA molecule (henceforth called DNA strands or sequences) are usually so large that they cannot be read in one piece. In 1977, 24 years after the discovery of DNA, two separate methods for DNA sequencing were developed: the chain termination method and the chemical degradation method. Later, in the late 1980's, an alternative and much faster method called *DNA sequencing by hybridization* was developed (see [1,17,13]).

DNA sequencing by hybridization works roughly as follows. The first phase of the method consists of a chemical experiment which requires a so-called DNA array. A DNA array is a two-dimensional grid whose cells typically contain all

* This work was supported by the Spanish CICYT project OPLINK (grant TIN-2005-08818-C04-01), and by the “Juan de la Cierva” program of the Spanish Ministry of Science and Technology of which Christian Blum is a post-doctoral research fellow.

** Corresponding author.

possible DNA strands—called probes—of equal length l . After the generation of the DNA array, the chemical experiment is started. It consists of bringing together the DNA array with many copies of the DNA sequence to be read, also called the target sequence. Hereby, the target sequence might react with a probe on the DNA array if and only if the probe is a subsequence of the target sequence. Such a reaction is called hybridization. After the experiment the DNA array allows the identification of the probes that reacted with target sequences. This subset of probes is called the *spectrum*. Two types of errors may occur during the hybridization experiment:

1. **Negative errors:** Some probes that should be in the spectrum (because they appear in the target sequence) do not appear in the spectrum. A particular type of negative error is caused by the multiple existence of a probe in the target sequence. This cannot be detected by the hybridization experiment. Such a probe will appear at most once in the spectrum.
2. **Positive errors:** A probe of the spectrum that does not appear in the target sequence is called a positive error.

In mathematical terms, the spectrum obtained from the hybridization experiment is not a multiset, that is, each member of the spectrum appears only once in the spectrum. The second phase of DNA sequencing by hybridization consists in the reconstruction of the target sequence from the spectrum. Let us, for a moment, assume that the obtained spectrum is perfect, that is, free of errors. In this case, the original sequence can be reconstructed in polynomial time with an algorithm proposed by Pevzner in [18]. However, as the generated spectra generally contain negative as well as positive errors, the perfect reconstruction of the target sequence is *NP*-hard.

1.1 DNA Sequencing by Hybridization

In order to solve the computational part of DNA sequencing by hybridization, one usually solves an optimization problem of which the optimal solutions can be shown to have a high probability to resemble the target sequence. In this work we consider the optimization problem that was introduced as a model for DNA sequencing by hybridization by Błażewicz et al. in [3]. In fact, this optimization problem—outlined in the following—is a version of the *selective traveling salesman problem*.

Henceforth, let the target sequence be denoted by s_t . The number of nucleotide bases of s_t shall be denoted by n (i.e., $s_t \in \{A, C, G, T\}^n$). Furthermore, the spectrum—as obtained by the hybridization experiment—is denoted by $S = \{1, \dots, m\}$. Remember that each $i \in S$ is an oligonucleotide (i.e., a short DNA strand) of length l (i.e., $i \in \{A, C, G, T\}^l$). In general, the length of any oligonucleotide i is denoted by $l(i)$. Let us now define a completely connected directed graph $G = (V, A)$ over the spectrum, that is, $V = S$ (see also [16]). To each link $a_{ij} \in A$ is assigned a weight o_{ij} , which is defined as the length of the longest DNA strand that is a suffix of i and a prefix of j . Let $p = (i_1, \dots, i_k)$ be a directed path without loops in G . The length of such a path p , denoted by

$l(p)$, is defined as the number of vertices (i.e., oligonucleotides) on the path. In the following we denote by $p[r]$ the r -th vertex in a given path p (starting from position 1). In contrast to the length, the cost of a path p is defined as follows:

$$c(p) \leftarrow l(p) \cdot l - \sum_{r=1}^{l(p)-1} o_{p[r]p[r+1]} \quad (1)$$

The first term sums up the length of the oligonucleotides on the path, and the second term (which is subtracted from the first one) sums up the overlaps between the neighboring oligonucleotides on p . In fact, $c(p)$ is equivalent to the length of the DNA sequence that is obtained by the sequence of oligonucleotides in p . The problem of DNA sequencing by hybridization consists of finding a directed Hamiltonian path p^* in G with $l(p^*) \geq l(p)$ for all possible paths p that fulfill $c(p) \leq n$. In the following we refer to this optimization problem as *sequencing by hybridization (SBH)*. In the following we will denote an SBH problem instance by (G, n) .

As an example consider the target sequence $s_t = \text{ACTGACTC}$. Assuming $l = 3$, the ideal spectrum is $\{\text{ACT}, \text{CTG}, \text{TGA}, \text{GAC}, \text{ACT}, \text{CTC}\}$. Let us assume that the hybridization experiment provides us with the following faulty spectrum $S = \{\text{ACT}, \text{TGA}, \text{GAC}, \text{CTC}, \text{TAA}\}$. See Figure 1 for the corresponding graph G . This spectrum has two negative errors, because ACT should appear twice, but can—due to the characteristics of the hybridization experiment—only appear once, and CTG does not appear at all in S . Furthermore, S has one positive error, because it includes oligonucleotide TAA, which does not appear in the target sequence. An optimal Hamiltonian path in G is $p^* = (\text{ACT}, \text{TGA}, \text{GAC}, \text{CTC})$ with $l(p^*) = 4$ and $c(p^*) = 8$. The DNA sequence that is retrieved from this path is ACTGACTC, which is equal to the target sequence.

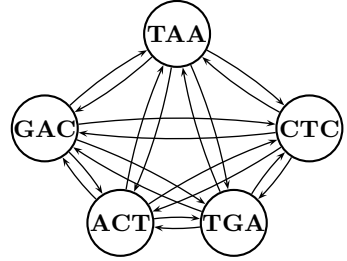


Fig. 1. The completely connected directed graph with spectrum $S = \{\text{ACT}, \text{TGA}, \text{GAC}, \text{CTC}, \text{TAA}\}$ as the vertex set. The edge weights (i.e., overlaps) are not indicated for readability reasons. For example, the weight on the edge from TGA to GAC is 2, because GA is the longest DNA strand that is a suffix of TGA and a prefix of GAC.

1.2 Existing Approaches

The first approach to solve the SBH problem was a branch & bound method proposed in [3]. However, this approach becomes impractical with growing problem size. For example, the algorithm was only able to solve 1 out of 40 different problem instances concerning target sequences with 200 nucleotide bases within one hour. Another argument against this branch & bound algorithm is the fact that an optimal solution to the SBH problem does not necessarily provide a

DNA sequence that is equal to the target sequence. This means that the importance of finding *optimal* solutions is not the same as for other optimization problems. Therefore, the research community has focused on (meta-)heuristic techniques for tackling the SBH problem. In addition to a few simple heuristics (see [3,2]), tabu and scatter search approaches [4,5,6] as well as evolutionary algorithms [7,6,14,11,10] were developed. Moreover, a GRASP method proposed in [15] deals with an easier version of the problem in which the first oligonucleotide of each target sequence is given.

1.3 Our Contribution

In this paper we propose an ant colony optimization (ACO) [12] algorithm for DNA sequencing by hybridization. The choice of ACO is motivated by the fact that neighborhood search based methods do not seem to work very well for this problem. In addition, we propose the application of our ACO approach within a multi-level framework, resulting in a so-called multi-level technique (see, for example, [20,19]). The basic idea of a multilevel technique is a simple one. Starting from some given problem instance, smaller and smaller problem instances are obtained by successive coarsening until some stopping criteria are satisfied. For example, in graph-based problems the coarsening of a problem instance is usually obtained by edge contractions. This creates a hierarchy of problem instances in which the problem instance of a given level is always smaller (or equal) to the problem instance of the next higher level. Then, a solution is computed to the smallest problem instance and successively transformed into a solution of the next higher level until a solution for the original problem instance is obtained. At each level, the obtained solution might be subject to a refinement process. In our case, we will use the ACO algorithm as refinement process at each level.

The organization of the paper is as follows. In Section 2 we describe our ACO algorithm, and in Section 3 we present the multi-level framework. Section 4 is devoted to the experimental evaluation of our approaches. A comparison to the best techniques from the literature is conducted. Finally, in Section 5 we offer conclusions and an outlook on future work.

2 ACO for DNA Sequencing by Hybridization

ACO algorithms are iterative stochastic search techniques which tackle an optimization problem as follows. At each iteration candidate solutions are constructed in a probabilistic way. The probabilistic solution construction is based on a so-called pheromone model (denoted by \mathcal{T}), which is a set of numerical values that encode the algorithms' search experience. After the construction phase, some of the generated solutions are used to update the pheromone values in a way that aims at biasing the future solution construction towards good solutions found during the search process.

2.1 The Objective Function

Before we outline our particular ACO implementation for SBH, we first deal with an issue concerning the objective function. Given a feasible solution p to the problem instance (G, n) ,¹ the objective function value $l(p)$ is the number of olionucleotides in p . This objective function has the following disadvantage when used in a search algorithm. Let p and p' be two solutions with $l(p) = l(p')$ and $c(p) < c(p')$.² Even that the objective function $l(\cdot)$ can not distinguish between p and p' , the intuition is to prefer p , because the DNA sequence it induces is shorter. This implies a higher chance for an extension of p while respecting the constraint $c(p) \leq n$. Therefore, we define a comparison operator $f(\cdot)$ for the purpose of tie-breaking as follows:

$$f(p) > f(p') \Leftrightarrow l(p) > l(p') \text{ or } (l(p) = l(p') \text{ and } c(p) < c(p')) \quad (2)$$

2.2 The Algorithm

Our ACO approach, which is a *MAX-MIN* ant system (*MMAS*) implemented in the hyper-cube framework (HCF) [8], solves the SBH problem as shown in Algorithm 1.. The data structures used by this algorithm, in addition to counters and to the pheromone model \mathcal{T} , are:

- the *iteration-best* solution p_{ib} : the best solution generated in the current iteration by the ants;
- the *best-so-far* solution p_{bs} : the best solution generated since the start of the algorithm;
- the *restart-best* solution p_{rb} : the best solution generated since the last restart of the algorithm;
- the *convergence factor* cf , $0 \leq cf \leq 1$: a measure of how far the algorithm is from convergence;
- the Boolean variable bs_update : it becomes true when the algorithm reaches convergence.

The algorithm works as follows. First, all the variables are initialized, and the pheromone values are set to their initial value 0.5 in procedure *InitializePheromoneValues*(\mathcal{T}). At each iteration, first n_f ants construct a solution each in procedure *ConstructForwardSolution*(\mathcal{T}), and then n_b ants construct a solution each in procedure *ConstructBackwardSolution*(\mathcal{T}). A *forward solution* is constructed from left to right, and a *backward solution* from right to left. Subsequently, the value of the variables p_{ib} , p_{rb} and p_{bs} is updated (note that, until the first restart of the algorithm, it holds that $p_{rb} \equiv p_{bs}$). Fourth, pheromone values are updated via the *ApplyPheromoneUpdate*(cf , bs_update , \mathcal{T} , p_{ib} , p_{rb} , p_{bs}) procedure. Fifth, a new value for the convergence factor cf is computed. Depending on this value, as well

¹ Remeber that G is the completely connected, directed graph whose node set is the spectrum S , and n is the length of the target sequence. A solution p is a path in G .

² Remeber that $c(p)$ denotes the length of the DNA sequence derived from p .

Algorithm 1. ACO for the SBH problem

```

input: a problem instance  $(G, n)$ 
 $p_{bs} \leftarrow \text{NULL}$ 
 $p_{rb} \leftarrow \text{NULL}$ 
 $cf \leftarrow 0$ 
 $bs\_update \leftarrow \text{FALSE}$ 
InitializePheromoneValues( $\mathcal{T}$ )
while termination conditions not satisfied do
  for  $j \leftarrow 1$  to  $n_f$  do
     $p_j \leftarrow \text{ConstructForwardSolution}(\mathcal{T})$ 
  end for
  for  $j \leftarrow n_f + 1$  to  $n_f + n_b$  do
     $p_j \leftarrow \text{ConstructBackwardSolution}(\mathcal{T})$ 
  end for
 $p_{ib} \leftarrow \text{argmax}(f(p_1), \dots, f(p_{n_f+n_b}))$ 
if  $p_{rb} = \text{NULL}$  or  $f(p_{ib}) > f(p_{rb})$  then  $p_{rb} \leftarrow p_{ib}$ 
if  $p_{bs} = \text{NULL}$  or  $f(p_{ib}) > f(p_{bs})$  then  $p_{bs} \leftarrow p_{ib}$ 
ApplyPheromoneUpdate( $cf, bs\_update, \mathcal{T}, p_{ib}, p_{rb}, p_{bs}$ )
 $cf \leftarrow \text{ComputeConvergenceFactor}(\mathcal{T})$ 
if  $cf > 0.9999$  then
  if  $bs\_update = \text{TRUE}$  then
    ResetPheromoneValues( $\mathcal{T}$ )
     $p_{rb} \leftarrow \text{NULL}$ 
     $bs\_update \leftarrow \text{FALSE}$ 
  else
     $bs\_update \leftarrow \text{TRUE}$ 
  end if
end if
end while
output:  $p_{bs}$ 

```

as on the value of the Boolean variable bs_update , a decision on whether to restart the algorithm or not is taken. If the algorithm is restarted, the procedure $\text{ResetPheromoneValues}(\mathcal{T})$ is applied and all the pheromones are reset to their initial value (0.5). The algorithm is iterated until some opportunely defined termination conditions are satisfied. Once terminated the algorithm returns the best-so-far solution p_{bs} . The main procedures of Algorithm 1. are now described in detail.

$\text{ConstructForwardSolution}(\mathcal{T})$: Starting from an empty path $p = ()$, this function constructs a path $p = (i_1, \dots, i_k)$ in G from left to right by adding exactly one oligonucleotide at each construction step. This is done probabilistically using a pheromone model \mathcal{T} , which consists of pheromone values τ_{ij} and τ_{ji} for each

pair $i, j \in S$ ($i \neq j$), that is, to each directed link of G is associated a pheromone value. Additionally, \mathcal{T} comprises pheromone values τ_{0i} and τ_{i0} for all $i \in S$, where 0 is a non-existing dummy oligonucleotide.

Given the current path $p = (i_1, \dots, i_t)$, $S^{\text{av}} = S \setminus \{i_1, \dots, i_t\}$ is the set of available oligonucleotides, that is, the set of oligonucleotides that can be added to p at the next construction step. Such a construction step is performed as follows. First, we compute a desirability value $\mu_{i_t j} := \tau_{i_t j} \cdot [\eta_{i_t j}]^\beta$ for all $j \in S^{\text{av}}$, where $\eta_{i_t j} := o_{i_t j} / (l - 1)$. Hereby, β is a positive constant that we have set to 5 in our experiments. The values $\eta_{i_t j}$ are called *heuristic information*. They are defined such that $\eta_{i_t j} \in [0, 1]$ grows with growing overlap $o_{i_t j}$ between the oligonucleotides i_t and j . Note that when the pheromone values are all equal, the desirability value $\mu_{i_t j}$ is high exactly when $o_{i_t j}$ is high. Then, we generate a so-called *restricted candidate list* $S^{\text{rc1}} \subseteq S^{\text{av}}$ with a pre-defined cardinality cls such that $\mu_{i_t j} \leq \mu_{i_t u}$ for all $j \in S^{\text{av}}$ and $u \in S^{\text{rc1}}$. Then, with probability $q \in [0, 1]$ the next oligonucleotide i_{t+1} is chosen from S^{rc1} such that

$$i_{t+1} := \arg \max_{j \in S^{\text{av}}} \{\mu_{i_t j}\} . \quad (3)$$

Otherwise, the next oligonucleotide i_{t+1} is chosen from S^{rc1} by roulette-wheel-selection according to the following probabilities:

$$\mathbf{p}_{i_t j} := \frac{\mu_{i_t j}}{\sum_{u \in S^{\text{rc1}}} \mu_{i_t u}} \quad (4)$$

Note that q (henceforth called the determinism rate) and cls are important parameters of the algorithm.

The only construction step that is different is the first one, that is, when $p = ()$. In this case, the desirability values are computed as $\mu_{0j} := \tau_{0j} \cdot [\eta_{0j}]^5 \in [0, 1]$ for all $j \in S^{\text{av}}$ (note that $S^{\text{av}} = S$ when $p = ()$). Hereby,

$$\eta_{0j} := \frac{l - o_{\text{pre}(j)} + o_{j \text{ suc}(j)}}{2(l - 1)} , \quad (5)$$

where

$$\text{pre}(j) \leftarrow \arg \max \{o_{i j} \mid i \in S, i \neq j\} , \quad (6)$$

$$\text{suc}(j) \leftarrow \arg \max \{o_{j i} \mid i \in S, i \neq j\} , \quad (7)$$

We henceforth call $\text{pre}(j)$ the best predecessor of j , that is, the oligonucleotide that has the highest overlap with j when placed before j . Accordingly, we call $\text{suc}(j)$ the best successor of j . In both cases, if there is more than one best predecessor (respectively, successor), the first one found is taken. Note that this way of defining the heuristic information favours oligonucleotides that have a very good “best successor”, and at the same time a bad “best predecessor”. The intuition is that the spectrum most probably does not contain an oligonucleotide that is a good predecessor for the first oligonucleotide of the target sequence.

Having defined the desirability value for the first construction step, the further procedure concerning the derivation of the restricted candidate list S^{rc1} and the

choice of one of the oligonucleotides from S^{rci} is the same as outlined above for standard construction steps.

Finally, the construction process stops as soon as $c(p) \geq n$, that is, when the DNA sequence derived from the constructed path p is at least as long as s_t . In case $c(p) > n$, we look for the longest (in terms of the number of oligonucleotides) subpath p' of p such that $c(p') \leq n$, and replace p by p' .

ConstructBackwardSolution(\mathcal{T}): In contrast to **ConstructForwardSolution(\mathcal{T})**, this function constructs solutions from right to left. Hereby—given a partial solution $p = (i_t, \dots, i_1)$ —the desirability values are still computed as if the solution construction were from left to right. For example, the desirability value of adding an oligonucleotide j to the front of p is μ_{ji_t} (instead of μ_{i_tj}). This is done such that for the construction of a solution p the same pheromone values are used, no matter if the solution is constructed from left to right, or from right to left.

ApplyPheromoneUpdate($cf, bs_update, \mathcal{T}, p_{ib}, p_{rb}, p_{bs}$): As usual for \mathcal{MMAS} implementations in the HCF, we use at each iteration a weighted combination of the solutions p_{ib} , p_{rb} , and p_{bs} for updating the pheromone values. The weight of each solution depends on the value of the convergence factor cf and on the Boolean variable bs_update . In general, the pheromone update is performed as follows:

$$\tau_{ij} \leftarrow \tau_{ij} + \rho \cdot (m_{ij} - \tau_{ij}) \quad , \forall \tau_{ij} \in \mathcal{T} \quad , \quad (8)$$

where $\rho \in (0, 1]$ is a constant called learning rate, and m_{ij} is composed as follows:

$$m_{ij} \leftarrow (\kappa_{ib} \cdot \delta_{ij}(p_{ib})) + (\kappa_{rb} \cdot \delta_{ij}(p_{rb})) + (\kappa_{bs} \cdot \delta(p_{bs})) \quad , \quad (9)$$

where κ_{ib} is the weight of solution p_{ib} , κ_{rb} is the weight of solution p_{rb} , κ_{bs} is the weight of solution p_{bs} , and $\kappa_{ib} + \kappa_{rb} + \kappa_{bs} = 1$. Moreover, when $i \neq 0$ and $j \neq 0$, $\delta_{ij}(p)$ is a function that returns 1 in case j is the direct successor of i in p , and 0 otherwise. In case $i = 0$, $\delta_{0j}(p)$ returns 1 in case j is the first oligonucleotide in p , and 0 otherwise. In case $j = 0$, $\delta_{i0}(p)$ returns 1 in case i is the last oligonucleotide in p , and 0 otherwise. After the pheromone update rule (Equation 8) is applied, pheromone values that exceed an upper limit of $\tau_{max} = 0.99$ are set back to τ_{max} , and pheromone values that fall below a lower limit $\tau_{min} = 0.01$ are set back to τ_{min} . This prevents the algorithm from complete convergence.

Equation 9 allows to choose how to schedule the relative influence of the three solutions used for updating pheromones. The exact schedule for the setting of the three solution weights used by \mathcal{MMAS} in the HCF is shown in Table 1. In the early stages of the search (i.e., when $cf < 0.7$), only the iteration-best solution is used. Then, when the value of the convergence factor increases (i.e., $0.7 \leq cf < 0.9$) one third of the total influence is given to the restart-best solution, which then increases to two thirds when $0.9 \leq cf < 0.95$. Eventually, all the influence is given to the restart-best solution (i.e., when $cf \geq 0.95$). Once the value of the convergence factor raises above 0.9999, the Boolean control variable bs_update is set to TRUE, and all the influence is given to the best-so-far solution.

Table 1. Setting of κ_{ib} , κ_{rb} and κ_{bs} depending on the convergence factor cf and the Boolean control variable bs_update

	$bs_update = \text{FALSE}$				$bs_update = \text{TRUE}$
	$cf < 0.7$	$cf \in [0.7, 0.9)$	$cf \in [0.9, 0.95)$	$cf \geq 0.95$	
κ_{ib}	1	2/3	1/3	0	0
κ_{rb}	0	1/3	2/3	1	0
κ_{bs}	0	0	0	0	1

$\text{ComputeConvergenceFactor}(\mathcal{T})$: The convergence factor cf , which is a function of the current pheromone values, is computed as follows:

$$cf \leftarrow 2 \left(\left(\frac{\sum_{\tau_{ij} \in \mathcal{T}} \max\{\tau_{\max} - \tau_{ij}, \tau_{ij} - \tau_{\min}\}}{|\mathcal{T}| \cdot (\tau_{\max} - \tau_{\min})} \right) - 0.5 \right)$$

This formula says that when the algorithm is initialized (or reset) so that all pheromone values are set to 0.5, then $cf = 0$, while when the algorithm has converged, then $cf = 1$. In all other cases, cf has a value in $(0, 1)$.

3 The Multi-level Framework

In [9] we proposed a constructive heuristic called *sub-sequence merger (SM)* for SBH. Instead of constructing a solution from left to right (or from right to left), SM starts from a set of $|S|$ paths, each of which contains exactly one oligonucleotide $i \in S$. In subsequent steps the heuristic merges paths until a path of sufficient size is obtained. We use the same idea for defining a multi-level framework for the ACO algorithm outlined in the previous section.

3.1 Instance Contraction

The first step of a multi-level framework consists in contracting the original problem instance iteratively in order to generate a sequence of smaller and smaller problem instances. In the case of the SBH problem we use the following contraction mechanism (see also Algorithm 2.): At each contraction step we have given a set P of paths in G (in fact, the contraction starts from a set of $|S|$ paths, each of which contains exactly one oligonucleotide $i \in S$). A contraction step consists of merging some of these paths. Hereby, we consider only those paths p and p' where the last oligonucleotide of p and the first one of p' have a fixed overlap, which is different for each construction step; starting from the maximum $l - 1$ and getting reduced step by step. In addition it is required that p' is the unique best successor of p , and that p is the unique best predecessor of p' . The best successor (respectively, predecessor) of a path p are hereby defined as follows:

$$\text{suc}(p) \leftarrow \text{argmax}\{o_{pp'} \mid p' \in P, p' \neq p\} , \quad (10)$$

$$\text{pre}(p) \leftarrow \text{argmax}\{o_{p'p} \mid p' \in P, p' \neq p\} . \quad (11)$$

Algorithm 2. Instance contraction

```

1: input: a problem instance  $(G, n)$ 
2:  $P \leftarrow \{(i) \mid i \in S\}$ 
3:  $stop = \text{FALSE}$ 
4:  $level = 1$ 
5: for  $overlap = l - 1, \dots, 1$  do
6:    $changed = \text{FALSE}$ 
7:   while  $\exists p, p' \in P$  s.t.  $o_{pp'} = overlap$  &  $|S_{\text{suc}}(p)| = 1$  &  $|S_{\text{pre}}(p')| = 1$  &
      $\text{suc}(p) = p' \text{ \& pre}(p') = p$  &  $stop = \text{FALSE}$  do
8:      $changed = \text{TRUE}$ 
9:     Add path  $p'$  to the end of path  $p$ 
10:     $P \leftarrow P \setminus \{p'\}$ 
11:    if  $c(p) \geq n$  then
12:       $stop = \text{TRUE}$ 
13:    end if
14:  end while
15:  if  $stop = \text{FALSE}$  and  $changed = \text{TRUE}$  then
16:     $(G^{level}, n) = \text{GenerateProblemInstance}(P)$ 
17:     $level = level + 1$ 
18:  end if
19: end for
20: output: A sequence of instances  $(G^0, n) = (G, n), (G^1, n), \dots, (G^d, n)$ 

```

Hereby, $o_{pp'}$ is defined as the overlap between the last oligonucleotide of p and the first one of p' . Furthermore, in Algorithm 2, $S_{\text{suc}}(p)$ is defined as the set of best successors of p , that is, $S_{\text{suc}}(p) \leftarrow \{p' \in P \mid o_{pp'} = o_{p \text{ suc}(p)}\}$; and $S_{\text{pre}}(p)$ is defined as the set of best predecessors of p , that is, $S_{\text{pre}}(p) \leftarrow \{p' \in P \mid o_{p'p} = o_{\text{pre}(p)p}\}$. The idea of this restriction is produce (if possible) error free sub-sequences of the original target sequence.

Each contraction step leads to a new set of paths P from which a new (smaller) problem instance is generated in function $\text{GenerateProblemInstance}(P)$. This is done by deriving from each path $p \in P$ the corresponding DNA strand. This mechanism generates a sequence $(G^0, n) = (G, n), (G^1, n), \dots, (G^d, n)$ of smaller and smaller problem instances. (G^d, n) denotes the smallest instances that can be obtained (that is, a further construction step would produce a path p with $c(p) \geq n$). Note that all these problem instances have the same target sequence. Moreover, a solution to any of these instances can directly be seen as a solution to any of the other instances.

3.2 Application of ACO in the Multi-level Framework

The application of the ACO algorithm proposed in Section 2.2 in the multi-level framework works as follows. Given the sequence $(G^0, n) = (G, n), (G^1, n), \dots, (G^d, n)$ of problem instances, ACO is first applied to the smallest instance (G^d, n) . Subsequently, ACO is applied in the given order to all problem instances

$(G^{d-1}, n), \dots, (G^0, n)$. Hereby we always use the best solution of the ACO algorithm found for an instance (G^{r-1}, n) as first best-so-far solution for the application of ACO to the instance (G^r, n) . As stopping condition for the whole procedure we use a CPU time limit. The given CPU time is distributed such that the application of ACO to an instance (G^r, n) is always allocated the double of the CPU time that is allocated for the application of ACO to instance (G^{r-1}, n) . Due to the fact that instance (G^{r-1}, n) is smaller than instance (G^r, n) it is reasonable to allocate more time to (G^r, n) . For the application of ACO to an instance (G^r, n) we use two stopping conditions: (1) the allocated CPU time, and (2) a maximum number of iterations without improving the best-so-far solution. Whenever one of the two conditions is fulfilled the application of ACO at the corresponding level is terminated, and the application to the next level starts. Note that the use of the second stopping condition implies that the last application of ACO (that is, the application to the original instance (G^0, n)) may use all the remaining CPU time, which is sometimes more than the allocated CPU time. Moreover, the second stopping condition is not used for the last application of ACO. Finally, for all the experiments outlined in the following section we have set the maximum number of iterations without improvement to 100. In general, this may be an important parameter of the algorithm whose tuning is part of future work. In our current version we performed a tuning by hand.

4 Experimental Evaluation

We implemented our approach in ANSI C++ using GCC 3.2.2 for compiling the software. Our experimental results were obtained on a PC with an AMD64X2 4400 processor and 4 Gb of memory, running Debian Linux.

A wide-spread set of benchmark instances for DNA sequencing by hybridization was introduced by Błażewicz and colleagues.³ It consists of 40 real DNA sequences of length 109, 209, 309, 409, and 509 (altogether 200 instances). Based on real hybridization experiments, the spectra were generated with probe size $l = 10$. All spectra contain 20% negative errors as well as 20% positive errors. For example, the spectra concerning the target sequences of length 109 contain 100 oligonucleotides of which 20 oligonucleotides do not appear in the target sequences. Therefore, an optimal solution contains 80 (respectively, 160, 240, 320, or 400) oligonucleotides.

4.1 Tuning of ACO

First we performed tuning experiments in order to fix the free parameters of the ACO algorithm: the candidate list size $cls \in \{2, 3, 5, 10, \text{all}\}$, the determinism rate $q \in \{0.0, 0.5, 0.75, 0.9, 0.95\}$, and the number of forward solutions, respectively backward solutions, $(n_f, n_b) \in \{(6, 0), (3, 3), (0, 6)\}$. Altogether, this results in 75 different settings of the ACO algorithm. We applied ACO with all 75 settings 10 times to each of the 200 problem instances, allowing 2 seconds for

³ They can be obtained at <http://bio.cs.put.poznan.pl>.

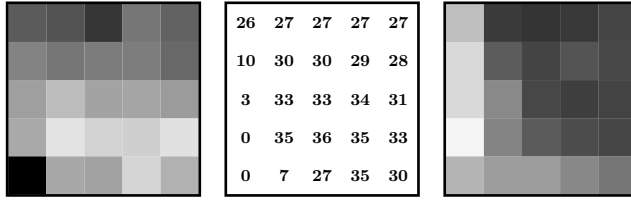
each application concerning the the instances with spectrum size 100 (respectively, 10, 50, 100, or 200 seconds for the bigger instances). Then, for each of the 5 instance groups we produced a summary of the results obtained by averaging over the best run (out of 10) for each of the 40 instances of the group. These summarized tuning results are shown graphically for the biggest instances (spectrum size 500) in Figure 2. Three different measures are considered in this figure: the average global similarity score (see below), the number of instances solved to optimality (out of 40), and the average computation time needed to reach the best solution found for each instance. Hereby, the global similarity score is a measure obtained by comparing the computed DNA sequences with the target sequences. We used the Needleman-Wunsch algorithm for global alignment with the following parameter settings: +1 for a match of oligonucleotides, -1 for a mismatch or a gap.⁴

The results in Figure 2 allow the following conclusions. When determinism is high and the candidate list is small, the algorithm can produce very good results in a short time. However, it pays off spending a little more time (by increasing the candidate list size, for example, to 10). This improves the results while maintaining short running times. Another important observation is that the setting $(n_f, n_b) = (3, 3)$ (that is, using forward as well as backward ants) generally improves over only using ants of one direction. Therefore, we decided to use the following settings for all the remaining experiments: $cls = 10$, $det = 0.9$, and $(n_f, n_b) = (3, 3)$.

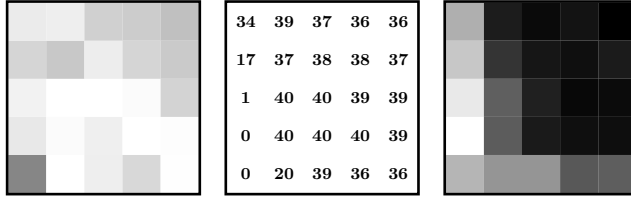
4.2 Final Experiments

We applied ACO as well as ACO in the multi-level framework (henceforth denoted by ML-ACO) to each of the 200 problem instances 10 times. From the best run for each instance we produced a summary of the results averaged over the 40 instances for each of the 5 instance groups. The results of ACO are shown in Table 2 (a). The second table row contains the average solution quality (i.e., the average number of oligonucleotides in the constructed paths). Remember that the optimization objective of the SBH problem is to maximize this value. The third table row provides the number (out of 40) of problem instances solved to optimality, that is, the number of instances for which a path of maximal length could be found. The fourth and fifth table row provide average similarity scores obtained by comparing the computed DNA sequences with the target sequences. The average scores in the fourth table row are obtained by the Needleman-Wunsch algorithm, which is an algorithm for global alignment. In contrast, the average scores that are displayed in the fifth table row are obtained by the application of the Smith-Waterman algorithm, which is an algorithm for local alignment. The local alignment scores are given for completeness. Both algorithms were applied with the following parameters: +1 for a match of oligonucleotides, -1 for a mismatch or a gap. Finally, the sixth table row provides the average computation times for solving one instance (in seconds).

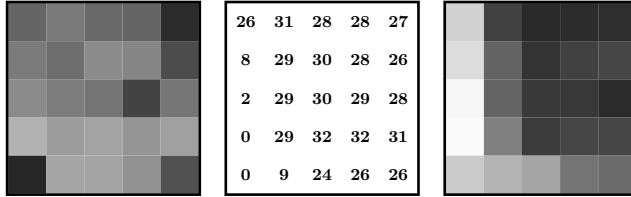
⁴ Remember in this context that an optimal solution to the SBH problem does not necessarily correspond to a DNA sequence that is equal to the target sequence.



(a) Tuning results for $n_f = 6$ (that is, 6 forward solutions per iteration), and $n_b = 0$ (that is, no backward solutions per iteration).



(b) Tuning results for $n_f = 3$ and $n_b = 3$.



(c) Tuning results for $n_f = 0$ and $n_b = 6$.

Fig. 2. Tuning results of ACO for the 40 instances with spectrum size 500. Each square of the 9 5x5 matrices corresponds to one algorithm setting. The matrix rows correspond to the 5 values of the candidate list size (that is (from top to bottom), $cls \in \{2, 3, 5, 10, \text{all}\}$), and the columns correspond to the determinism rate (that is (from left to right), $q \in \{0.0, 0.5, 0.75, 0.9, 0.95\}$). The first matrix of each subfigure corresponds to the values of the global similarity score, the second matrix shows the number of instances solved to optimality, and the third matrix visualizes the computation times. For the 3 matrices on the left holds: The lighter the color, the better then algorithm setting. For the 3 matrices on the right holds: The darker the color, the faster the algorithm setting.

The results show the following. ACO is the first algorithm that is able to solve all 200 problem instances to optimality, which does not mean that all produced DNA sequences are identical to the target sequences (see the similarity scores). Figure 3 shows a comparison of the results of ACO with the results of the best metaheuristics from the literature. Hereby, EA1, EA2, and EA3 are evolutionary algorithms proposed in [7,6], respectively [14] and [11]. TS is a tabu search approach and TS/SS a tabu search approach combined with scatter search proposed in [5]. The results show that only EA2 produces DNA sequences with similarly high global similarity scores. Concerning the number of instances

Table 2. Results of (a) ACO and (b) ML-ACO for the instances by Błażewicz et al.

Spectrum size	100	200	300	400	500
Average solution quality	80	160	240	320	400
Solved instances	40	40	40	40	40
Average similarity score (global)	108.40	208.13	297.78	401.93	503.60
Average similarity score (local)	108.70	208.60	304.98	403.63	503.93
Average computation time (sec)	0.14	1.86	5.09	15.72	38.33

(a) Results of ACO

Spectrum size	100	200	300	400	500
Average solution quality	80	160	240	320	400
Solved instances	40	40	40	40	40
Average similarity score (global)	108.40	208.35	301.05	403.45	503.60
Average similarity score (local)	108.70	208.68	306.05	403.85	503.93
Average computation time (sec)	0.005	0.41	0.41	4.97	7.85

(b) Results of ML-ACO

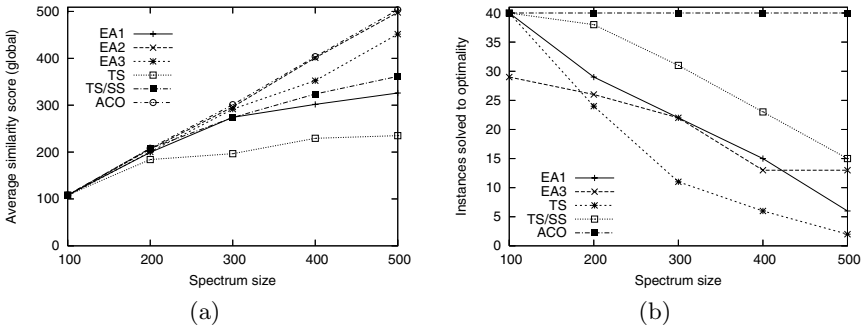


Fig. 3. Comparison of ACO with the best metaheuristics from the literature concerning (a) the global average similarity score obtained, and (b) the number of instances solved to optimality. The comparison concerns the instances of Błażewicz et al. Note that for EA2 the number of solved instances is not given in the literature.

solved to optimality, ACO is clearly superior to the other approaches. However, note that this measure is not given for EA2 in the literature.

Finally, we also applied ML-ACO (in the same way as ACO) to all 200 problem instances. The results are shown in Table 2 (b). ML-ACO also solves all problem instances to optimality. Moreover, the obtained average similarity scores are comparable to the ones produced by ACO. The difference is in the computation time. The application of ACO in the multi-level framework substantially reduces the computation time. More in detail, the computation times are up to 28 times lower (concerning the smallest problem instances). In the worst case (see problem instances with spectrum size 400), the computation times are about 3 times lower.

5 Conclusions

We proposed an ant colony optimization algorithm for DNA sequencing by hybridization. The results show that our algorithm is among the state-of-the-art algorithms proposed in the literature. Moreover, we presented a framework for the application of our ant colony optimization algorithm in multiple levels, a so-called multi-level framework. The results show that the application of our ant colony optimization approach in the multi-level framework results in a substantial CPU time reduction.

Future work consists in a more detailed study of the multi-level framework, and in the application of our approach to large-scale instances. In particular the latter is important, because biologists are often faced with spectra of several 10000 oligonucleotides. We suppose that when applied to larger problem instances, the multi-level framework may not only reduce the necessary CPU time, but may also improve the quality of the obtained solutions.

References

1. W. Bains and G. C. Smith. A novel method for nucleic acid sequence determination. *Journal of Theoretical Biology*, 135:303–307, 1988.
2. J. Błażewicz, P. Formanowicz, F. Guinand, and M. Kasprzak. A heuristic managing errors for DNA sequencing. *Bioinformatics*, 18(5):652–660, 2002.
3. J. Błażewicz, P. Formanowicz, M. Kasprzak, W. T. Markiewicz, and J. Weglarz. DNA sequencing with positive and negative errors. *Journal of Computational Biology*, 6:113–123, 1999.
4. J. Błażewicz, 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*, 125:257–265, 2000.
5. J. Błażewicz, F. Glover, and M. Kasprzak. DNA sequencing—Tabu and scatter search combined. *INFORMS Journal on Computing*, 16(3):232–240, 2004.
6. J. Błażewicz, F. Glover, and M. Kasprzak. Evolutionary approaches to DNA sequencing with errors. *Annals of Operations Research*, 138:67–78, 2005.
7. J. Błażewicz, M. Kasprzak, and W. Kuroczycki. Hybrid genetic algorithm for DNA sequencing with errors. *Journal of Heuristics*, 8:495–502, 2002.
8. C. Blum and M. Dorigo. The hyper-cube framework for ant colony optimization. *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, 34(2):1161–1172, 2004.
9. C. Blum and M. Yábar Vallès. New constructive heuristics for DNA sequencing by hybridization. Technical Report LSI-06-23-R, LSI, Universitat Politècnica de Catalunya, Barcelona, Spain, 2006.
10. C. A. Brizuela, L. C. González, and H. J. Romero. An improved genetic algorithm for the sequencing by hybridization problem. In G. R. Raidl et al., editors, *Proceedings of the EvoWorkshops – Applications of Evolutionary Computing: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoMUSART, and EvoSTOC*, volume 3005 of *Lecture Notes in Computer Science*, pages 11–20. Springer Verlag, Berlin, Germany, 2004.

11. T. N. Bui and W. A. Youssef. An enhanced genetic algorithm for DNA sequencing by hybridization with positive and negative errors. In K. Deb et al., editors, *Proceedings of the GECCO 2004 – Genetic and Evolutionary Computation Conference*, volume 3103 of *Lecture Notes in Computer Science*, pages 908–919. Springer Verlag, Berlin, Germany, 2004.
12. M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, Boston, MA, 2004.
13. R. Drmanac, I. Labat, R. Brukner, and R. Crkvenjakov. Sequencing of megabase plus DNA by hybridization: Theory of the method. *Genomics*, 4:114–128, 1989.
14. T. A. Endo. Probabilistic nucleotide assembling method for sequencing by hybridization. *Bioinformatics*, 20(14):2181–2188, 2004.
15. E. R. Fernandes and C. C. Ribeiro. Using an adaptive memory strategy to improve a multistart heuristic for sequencing by hybridization. In S. E. Nikolettseas, editor, *Proceedings of WEA 2005 – 4th International Workshop on Experimental and Efficient Algorithms*, volume 3503 of *Lecture Notes in Computer Science*, pages 4–15. Springer Verlag, Berlin, Germany, 2005.
16. R. M. Idury and M. S. Waterman. A new algorithm for DNA sequence assembly. *Journal of Computational Biology*, 2(2):291–306, 1995.
17. Y. P. Lysov IuP, V. L. Florentiev, A. A. Khorlin, K. R. Khrapko, and V. V. Shik. Determination of the nucleotide sequence of DNA using hybridization with oligonucleotides. a new method. *Doklady Akademii nauk SSSR*, 303:1508–1511, 1988.
18. P. A. Pevzner. l-tuple DNA sequencing: Computer analysis. *Journal of Biomolecular Structure and Dynamics*, 7:63–73, 1989.
19. C. Walshaw. Multilevel refinement for combinatorial optimization problems. *Annals of Operations Research*, 131, 2004.
20. C. Walshaw and M. Cross. Mesh partitioning: A multilevel balancing and refinement algorithm. *SIAM Journal on Scientific Computing*, 22(1):63–80, 2000.

Hybrid Approaches for Rostering: A Case Study in the Integration of Constraint Programming and Local Search

Raffaele Cipriano¹, Luca Di Gaspero², and Agostino Dovier¹

¹ Dip. di Matematica e Informatica

raffaele.cipriano@gmail.com, dovier@dimi.uniud.it

² Dip. di Ingegneria Elettrica, Gestionale e Meccanica

Università di Udine, via delle Scienze 208, I-33100, Udine, Italy

l.digaspero@uniud.it

Abstract. Different approaches in the hybridization of constraint programming and local search techniques have been recently proposed in the literature. In this paper we investigate two of them, namely the employment of local search to improve a solution found by constraint programming and the exploitation of a constraint model to perform the exploration of the local neighborhood. We apply the two approaches to a real-world personnel rostering problem arising at the department of neurology of the Udine University hospital and we report on computational studies on both real-world and randomly generated structured instances. The results highlight the benefits of the hybridization approach w.r.t. their component algorithms.

1 Introduction

Constraint Satisfaction Problems (CSPs) are a useful formalism for modeling many real problems, either discrete or continuous. Remarkable examples are planning, scheduling, timetabling, and so on. A CSP is generally defined as the problem of associating values (taken from a set of domains) to variables subject to a set of constraints. A solution of a CSP is an assignment of values to all the variables so that the constraints are satisfied. In some cases not all solutions are equally preferable, but we can associate a cost function to the variable assignments. In these cases we talk about Constrained Optimization Problems (COPs), and we are looking for a solution that (without loss of generality) minimizes the cost value. The solution methods for CSPs and COPs can be split into two categories:

- *Complete methods*, which systematically explore the whole solution space in search of a feasible (for CSPs) or an optimal (for COPs) solution.
- *Incomplete methods*, which rely on heuristics to focus on interesting areas of the solution space with the aim of finding a feasible solution (for CSPs) or a “good” one (COPs).

Constraint programming (CP) languages [2] are usually based on complete methods that analyze the search space alternating deterministic phases (constraint propagation) and non-deterministic phases (variable assignment), exploring implicitly or explicitly the whole search space. Local search (LS) methods [1], instead, rely on the definition of “proximity” (or neighborhood) and they explore only specific areas of the search space. Local search method, concentrating on some parts of the search space, can approximate optimal solutions in shorter time.

Two major types of approaches for combining the abilities of constraint programming and local search are presented in the literature [6,7] (in [8] constraint programming and local search are hybridized in a more liberal context):

1. a systematic-search algorithm based on constraint programming can be improved by inserting a local search algorithm at some point of the search procedure, e.g.:
 - (a) at a leaf (i.e., on complete assignments or on an internal node (i.e., on a partial assignment) of the search tree explored by the constraint programming procedure, in order to improve the solution found;
 - (b) at a node of the search tree, to restrict the list of child-nodes to explore;
 - (c) to generate in a greedy way a path in the search tree;
2. a local search algorithm can benefit of the support of constraint programming, e.g.:
 - (a) to analyze the neighborhood and discarding the neighboring solution that do not satisfy the constraints;
 - (b) to explore a fragment of the neighborhood of the current solution;
 - (c) to define the search of the best neighboring solution as a problem of constrained optimization (COP).

In this work we adopt the two hybrid techniques 1(a) and 2(b) and we apply them to a hospital personnel rostering problem. In the first approach we employ constraint programming for searching an initial feasible solution and subsequently we improve it by means of local search, using classical algorithms like hill climbing, steepest descent and tabu search. In the second approach we devise a local search algorithm (called *hybrid steepest descent*) that exploits a constraint programming model for the exploration of neighborhood fragments.

The employment of constraint programming for finding an initial feasible solution exhibits several advantages w.r.t. generating an initial random solution. Indeed, constraint programming allows us to find in short times (about 5-10 seconds) a feasible solution, providing a good starting point for local search.

The local search methods that exploits the constraint programming neighborhood model has evidenced how constraint programming can be used in the exhaustive exploration of neighborhood fragments, obtaining competitive local search procedures; furthermore, this approach is favorable only with huge search spaces. since for small problems the high computational overhead of the constraint programming is not payed back by remarkable improvements of the cost function.

The remainder of the paper is organized as follows: in Section 2 we present the formalization of the rostering problem analyzed. In Section 3 we provide

some details about the implementation of the hybrid methods and in Section 4 we report the comparisons among the different solution techniques employed. Possible future directions of research are discussed in Section 5.

2 Hospital Personnel Rostering

The personnel *rostering* problem consists in the assignment of personnel requirements (usually expressed in terms of shift types) to qualified personnel over a certain planning period. The goal is to find a feasible assignment which minimizes a suitable objective function. Finding high-quality solutions to this problem is of extreme importance in knowledge-intensive labors, where the very specialized skills of personnel make impossible to exchange the duties among persons thus hardening the problem. This situation especially arises in hospital departments, which need an optimal schedule of the workforce that balances the trade-off between internal and external requirements such as the fair distribution of the workload among the different doctors and the assurance of a constant and efficient medical service to citizens.

Although in the last 30 years several studies proposes different solutions to this problem (e.g., by means of mathematical programming, multi-objective programming, constraint programming, expert systems, heuristic and meta-heuristic methods, see [3] for a comprehensive review), at present in Italian hospitals the problem is usually solved with pencil-and-paper by a doctor (*self-scheduling*). In this work we studied a particular family of rostering problems we called **Neu-Rostering** (NR for short), which model the personnel assignment problem of the department of Neurology of the University Hospital of Udine.

The problem is described in details in the following. Given m doctors, n days (the *temporal horizon*) and k possible shifts, the NR problem consists in assigning to each doctor the shifts to cover during the temporal horizon considered. In particular, for each day some shifts must be covered, based on the type of day (a workday or in the weekend) and of the weekday (some shifts are required only on Mondays, other on Tuesdays, and so on). Each shift must be assigned to one (and only one) doctor. Some shifts (e.g., the most general ones like urgent calls shifts) can be covered by any doctor, while others can be covered only by a restricted number of doctors on the basis of their competence. Every doctor can specify a list of days of the *temporal horizon* in which he/she is not available (e.g., because of days off, conferences, courses, teaching, ...). Moreover, there are a set of “temporal” restrictions that regulate the coverage of the shifts: every shift has fixed working times, every doctor can be assigned to consecutive shifts but, in this case, there is an upper bound on the number of consecutive working hours that cannot be exceeded, a doctor should have a rest between two assigned shifts.

2.1 Formulation of NR as CSP

Let $E = \{e_1, \dots, e_k\}$ be the set of all possible shifts; we consider the variables G_1, \dots, G_n , one for each of the n days, where $G_i \subseteq E$ is defined as follows:

$$G_i = \{e_k \in E \mid e_k \text{ is a shift to be covered on the day } i\}$$

The skills of each doctor $j \in 1..m$ are encoded in a set $M_j \subseteq E$, defined as follows:

$$M_j = \{e_k \in E \mid e_k \text{ is a shift that doctor } j \text{ is allowed to cover}\}$$

Furthermore, let consider m sets $F_j \subseteq \{1, \dots, n\}$ that represent the days of the temporal horizon in which doctor j is not available.

Let $C \subseteq E \times E$ be the set of pairs (e_i, e_j) of shifts that *can* be covered in the same day by the same doctor. Then, we define as T the set of shift sets (singletons or doubletons) that can be covered by a doctor in the same day:

$$T = \{\{e_i\} \mid e_i \in E\} \cup \{\{e_i, e_j\} \mid (e_i, e_j) \in C\}$$

Let $S \subseteq E \times E$ be the set of pairs of shifts (e_i, e_j) that can be covered by the same doctor in consecutive days.

The problem input is defined by the sets generated above. In order to coding the problem constraints we consider for all $i \in 1..n, j \in 1..m$ the variables $O_{i,j}$ whose domains are the sets $T \cup \{\emptyset\}$. $O_{i,j} = \text{set of shifts}$ means that in day i doctor j will cover the set of shifts indicated. An assignment to these variables is an (admissible) *solution* if and only if:

1. $\forall i \in 1..n \forall j \in 1..m \quad O_{i,j} \subseteq M_j$ (*competence*: each doctor can only cover shifts he/she is qualified for);
2. $\forall i \in 1..n \forall j \in 1..m \quad i \in F_j \Rightarrow O_{i,j} = \emptyset$ (*availability*: each requirements for days off is satisfied);
3. $\forall i \in 1..n \forall j \in 1..m \quad O_{i,j} \in T$ (*max hours*: each assignment is coherent with the maximum number of hours per day);
4. $\forall i \in 1..n - 1 \forall j \in 1..m \quad \forall t \in O_{i,j} \forall t' \in O_{i+1,j} \quad (t, t') \in S$ (*legal rules*: there is the suitable distance between consecutive shifts for the same doctor);
5. $\forall i \in 1..n \quad \bigcup_{j=1}^m O_{i,j} = G_i$ (*coverage*: all required shifts are covered);
6. $\forall i \in 1..n \forall j_1 \in 1..m \forall j_2 \in 1..m (j_1 \neq j_2 \rightarrow O_{i,j_1} \cap O_{i,j_2} = \emptyset)$ (*mutual exclusion*: each shift is covered by at most one doctor).

Let us observe that our encoding of the NR problem uses $m \cdot n \cdot k$ Boolean variables with the following intuitive meaning: $\forall i \in 1..m \forall j \in 1..n \forall z \in 1..k \quad X_{i,j,z} = 1$ if and only if doctor i covers shift z in the day j . Let us observe that establish the existence of a solution to an instance of NR is NP-complete (by means of a straightforward encoding of 3-GRAPHCOLORING).

2.2 COP Model for NR

Our NR implementation is endowed with an objective function used to model some soft constraints associated to the problem and to choose one solution w.r.t. others. This function has been obtained by eliciting information from the manager of the Neurology Dept. and it is based on five parameters.

Weekend, Nights, and Guards. One of the objectives is to balance the work of the doctors in the week-ends. In the week-ends there are only two types of shifts, denoted by Urgent calls Morning and Afternoon (UMUP) and Urgent calls Night (UN). We looked for an expression that assumes high values when shifts are badly distributed among the doctors. Given a doctor i we propose to sum all the values (actually Boolean values are seen as integer values here) related to shifts UMUP and UN in the weekends and in the holidays:

$$\text{Wk} = \sum_{i=1}^m (\sum_{j \in GWE, z \in TWE} X_{i,j,z})^2$$

In the above formula, i ranges over doctors, j over days to be selected in the set GWE of days in weekends and holidays (e.g., if December 1st is on Monday, then $n = 31$ and $GWE = \{6, 7, 13, 14, 20, 21, 25, 27, 28\}$), and z ranges over shifts in the set TWE (in this case $TWE = \{\text{UMUP}, \text{UN}\}$). In order to balance the amounts of night shifts UN and of Urgent calls morning UM (save those in the weekends and holidays, already considered in Wk—shifts UMUP) we define the two following formulas in analogous way:

$$\text{Nt} = \sum_{i=1}^m (\sum_{j \in 1..n \setminus GWE} X_{i,j,\text{UN}})^2, \text{Gu} = \sum_{i=1}^m (\sum_{j \in 1..n \setminus GWE} X_{i,j,\text{UM}})^2$$

Undesired Pairs. Here we take care of the number of days where a doctor works either in a morning and in a afternoon shift. This is highly undesirable for doctors that work both in the public hospital and as private professionals. We define the following formula for the variable Dp:

$$\text{Dp} = \sum_{i \in 1..m, j \in 1..n} \left(\sum_{z \in \text{AM}} X_{i,j,z} \cdot \sum_{z \in \text{PM}} X_{i,j,z} \right)$$

where AM (resp., PM) is the set of all morning (resp., afternoon) shifts. Let us observe that the product of the two inner sums assume value 1 only when a doctor is employed either in the morning or in the afternoon and 0 elsewhere (values greater than 1 are forbidden by max hours constraints).

Consecutive Shifts. Here we consider situations where a doctor is employed in the same type of shifts for three consecutive days. Some of these sequences are penalize (we count as 1), other (e.g. RMu and RMg) are encouraged (we count as -1). Therefore, we assign to Cn the following formula, where E is the set of all possible shifts:

$$\text{Cn} = \sum_{\substack{i \in 1..m, j \in 1..n-2, \\ z \in E \setminus \{\text{RMu}, \text{RMg}\}}} (X_{i,j,z} X_{i,j+1,z} X_{i,j+2,z}) - \sum_{\substack{i \in 1..m, j \in 1..n-2, \\ z \in \{\text{RMu}, \text{RMg}\}}} (X_{i,j,z} X_{i,j+1,z} X_{i,j+2,z})$$

Objective Functions. Finally, we assign a weight to each one of the just defined five variables in order to balance solutions to NR. In the case studied, where $m = 20$, $k = 28$, and $n \in 28..31$ (for a temporal horizon of one month there are more or less 16000 variables) is the following:

$$\text{FObj} = 50\text{Wk} + 40\text{Nt} + 30\text{Dp} + 20\text{Gu} + 10\text{Cn}$$

Weights have been chosen in the following way. Starting from the by hand computed solutions for the year 2005, we deduced the holidays and the various constraints required by the doctors. Then we generated some sequences of solutions and showed them to the responsible of the Neurology. We modified the weights using his feedback in such a way that minimal values of the function are associated to more preferable solutions.

3 Implementation

In this work, for the solution of the NR problem we adopt two hybrid techniques, which integrate constraint programming and local search. In the first approach we employ constraint programming for searching an initial feasible solution and subsequently we improve it by means of local search. In the second approach we devise a local search algorithm that exploits a constraint programming model for the exploration of the neighborhood.

3.1 Application Architecture

In order to solve the NR we design a software tool made up of two main modules:

1. the *FirstSolution* module, a program implemented by means of the `clpfd` SICStus Prolog package [4] which models the NR problem. The module let the user specify a problem instance and it starts processing it as soon as a feasible solution is found. If a feasible solution does not exists this module raises an error and stops the execution or, whenever possible, it relaxes some parts of the model leading to an approximate solution.
2. the *LocalSearch* module, which implements a set of local search algorithms for the NR problem. This module has been developed using the JEASYLOCAL framework, a Java version of the C++ framework EASYLOCAL++ [5]. The module takes as input a feasible solution obtained by the FirstSolution module and improves it by means of a local search algorithm that can be chosen by the user. The final solution found by this module can be further improved applying a different local search algorithm in an iterative process. The local search techniques implemented in this module are hill climbing, steepest descent and tabu search. Moreover, this module features a local search solver that uses the steepest descent technique for driving a constraint programming formulation of the exploration of the neighborhood.

Local Search. Among other entities (i.e., the definition of the search space and the cost function that in this case are borrowed from the constraint programming formulation), to specify a local search algorithm it is necessary to define the *move*; that is, the local perturbation to be applied to a solution in order to obtain a neighboring one. To this aim we define the following move, called *exchange*:

“Given a specific day and working time, exchange the shifts of two doctors”

Dr.	1	2	...
Jones	...	DH	...
...
Freud		UM	...
...

 \Rightarrow

Dr.	1	2	...
Jones	...	UM	...
...
Freud		DH	...
...

Fig. 1. An example of an exchange move

For example, if Dr. Freud covers the shift UM in the morning of day 2 and Dr. Jones covers the shift DH (Day Hospital) in the morning of the same day, a possible exchange move consists in swapping the shifts UM and DH between the two doctors, so that in the morning of day 2 Dr. Jones will be assigned to the shift UM and Dr. Freud will cover the shift DH as shown in Figure 1.

The shifts involved in the exchange move can be working shifts or rest periods. When we exchange a working shift with a rest one, the doctor currently in rest will get the working shift and the doctor currently working will get a rest shift. Exchange moves that involve doctors that are both currently in a rest period do not affect the solution and therefore are *idle* moves; conversely, all other types of exchange moves are meaningful and modify the solution. An exchange move is therefore identified by: the two doctors participating in the exchange, the day of the time horizon and the working time (that can be “Morning” or “Afternoon”).

Notice that, given a solution, the size of the neighborhood (i.e., the number of neighboring solutions of the current state) is equal to $\frac{m(m-1)}{2}2n = O(m^2n)$. It is possible to generalize the concept of exchange move by introducing the *compound* exchange move defined as follows:

“A compound exchange move is a sequence of one or more exchange moves”

The compound moves are very useful to handle consecutive shifts that must (or it is preferable to) be moved together. It is worth to notice that the definition of the exchange moves always lead to states where the covering constraints are satisfied: indeed, if solution *A* satisfy the covering constraints so will solution *B* obtained by an exchange move since no shift is added or removed from a day column, but simply the assignment of two doctors are swapped. As a consequence, making sure that the local search procedure will start from a feasible solution (like the one obtained by the FirstSolution module) and applying only exchange moves there is no need to make the covering constraint explicit. However, we observe that an exchange move could lead to a state where other types of constraints are violated: for example a doctor could be assigned a shift for which he/she is not qualified, or he/she could be not available on that day. These violations are taken into account by an objective function *FObj* which penalizes such situations.

In the following sections we briefly outline the local search algorithms we have developed.

Hill Climbing. The hill climbing (HC) strategy adopted in this work is the so-called randomized hill climbing: an exchange move is randomly drawn and

applied to the current solution. If the solution obtained improves or has an equal value of the objective function, then it is accepted and it becomes the new incumbent solution; conversely, if the new solution worsens the cost function it is discarded and a new random move is drawn. This procedure is iterated and the whole process stops when a user specified time-out has expired.

The crucial aspect in the implementation of this method concerns the random generation of the moves, which could lead to the generation and testing of a lot of idle moves. The random procedure has been therefore biased toward meaningful moves by enforcing that the first doctor of the exchange move must not be on a rest period, thus avoiding the generation of idle moves.

Steepest Descent. The steepest descent (SD) strategy consists in the full exploration of the neighborhood of the current solution, looking at the solution that gives the biggest improvement of the objective function. This move is then applied to the current state to obtain the new incumbent solution. The procedure is iterated and it stops as soon as no improving move can be found.

Compared to HC, this procedure is more time-consuming but it generally leads toward bigger improvements of the objective function. The key aspect of this method is the procedure employed for the enumeration of the moves. Since the evaluation of the objective function is a costly operation it is advisable to avoid unnecessary computations, especially on moves that lead to states where the constraints are violated. The enumeration procedure we have implemented makes use of a basic knowledge about the constraints and it skips such moves, allowing us to save computation time.

Tabu Search. This method (TS) explores a subset of the neighborhood of the current solution and applies the move that gives the minimum value of the objective function, regardless the fact that this value is better or worse than the one of the previous solution. This allows the method to escape from local minima, but at the risk of cycling among a set of solution. To avoid the latter phenomenon the method employs the so-called “Tabu List”, a memory of recently applied moves, and it forbids the application of moves that are inverses of the moves in the list (which would lead to an already visited state).

Among different variants of the memory mechanism presented in the literature we employ the so-called dynamic tabu list. The list contains a number of moves comprised between two values k_{min} e k_{max} , which are parameters of the method. Once a move enters the list it is assigned a random integer value in the range $k_{min}..k_{max}$ that corresponds to the number of iterations the move is kept in the tabu list. For this problem we find out experimentally that the best setting of these parameters is $k_{min} = 5$ and $k_{max} = 10$.

As for the aspiration criterion, which overrides the prohibition status of the moves, we choose to accept also moves in the tabu list when they lead to a state that is better than the current best solution.

Hybrid Steepest Descent. Finally, we implemented an hybrid local search algorithm driven by the steepest descent strategy (HSD), which employs a neighborhood model encoded in SICStus Prolog. The idea behind this algorithm is

to explore fragments of the neighborhood of the current solution by letting the constraint programming solver to find a representative solution (the best neighbor) of the current fragment. The neighborhood fragments have to be chosen so that they form a partition of the whole neighborhood of the current solution. The algorithm then accepts the best among the representative solutions that improve the objective function, inspired by the steepest descent strategy. The search of the best representative solution is performed by the `labeling` predicate of SICStus Prolog.

The concept of “neighborhood fragment” we had taken into account is based on a single day: given a solution S and a day of the temporal horizon x , the neighborhood of S w.r.t. the day x consists in the set of all solutions S' that are identical to S for all days $d \neq x$ and they differ from S for the shifts of day x . Hence, every solution will have n possible neighborhood fragments. The exploration of a neighborhood fragment w.r.t. the day x means the evaluation of all possible permutations of shifts on that day: the best permutation will be the representative of that neighborhood fragment.

This approach shares some similarities with the work of Pesant and Gendreau [9], however it differs in the type of move employed and in the granularity of the neighborhood exploration. In our case, indeed, we explore a full set of exchange moves (i.e., duty exchanges between doctors) whereas in [9] the authors use a insertion move (i.e., a duty is assigned to a doctor and not removed elsewhere). Furthermore, our neighborhood model involves a portion of shifts that insist on a single day only, while [9] neighborhood is restricted to a single shift.

4 Experiments

To the aim of comparing the four hybrid algorithms described in the previous section, we carried out an experimental evaluation of the solvers. Three types of experiments have been performed: (i) on randomly built structured instances of variable size; (ii) on real-world instances in long-runs; (iii) the best solver is compared with self-scheduling solutions.

First Test — Methodology. The goal of the first test is to analyze the behavior of the algorithms on sets of structured instances that are similar to real-world ones. We randomly generate 4 series of 10 instances whose temporal horizon n consists of 10, 20, 30 and 40 days, respectively.

All the instances have been solved by the four algorithms, accounting for a total of 160 runs. Each algorithm was granted a running time proportional to the instance size, namely of $80 \cdot n$ seconds. During each run, we record the values of the objective function that corresponds to improvements of the current solution together with the running time spent. These data have been aggregated in order to analyze the average behavior of the different algorithms on each group of 10 instances. To this aim we perform a discretization of the data on regular time intervals; subsequently, for each discrete interval we compute the average value of the objective function on the 10 instances.

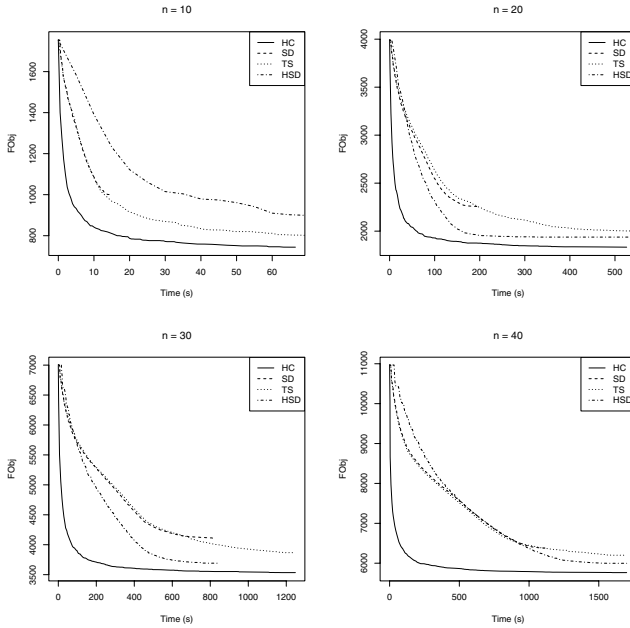


Fig. 2. Average evolution of the objective function on instances with temporal horizon n equals to 10, 20, 30 and 40 days

First Test – Results. Figure 2 reports the evolution of the objective function for the different methods on the instances with temporal horizon n equals to 10, 20, 30 and 40 days.

From the picture it is apparent that HC outperforms all the other methods on all groups of instances. Indeed, thanks to the non-exhaustive sampling of the neighborhood this method is able to find reasonably good improvements quickly, leading to a fast decrease of the objective function from the very beginning of the search process. Furthermore, HC does not get stuck in local minima (as other methods do) but it keeps perturbing the current solution with sideways moves that could possibly lead to explore new regions of the search space.

The classical SD strategy, instead, is the worst method among the ones tested. Because of the full exploration of the neighborhood, the method is slower than the hill climbing. Furthermore, the thoroughness of the exploration at each search step is not rewarded by a substantial decrease of the objective function. Finally, the method shows the intrinsic shortcoming of getting stuck in local minima and the search stagnates as soon as the first local minimum is found.

The behavior of TS lies between the two previous methods: in early stages of the search the method behaves exactly as the steepest descent (indeed, initially in the graph the two lines overlap), while the prohibition mechanisms start playing its role in diversifying the solution as soon as the first local minimum is found. Unfortunately, due to the high computational cost of the neighborhood

exploration (especially on mid- and big-sized instances) the method performs worse than hill climbing.

HSD deserves a more thorough analysis since its behavior varies on the basis of the size of the instances. On smaller instances (10 days) its behavior is the worst in terms of decrease speed of the objective function. This can be explained by the high computational overhead needed to setting up the exploration of the neighborhood fragments. In fact, for each fragment a new constraint model should be posted to the constraint store giving rise to a significant computation effort. This overhead, on smaller instances, is not rewarded by a high decrease of the objective function at each step of the search. Conversely, on the mid-sized instances (20 and 30 days) the increased computational effort is repaid by a greater decrease of the objective function and this method result more competitive than tabu search just after the first tens of seconds. However, the method does not scale well on big-sized instances (40 days): although the tendency of having a better behavior than tabu search is confirmed, this behavior is apparent only after some hundred of seconds.

Second Test — Methodology. The aim of this experiment is to analyze the behavior of the four methods on two real-world instances in a deployment situation, i.e., the methods are granted a running time of 12 hours in order to evaluate more precisely their behavior on longer runs. Both the instances have a temporal horizon of 30 days. The recorded data are the same of the previous experiment. However, differently from the previous test we need not to process the data since the results are relative to singular instances.

Second Test — Results. The behavior of the objective function on the two instances is comparable to the outcomes of the previous test on the instances of size 30, therefore we do not show it here for brevity. Here we report in Table 2a the values of the objective function reached by the four methods after 12 hours.

In the last line we report also the best value reached after 12 hours by the exhaustive search performed by the constraint solver employing the constraint programming model alone. Those values are about two times higher than the results of the hybrid methods and they fully justify the employment of the hybrid approaches for this problem. For both instances the hill climbing method is able to find the best result and it is not outperformed by any other method. However, it is worth to notice that, when granted with sufficient time, tabu search shows a good behavior and its result is not that far from the one of hill climbing.

Finally, in Table 2b we report the time needed to reach an approximation within a given percentage of the best solution value found by the methods after 12 hours.

From the table it is possible to notice that, for hill climbing and tabu search on instance 1, the convergence to good values (2% from the best value known) is obtained in less than 25 minutes. However, only HC could reach values close to the best known for instance 2. The reasons of this modest performance of tabu search will be matter of further investigation.

Table 1. Results in 12 hours of computation

Method	FObj	
	Instance 1	Instance 2
HC	3500	3240
SD	4120	3760
TS	3520	3350
HSD	3630	3460
CLP(FD)	7590	5990

Method	Instance 1			Instance 2		
	+5%	+2%	+1%	+5%	+2%	+1%
HC	111	1381	4815	329	752	2758
SD	—	—	—	—	—	—
TS	983	1343	1471	1590	—	—
HSD	664	—	—	—	—	—

(a) Final results of the four methods and of the constraint solver

(b) Time (in seconds) to reach an approximation within $x\%$ of the best value found

Table 2. Percentage of improvement of HC with respect to self-scheduling

Component	Sep	Oct	Nov	Dec
Wk	0%	0%	0%	29%
Nt	-22%	6%	0%	12%
Dp	56%	5%	30%	29%
Gu	41%	6%	22%	22%
Cn	-143%	31%	50%	-17%
FObj	15%	6%	12%	20%

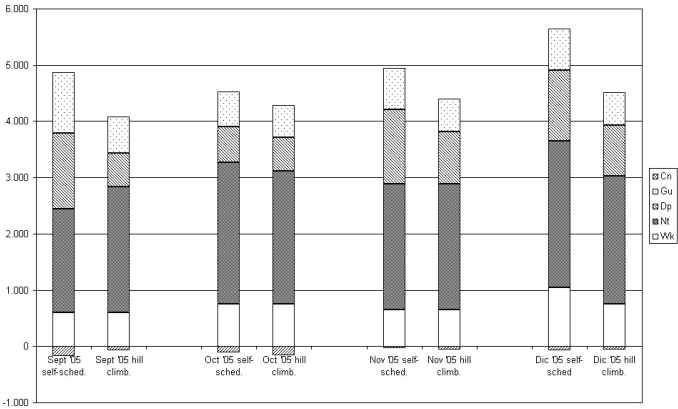


Fig. 3. Comparison of HC solver runs of 5 minutes with self-scheduling

Third Test — Methodology. From the previous tests, hill climbing results the best method (especially when provided with a short time limit), so we decided to compare the solutions obtained from this method with solutions manually obtained by the doctor who is in charge of self-scheduling. The available data ranges from the monthly requirements of September to December 2005. We compare the values obtained for the various components of the objective functions.

Third Test — Results. For the four months considered in the experiment we show in Figure 3 the outcomes of the self-schedule (the left-hand column of each pair) and the result of a short hill climbing run (5 minute of CPU time). In Table 2 we report the the percentage of improvement over the self-schedule. The data is disaggregated for the various components of the objective function and is 1 minus the ratio of the cost value found by hill climbing over the cost found by the human (so that positive values indicate improvements).

You can notice that, even though the self-schedules are really high-quality ones, the hill climbing method is able to achieve some further improvements, especially on critical components like Guards and Undesired Pairs. For Consecutive Shifts (which is a negative component, since they are preferred), instead, we can improve the values only in two cases out of the four instances.

5 Future Work and Conclusions

This work is still ongoing and the presented results are still preliminary. We wish to extend the research pursued in this paper along the following two lines:

1. integrating the tool with new local search methods, and
2. improving the resolution technique implemented.

We have experimentally verified that HC outperforms other tested methods: it employs less time than the others to find a good move. We plan to test other local search methods, such as Tabu Search with First Improvement or Elite Strategy that, visiting a small part of the neighbors, should be comparable with HC w.r.t. the time for finding a good move. We also plan to implement a Simulated Annealing method. However, we believe that a long stage for tuning parameters is needed in order to effectively use this method.

Moreover, we would like to develop a constraint solver on finite domains ad hoc for this problem in order to speed-up this stage, to ease the integration with JEasylocal, and to be independent from commercial languages. From a methodological point of view, we would like to analyze more carefully the behavior of our algorithms, using statistical methods and tuning more precisely the various parameters. In particular, further insight is needed to explain the modest behavior of tabu search.

Our system is currently in use in the Neurology Dept. of the Udine University Hospital. Acceptable solutions (with 20 doctors and a temporal horizons of one month) are obtained in a couple of minutes on a (average) PC. Future works will also include the generalization of the system in order to be usable by other hospital departments.

References

1. Emile Aarts and Jan Karel Lenstra. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Chichester (UK), 1997.
2. Krzysztof R. Apt. *Principles of Constraint Programming*. Cambridge University Press, Cambridge (UK), 2003.

3. Edmund K. Burke, Patrick De Causmaecker, Greet Vanden Berghe, and Hendrik Van Landeghem. The state of the art of nurse rostering. *Journal of Scheduling*, 7(6):441–499, 2004.
4. Mats Carlsson, Greger Ottosson, and Björn Carlson. An open-ended finite domain constraint solver. In H. Glaser, Hartel P., and Kucken H., editors, *Programming Languages: Implementations, Logics, and Programming*, number 1292 in Lecture Notes in Computer Science, pages 191–206. Springer-Verlag, Berlin (Germany), 1997.
5. Luca Di Gaspero and Andrea Schaerf. EASYLOCAL++: An object-oriented framework for flexible design of local search algorithms. *Software — Practice & Experience*, 33(8):733–765, July 2003.
6. Filippo Focacci, François Laburthe, and Andrea Lodi. Local search and constraint programming. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, chapter Local Search and Constraint Programming, pages 369–403. Kluwer Academic Publishers, 2003.
7. Narendra Jussien and Olivier Lhomme. Local search with constraint propagation and conflict-based heuristic. *Artificial Intelligence*, 139(1):21–45, 2002.
8. Eric Monfroy, Frédéric Saubion, and Tony Lambert. On hybridization of local search and constraint propagation. In Bart Demoen and Vladimir Lifschitz, editors, *Proceedings of the 20th International Conference on Logic Programming (ICLP 2004)*, number 3132 in Lecture Notes in Computer Science, pages 299–313. Springer-Verlag, Berlin (Germany), 2004.
9. Gilles Pesant and Michel Gendreau. A constraint programming framework for local search methods. *Journal of Heuristics*, 5:255–279, 1999.

A Reactive Greedy Randomized Variable Neighborhood Tabu Search for the Vehicle Routing Problem with Time Windows

Panagiotis P. Repoussis*, Dimitris C. Paraskevopoulos,
Christos D. Tarantilis, and George Ioannou

Athens University of Economics & Business
Evelpidon 47A & Leukados 33, 11362, Athens, Greece
Tel.: +302108203921; Fax: +302108828078
`prepousi@aueb.gr`

Abstract. This paper presents a hybrid metaheuristic to address the vehicle routing problem with time windows (VRPTW). The VRPTW can be described as the problem of designing least cost routes from a depot to geographically dispersed customers. The routes must be designed such that each customer is visited only once by exactly one vehicle without violating capacity and time window constraints. The proposed solution method is a multi-start local search approach which combines reactively the systematic diversification mechanisms of Greedy Randomized Adaptive Search Procedures with a novel Variable Neighborhood Tabu Search hybrid metaheuristic for intensification search. Experimental results on well known benchmark instances show that the suggested method is both efficient and robust in terms of the quality of the solutions produced.

1 Introduction

The Vehicle Routing Problem (VRP) is a focal problem of distribution management within the area of service operations management and logistics [1]. This paper presents a hybrid metaheuristic methodology for solving the vehicle routing problem with time windows (VRPTW). The latter is one of the representative combinatorial optimization problems with a wide range of applications and is known to be \mathcal{NP} -hard [2]. In VRPTW, customers with known demands are serviced by a homogeneous fleet of depot-returning vehicles with limited capacity. Each customer provides a time interval during which service must take place. The VRPTW considers both the number of vehicles required and the total distance traveled by the vehicles. Therefore, a hierarchical objective function is typically followed, where the number of routes is first minimized and then, for the same number of routes, the total traveled distance is minimized.

The VRPTW due to its wide applicability and high complexity has been the subject of extensive research efforts. A variety of algorithms have been proposed, including exact methods, heuristics and metaheuristics. The survey of Cordeau

* Corresponding author, supported by GSRT contract EP-1253-01.

et al. [2] provides all necessary pointers to the research efforts in the area of solution methods developed for the VRPTW in the 1980s and 1990s. Since then, the focus of most researchers has shifted to more complicated and sophisticated metaheuristics, capable of producing high-quality solutions in reasonable computational time. The recent surveys of Bräysy et al. [3], Bräysy and Gendreau [4], Bräysy and Gendreau [5] reveal the latest algorithmic developments in the fields of evolutionary, traditional local search improvement heuristics and metaheuristic approaches, respectively.

Although, the adaptation of metaheuristics addressed effectively a variety of hard combinatorial optimization problems, it has become evident that the concentration on a sole metaheuristic is rather restrictive. Gendreau and Potvin [6] state that many well-known metaheuristics converge towards a unified framework. Within this unified view new opportunities emerge for combining the strengths and alleviating the weaknesses of different metaheuristics, which may lead to even more powerful and flexible search methods. The latter combination of different concepts or components of various metaheuristics, forms the so called hybrid metaheuristics. Given that hybrids aim at exploiting the strengths of different methods, interaction can take place either at low-level, by designing new metaheuristics that combine various algorithmic principles of the original methods, or at high-level, by developing multi-agent architectures in which the individual agents run pure methods and communicate among themselves [6].

Many recently proposed metaheuristic approaches for the VRPTW are based on some forms of hybridization. Bräysy et al. [7] presented a two-phase multi-start local search procedure. In the first phase, using a construction heuristic, several solutions were produced followed by an Injection Tree route elimination heuristic. In the second phase, solutions were improved in terms of distance traveled by a threshold accepting post-processor. Bent and van Hentenryck [8] proposed a two stage hybrid local search approach. An enhanced simulated annealing was applied to minimize the number of vehicles along with a large neighborhood search for distance traveled minimization. Le Bouthillier et al. [9] proposed a guided parallel cooperative search along with an identification pattern mechanism, which was based on a central memory structure that coordinated various individual metaheuristics. Russell and Chiang [10] proposed a scatter search solution framework combined with a reactive tabu search, an advance recovery and a set covering procedure. Finally, Ibaraki et al. [11] presented multi-start (MLS), iterative (ILS) and adaptive multi-start (AMLS) local search approaches, enhanced with acyclic and cyclic neighborhood structures.

The main contribution of this paper is the development of an efficient multi-start hybrid metaheuristic for the VRPTW. The solution framework utilizes the basic structure of Greedy Randomized Adaptive Search Procedures (GRASP)[12]. The GRASP construction phase is equipped with a new greedy function along with a route elimination heuristic. The GRASP local search phase employs a Variable Neighborhood Tabu Search (VNTS). The latter scheme exploits the systematic changes of neighborhood structures, and thus, the

neighborhood topologies as proposed by Variable Neighborhood Search (VNS) [14], to guide a Tabu Search (TS)[15] which performs a trajectory local search. Finally, a long term memory is used that controls construction randomization, provides better sampling of the solution space and allows less reliance on parameter tuning.

The remainder of the paper is organized as follows. In Section 2 the problem definition is given. Subsequently, an overview of the solution methodology is provided, while each different component is described in Section 3. Computational experiments along with a comparative performance analysis, are then depicted in Section 5. Finally, in Section 6 conclusions are drawn.

2 Problem Definition

Following the model formulation provided in [2], let a complete graph $G = (V, A)$, where $V = \{0, 1, \dots, n+1\}$ is the node set, $A = \{(i, j) : 0 \leq i, j \leq n, i \neq j\}$ is the arc set and depot is represented by nodes 0 and $n+1$. All feasible vehicle routes correspond to paths in G that start from 0 and end at $n+1$. A set K represents homogeneous vehicles with known capacity C_k , where $k = 1, 2, \dots, |K|$. Each customer i is associated with a known demand d_i , and poses a time window $[a_i, b_i]$ that models the earliest and latest time that the service of i can take place. The service of each customer must start within the associated time window, while the vehicle must stop at the customer's location for s_i time instants. In case of early arrival at the location of i , the vehicle is allowed to wait until a_i .

There is a nonnegative travel cost c_{ij}^k , a travel time t_{ij}^k and a distance h_{ij}^k associated with each arc (i, j) of set A , with respect to the vehicle $k \in K$. Furthermore, a cost z_k is relevant to the activation of a vehicle $k \in K$. The total number of customers is $n = |V| - 2$. Indices i, j and u refer to customers and take values between 1 and n . A time window is also associated with nodes 0 and $n+1$, i.e., $[a_0, b_0] = [a_{n+1}, b_{n+1}] = [E, L]$, where E and L represent the earliest possible departure from the depot and the latest possible arrival. Feasible solutions exist only if $a_0 = E \leq \min_{i \in V \setminus \{0\}} b_i - t_{0i}$ and $b_{n+1} = L \geq \min_{i \in V \setminus \{0\}} a_i + s_i + t_{i0}$.

Let flow binary variables x_{ij}^k model the sequence in which vehicles visit customers (x_{ij}^k equals 1 if i precedes j in the route of vehicle k , 0 otherwise). Variable w_{ik} specifies the arrival time at i when serviced by vehicle k . Furthermore, each route must satisfy capacity and time window constraints. Time window state that $a_i \sum_{j \in \Delta^+(i)} x_{ij}^k \leq w_{ik} \leq b_i \sum_{j \in \Delta^+(i)} x_{ij}^k$ for all $k \in K$ and $i \in V \setminus \{0, n+1\}$, where $\Delta^+(i)$ denote the set of nodes j such that arc $(i, j) \in A$. Finally, capacity constraints state that $\sum_{i \in V \setminus \{0, n+1\}} d_i \sum_{j \in \Delta^+(i)} x_{ij}^k \leq C_k$ for all vehicles $k \in K$. The objective of VRPTW is a) to minimize the fleet size and b) total distance traveled. This combined objective reflects the trade off between fixed vehicle activation and variable transportation costs. However, it is assumed that the cost of additional vehicles always outweigh any variable transportation costs that could be saved by their use. Given the above-defined variables, the objective function can be formulated as: $\min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij}^k x_{ij}^k + \sum_{k \in K} z_k \sum_{j \in N} x_{0j}^k$.

3 Solution Methodology

A hybrid multi-start metaheuristic approach is proposed for solving the VRPTW. The main effort is focused on delivering a robust methodology with a fairly simple algorithmic structure invoking the least possible number of parameters. More specifically, the suggested method utilizes the GRASP iterative solution framework, which combines greedy heuristics, randomization and local search [12]. The GRASP construction phase builds a solution, whose neighborhood is investigated during an improvement phase, until a local minimum is reached with respect to an evaluation function. The best found solution is returned upon termination. Given that greedy randomized constructions inject a degree of diversification to the search process, the GRASP improvement phase may consist of an effective procedure fine-tuned for intensification local search. Particularly, a novel hybrid metaheuristic approach which combines VNS with TS is applied for distance traveled minimization. Finally, a long term memory structure which links each independent restart is introduced for the better sampling of the solution space and the less reliance on parameter tuning. Recent developments propose such GRASP hybrids where construction is followed by sophisticated local search, post optimization procedures and memory structures [6].

3.1 GRASP Constructive Mechanism

The GRASP constructive mechanism is characterized by a dynamic constructive heuristic and randomization. Initially, a solution is constructed iteratively by adding a new element to the partial incomplete solution. All elements are ordered in a list, called restricted candidate list (*RCL*) composed of the λ high quality elements, with respect to an adaptive greedy function. The probabilistic component is characterized by randomly choosing one of the best from the list, but not necessary the top of the list. Thus, the length of the candidate list determines and controls the extent of randomization and greediness of construction.

The proposed GRASP construction mechanism utilizes a parallel insertion solution construction scheme along with a penalty based greedy function which combines in a weighted fashion a set of criteria. However, a determinant factor which significantly affects the quality of the solutions produced, is the value of the weight parameters associated with the greedy function, since different settings dominate solution quality and fine tuning is prerequisite. In general terms, although these parameters are interrelated and heavily depend on the problem's characteristics, several authors highlight that it is impossible either to determine a robust value for each one of them or to find an evident correlation among them, which would provide good results for all test problems [13].

In literature, most metaheuristics of multi-start nature work on a set of pre-determined initial solutions. Obviously, the inherent complexity of local search suggest that it may be preferable to construct a number of feasible solutions and then apply local search only to the most promising. Therefore, the manner usually followed suggests initially the identification of specific parameters value

ranges and subsequently varying input values in small increment units within these ranges, in order to determine the parameter values that produce the best possible output [7]. In our implementation, instead of building a set of initial solutions, a set of well performing parameter settings is first determined, while only the best performing parameter settings are used at each GRASP iteration.

Finally, an attempt is made to reduce the number of routes. The idea of using separate strategies for minimizing the number of routes in addition to distance traveled is followed by most recent applications [8,7,13] for the VRPTW. Thus within the proposed solution methodology, prior GRASP improvement phase the route elimination procedure suggested in [13] is applied. The latter is based on the Ejection Chains (EC) heuristic enhanced with an intelligent reordering mechanism, called IR-insert. The basic idea is to combine series of moves into a compound move with the goal to make room for relocation by first removing or reordering other customers from the same route.

Greedy Function. The proposed greedy function adapts the penalty measurements and the parallel construction framework introduced in [17], although, enhanced with additional mechanisms and customer selection criteria. According to the sequential insertion framework of Solomon [18], a feasible solution is constructed by inserting a non routed customer into a current partial constructed route at each iteration. In the context of the parallel construction schemes, Solomon’s sequential insertion is applied considering several routes simultaneously. Particularly, after initializing a set of r routes a customer is iteratively assigned between two adjusted customers in a current partial route. If at some iteration an unassigned customer cannot be inserted in any of the existing set of routes, an unassigned “seed” customer is identified and a new route is initialized. The overall procedure is repeated until all customers are assigned to vehicles.

A point of prime importance, is the selection and assignment of “seed” customers to create both the initial set of vehicles and the additional “surplus” vehicles, if required. What complicates further matters is the strong relationship between customers time availability for service, introduced by time windows that dictate the sequence in which customers are serviced by vehicles. On the other hand, the more distant are the customers from the depot, the more difficult is their assignment. In order to capture both these trends a two phase “seed” customer selection scheme is followed. For the initial set of routes, “seed” customers are determined such that the most geographically dispersed or most time constrained are considered first, as proposed in [17]. In later phases of construction, if initialization of additional vehicles is needed random selection is followed.

Let $\pi_{ij,u}$ denote the insertion cost of an unassigned customer u when is inserted between i and j in a partial solution Ω . For every feasible insertion position of u into a route ρ , the minimum insertion cost $\pi_{\rho,u} = \min_{i,j \in \rho} \pi_{ij,u}$ is found. Similarly, the overall minimum insertion cost $\pi_{\rho^*,u}$ corresponds to the $\min_{\rho \in \Omega} \pi_{\rho,u}$ and denotes the best feasible insertion position at route ρ^* of u . Subsequently, a penalty cost, Π_u is calculated for every unassigned customer. This penalty can be viewed as a measure of the cost that would have to be paid later if the corresponding customer is not assigned to its current best position.

$$\Pi_u = \sum_{\rho \in \Omega} (\pi_{\rho,u} - \pi_{\rho^*,u}) \quad (1)$$

Large Π_u values indicate that u should be considered first to avoid paying later a relatively high cost. Contrary, customers with small penalty values can wait for insertion. Thus, customers that cannot be feasibly inserted into a route the insertion cost must be set to a large value lv , in order to force Π_u to large values as well. In [17] lv was set to infinity. Alternatively, we propose an intuitively intelligent approach that adaptively tunes values of lv . In particular, lv is set equal to difference between the overall maximum and minimum insertion cost of all unassigned customers u for the existing set of routes. Thereafter, whenever a customer u cannot be feasibly insert into a route ρ the current penalty Π_u is incremented by $\max_{u \in V} \{\max_{\rho \in \Omega} \{\pi_{\rho,u}\}\} - \min_{u \in V} \{\min_{\rho \in \Omega} \{\pi_{\rho,u}\}\}$. Finally, cost $\pi_{ij,u}$, is defined as a weighted combined result from several sub-metrics.

$$\pi_{ij,u} = \vartheta_1 \pi_{ij,u}^1 + \vartheta_2 \pi_{ij,u}^2 + \vartheta_3 (\pi_{ij,u}^{3s} + \pi_{ij,k}^{3g}) \quad (2)$$

where ϑ_1, ϑ_2 and ϑ_3 are nonnegative weights such that $\vartheta_1 + \vartheta_2 + \vartheta_3 = 1$. Component $\pi_{ij,u}^1$ measures the distance increase caused by insertion of u [18].

$$\pi_{ij,u}^1 = t_{iu} + t_{uj} - t_{ij} \quad (3)$$

Component $\pi_{ij,u}^2$ measures vehicle utilization in terms of total waiting time prior and after the insertion of u [13].

$$\pi_{ij,u}^2 = \sum_{i \in \rho \cup \{u\}} (a_i - w_{ik})^+ - \sum_{i \in \rho \cap \{u\}} (a_i - w_{ik})^+ \quad (4)$$

Finally, the third component combines two metrics $\pi_{ij,u}^{3s}$ and $\pi_{ij,u}^{3g}$. The first metric refers to the closeness of the earliest time that service can take place a_u , compare to vehicle's k arrival time w_{uk} at u [16].

$$\pi_{ij,u}^{3s} = w_{uk} - a_u \quad (5)$$

The second metric express the compatibility of the time window of the selected customer u with the specific insertion position in the current route [16].

$$\pi_{ij,u}^{3g} = b_u - (w_{ik} + s_i + t_{iu}) \quad (6)$$

3.2 GRASP Local Search

The GRASP improvement phase consists of a Variable Neighborhood Tabu Search. VNS is based upon a basic principle: systematic change of neighborhoods during the search. Given a set of pre-selected neighborhood structures, a solution is randomly generated in the first neighborhood of the current solution, from which a local descent is performed. If the local optimum obtained is not

better than the incumbent, the procedure is repeated using the next neighborhood. The search restarts from the first neighborhood whenever a solution is better than the incumbent or every neighborhood has been explored.

Although, VNS framework treats a single solution at each iteration, the trajectory followed during the search of the solution space is discontinuous. Indeed, it explores increasingly distant neighborhoods of a current solution, and moves at random from one solution to another (shaking). Thus, favorable characteristics of a solution, are often kept and used to obtain promising neighboring solutions. Although the basic VNS scheme is a descent improvement method with randomization, it could be transformed into a descent-ascent method. However, the basic VNS some times meets difficulties to escape from the local optima. Contrary, TS has no such difficulties, since the current solution is allowed to deteriorate, while the recency-based memories prevent cycling allowing to overcome local optima.

In literature, two ways of making hybrids of VNS and TS appear; the use of TS within VNS and the opposite. In the first case the local descent of VNS is replaced by TS while in the second case, different neighborhoods are exploited by TS. The implementation proposed herein uses TS internally within VNS performing the local search for given neighborhood structures, while externally VNS performs systematical neighborhood changes and controls the shaking mechanism. Using this rationale, it is reasonable to expect a thorough and systematic exploration of the solution space by utilizing trajectory local search and neighborhood topologies. A recent compilation of such hybrids can be found in [14].

Variable Neighborhood Tabu Search. The VNS iteration framework consists of three phases: shaking, local search and move. At the initialization step, a set of neighborhoods is selected. In the *shaking* phase a solution \hat{s} in the y th neighborhood of the current solution s is randomly selected. *Local search* is then applied in order to find the best neighbor \tilde{s} of \hat{s} . If $f(\tilde{s}) < f(\hat{s})$ \hat{s} is replaced by \tilde{s} . The latter is the so called *move* phase. The overall scheme is repeated from a new shaking. Otherwise, y is incremented and a new shaking phase starts using a different neighborhood structure, until some termination conditions are met.

On the other hand, TS explores the solution space by moving at each iteration from a solution s to the best solution of the neighborhood $N_y(s)$. To avoid cycling, solutions possessing some attributes of recently explored ones are temporarily declared as *tabu* (short term memory). Tabu moves are represented by attributes which are stored in an ordered queue called *tabu list*. The best admissible move is chosen as the highest evaluation move in the neighborhood of the current solution in terms of objective function and tabu restrictions. Obviously, the tabu list is imposed to restrict the search from revisiting solutions that were considered previously and to discourage the search from cycling between subsets of solutions. At each iteration the best solution of the reduced $N_y(s)$ is chosen as the new current solution and subsequently added to the tabu list. The duration that an attribute remains tabu is called *tabu tenure* and tabu status can be overridden if certain conditions are met. The latter is called aspiration criterion and occurs when a tabu solution is better than any previously met solution. The overall procedure iterates until a termination criterion is met.

In this paper, the proposed VNTS can be illustrated as follows. At the initialization step, a set of neighborhood structures with increasing cardinality ($|N_1| < |N_2| \dots |N_{y_{max}}|$) is defined. Given an initial solution \acute{s} (from GRASP constructive mechanism), the neighborhood index y is initialized. Subsequently, Tabu Search begins considering neighborhood y . Initially, tabu lists of y are realized and iteratively the best admissible neighbor solution of $N_y(\acute{s})$ is depicted. The latter procedure iterates for *maxCount* times without observing any further improvement. At the end the new local optima solution \ddot{s} obtained is returned. If $f(\ddot{s}) < f(s)$, s is replaced by \ddot{s} and a new shaking phase is performed with $y=1$. In the shaking phase, a solution \acute{s} in the y th neighborhood of the current solution s is randomly selected. Otherwise, y is incremented and a new shaking phase starts using a different neighborhood structure. The oscillations between shaking and local search are repeated until all possible neighborhood structures are examined, i.e. $y=y_{max}$, and no further improvement is observed $f(\ddot{s}) > f(s)$.

Variable Neighborhood Tabu Search

Select a set of neighborhood structures N_y , $y=1, 2, \dots, y_{max}$

$y \leftarrow 1, \acute{s} \leftarrow s, \ddot{s} \leftarrow s$

While ($y \leq y_{max}$) **do**

InitializeTabulist(y) of *tabuTenure* size

AspirationConditions(\acute{s}), $counter \leftarrow 0$

While $counter \leq maxCount$ **do**

Find $\acute{s} \in N(\acute{s})$ | \acute{s} subject to tabu conditions & aspiration criteria

AllowedSet(\acute{s}) $\leftarrow \acute{s}$

$\acute{s} \leftarrow ChooseBestOf(AllowedSet(\acute{s}))$

UpdateTabulist(y)

If ($f(\acute{s}) < f(\ddot{s})$) **then**

$\ddot{s} \leftarrow \acute{s}$, *AspirationConditions*(\ddot{s}), $counter \leftarrow 0$

Else $counter \leftarrow counter + 1$

Endwhile

If ($f(\ddot{s}) < f(s)$) **then** “Move phase”

$s \leftarrow \ddot{s}$, $y \leftarrow 1$, $\acute{s} \leftarrow PickAtRandom(N_y(s))$ “Shaking phase”

Else

$y \leftarrow y + 1$, $\acute{s} \leftarrow PickAtRandom(N_y(s))$ “Shaking phase”

EndIf

EndWhile

The above framework introduces two user defined parameters, *maxCount* and *tabuTenure*. In subsequent section, sensitivity analysis on both these parameters is given. Finally, an important factor is the neighborhood change scheme. In literature strategies are proposed including sequential and nested changes. In this paper, a sequential selection is applied based on cardinality, which implies moving from relatively poor to richer neighborhood structures. The latter scheme significantly increases the possibilities of finding higher quality solutions. The proposed sequence is the defined as follows GENI, Or-Opt, Cross, 2-Opt, 0-1 Relocate, 1-1 Interchange, on both single route and pair of routes.

3.3 Long Term Memory Structure

One shortcoming of GRASP schemes comes from the fact that each restart is independent of the previous ones, thus preventing the exploitation of previously obtained solutions to guide the search. Furthermore, the use of a randomized greedy heuristic to generate starting solutions is attractive only if the greedy solutions are different enough to allow a good sampling of the solution space. For this reason, a long term memory structure is introduced as an extension to the basic memoryless GRASP scheme. A similar probabilistic learning mechanism has been proposed also in [20], called Reactive GRASP, in which the size of RCL varied according to the quality of solutions obtained at each iteration. In our implementation we extend the aforementioned reactive tuning of the size of RCL considering both quality and diversification measures among solutions produced during construction phase. The latter approach provides efficient sampling of solution space since diversified regions of good quality solutions are identified and limited search in regions of the search space already explored is avoided.

As mentioned above, the repetitive sampling mechanism of GRASP is controlled by RCL size, which in our implementation RCL is cardinality based. Indeed, it is made up of λ elements with the largest penalty costs. Thus, cases where λ equals to 1 corresponds to pure greedy construction, while $\lambda \gg 1$ is equivalent to random construction. Let \dot{s} and \ddot{s} denote two solutions produced by sequent iterations of GRASP construction phase. In order to measure quantitatively their diversity the so-called similarity $D_s^{\ddot{s}}$ is used. The latter is defined as the number of common arcs between \dot{s} and \ddot{s} ,

$$D_s^{\ddot{s}} = \sum_{(i,j) \in A} \xi_{ij}, \quad (7)$$

where binary variable ξ_{ij} is equal to 1 if (i, j) is an arc of both \dot{s} and \ddot{s} , 0 otherwise. The larger the λ the smaller is the distance $D_s^{\ddot{s}}$, and thus the better the sampling of the solution space obtained. On the other hand, while λ tends to increase from 1, the worse is the quality of solutions produced, and thus, the more is the computational effort needed by local search phase to improve the incumbent solution's quality. Thus, the appropriate choice of λ is crucial.

Let a non fixed size λ take values at each iteration from a discrete set such that $A = \{\lambda_1, \lambda_2, \dots, \lambda_\nu\}$. The probabilities associated with the choice of each value are initially set equal to $B_\tau = 1/|A|$ where $\tau = 1, 2, \dots, \nu$. Moreover, let \dot{s} and \ddot{s} be two sequent solutions, A_τ denote the average objective function values and D_τ the average similarity of all solutions found so far using $\lambda = \lambda_\tau$. All probabilities are reevaluated once by taking $B_\tau = \beta_\tau / \sum_{v=1}^{\nu} \beta_v$, where $\beta_\tau = A_\tau / f(\ddot{s}) + D_\tau / D_s^{\ddot{s}}$ for $\tau = 1, 2, \dots, \nu$. The first component, A_τ / \ddot{s} , express the ratio between the overall average values found so far, A_τ , and the value of the current solution found \ddot{s} using λ_τ . Obviously, the $A_\tau / f(\ddot{s})$ will increase when better on average solution values are found. Similarly, the larger the distance between \dot{s} and \ddot{s} against the average similarity D_τ , the smaller their ratio.

Therefore, the value of β_τ will be larger for values of λ_τ leading to the best valued and most diversified solutions on average. Larger values of β_τ correspond

to more suitable values of λ . Thus, the probabilities of these more appropriate values will then increase when they are reevaluated. The above reactive approach of reducing progressively the set A , improves significantly robustness and solution quality, due to greater diversification and less reliance on tuning.

Below, the overall Reactive Greedy Randomized Variable Neighborhood Tabu Search solution framework, is illustrated. The termination condition used bounded the allowed computational time consumption to an upper limit ζ .

Reactive GRASP-VNTS

```

 $A \leftarrow \text{InitializeSet}(|N|/2)$ ,  $index \leftarrow 1$ 
For all elements  $\lambda_i \in A$  Do Initialize ( $D_i, A_i, B_i$ )
While CPU time consumed  $\leq \zeta$  Do  $\lambda \leftarrow \lambda_{index}$ ,  $s \leftarrow \emptyset$ 
  While solution not complete Do
     $RCL_{\lambda_{index}} \leftarrow \text{Build Restricted Candidate List}(s)$ 
     $x \leftarrow \text{Select Element At Random}(RCL_{\lambda_{index}})$ 
     $s \leftarrow s \cup \{x\}$ 
    Update Greedy Function( $s$ )
  End while
  Route Elimination Procedure( $s$ )
  VNTS( $s$ )
  UpdateBestSolution( $s, elite$ ), Reevaluate( $D_{index}, A_{index}, B_{index}$ )
  If  $index = |A|$  AND  $|A| > 1$  Do
    Remove  $\lambda_i$  with the smallest  $B_i$  from set  $A$ ,  $index \leftarrow 1$ 
  Else  $index \leftarrow index + 1$ 
Endwhile

```

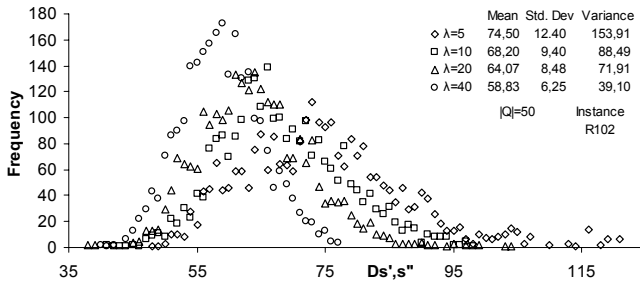
4 Computational Results

4.1 Data Sets and Parameter Sensitivity

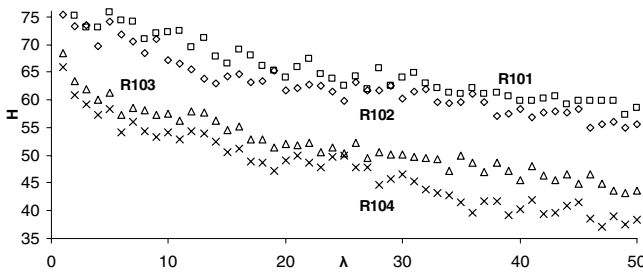
For the evaluation of the proposed methodology, various computational experiments were conducted. Computational results reported herein, are obtained using Solomon's 100-node benchmark data sets R, C and RC containing randomly dispersed, clustered, and semi-clustered customers respectively. Moreover, R1, C1, RC1 have short scheduling horizon contrary to R2, C2, RC2 data sets [18].

Contrary to other metaheuristic approaches, the proposed solution methodology introduced relatively few parameters, while most of them are relatively insensitive to the characteristics of the problem considered. Indeed, using only simple adjustments one can determine very well performing parameter settings with modest effort except for those concerning construction. As described previously, for the parameters associated with the greedy function, θ_1 , θ_2 and θ_3 , all values within ranges 0.2-1.0, 0.1-0.6 and 0.1-0.4 are applied in increments of 0.05 units respectively, and the best possible combination is used. On other hand, the rest parameters remained fixed for all computational experiments reported.

In terms of GRASP construction, despite the fact that λ is self-adjusted, the range within it fluctuates must be defined. The computational experiments conducted, indicated that a size between 5 and 30 is suitable for small-scale (100 customers) problems. Let a population of Q solutions produced with constant λ . The average similarity among all possible pairs of solutions is denoted as H equals to $\frac{\sum_{s' \in Q} \sum_{s'' \in Q-s'} D_{s'}^{s''}}{|Q|(|Q|-1)}$. Although, large values of λ generate solutions of relatively poor quality, based on similarity $D_{s'}^{s''}$ it is evident that relatively only large values of λ can generate adequate distant solutions. Figure 1(a) indicates the $D_{s'}^{s''}$ achieved for constant values of λ considering a population of 50 solutions. Figure 1(b) illustrates average similarity H obtained for different values of λ and different problem instances. Obviously, as moving from R101 to R104 the H obtained is smaller due to the fact that time windows are relaxed.



(a) Distribution of $D_{s'}^{s''}$ among all possible pairs



(b) H versus λ for different problem instances

Fig. 1. Similarity versus size of *RCL* list

Other critical user defined parameters are *tabuTenure* and the maximum number of iterations *maxCount*. Based on computational experiments, a relatively small value of *tabuTenure* close to 10, best fits intensification search. On the other hand, for tuning *maxCount* several experiments conducted. Figures 2(a)-2(d) illustrate distance cost (y-axes) obtained during VNTS search

versus computational time consumed in seconds (x-axes). Particularly, figures demonstrate the search progress of VNTS for different values of $maxCount$ with and without invoking the shaking mechanism. This distinction is made due to the fact that shaking mechanism sometimes deteriorate extensively solution's quality and thereafter, small values of $maxCount$ are not sufficient to ensure smooth performance of TS. Contrary, large values of $maxCount$ result in excessive computational time consumption without the analogous output. Generally, the scheme of VNTS with shaking requires a value close to 20 while without shaking an appropriate value is close to 15. In terms of shaking mechanism efficiency, it is observed that on average higher quality solutions are produced.

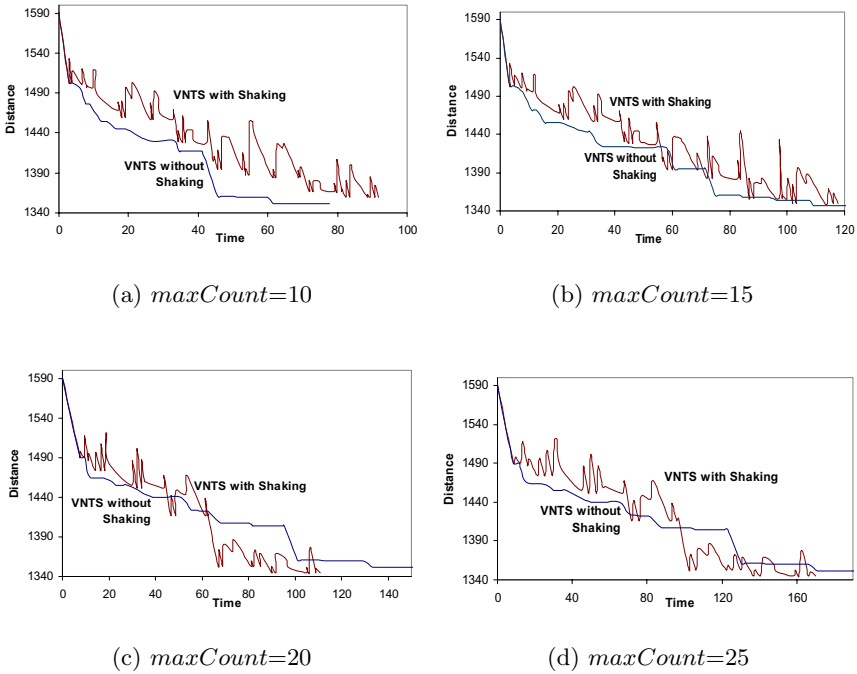


Fig. 2. Search progress on R103: VNTS for different $maxCount$

Finally, in order to demonstrate the performance of VNTS, comparisons are made against the Variable Neighborhood Descent (VND) scheme and the Tabu Search (TS) with random neighborhood selection, $maxCount$ equal to 200 and constant tabu list size equal to 20, for a given initial solutions. In particular, R103 and R104 problem instances used as the test beds of experiments. Figures 3(a)-3(b) illustrate the search progress of VND, TS, VNTS with and without shaking. In all cases observations are similar. Although the VND scheme is quite effective in terms of computational time consumption, the quality of solutions produced is relatively poor. Contrary, TS is the most time effective producing

high quality solutions in short computational times, however, it prematurely converges to local optimum solutions. Lastly, both variants of VNTS (with and without shaking) performed best in terms of final output. The fact that the computational time required is larger compared to the other methods is expected, since they invest in a more thorough exploration of the solution space.

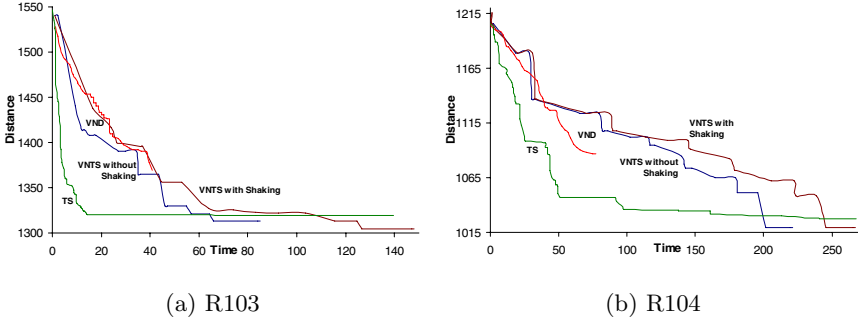


Fig. 3. Comparative performance of VND, TS and VNTS

4.2 Comparative Analysis

Table 1 compares the results obtained by the proposed hybrid metaheuristic, denoted as ReGRVNTS, with the best performing metaheuristics developed for the VRPTW with limited computational resources. Considering as the primary objective the minimization of the number of routes and secondarily the minimization of the total traveled distance, Table 1 illustrates the mean number of vehicles and the mean distance traveled for each benchmark data set of Solomon [18]. The results indicate the effectiveness and efficiency of ReGRVNTS since the worst aggregate average percentage deviation observed was 0.42% from the best known solutions. In particular, for clustered C1 and C2 sets all optimal solutions obtained while in random R1 and semi-clustered RC1 sets, competitive quality solutions produced. In terms of computational time consumption, the upper limit ζ was set equal to 2000 secs. The latter threshold is close to the relative computational time consumption of all other approaches.

Due to the special structure of the benchmark data sets considered, none method is able to produce the best known solutions for each respective data set. Only approaches of Bräysy [13], Bräysy et al. [21], Ibaraki et al. [11] and Le Bouthillier et al. [9] produce the optimum number of vehicles. In terms of ReGRVNTS only data set R1 is above optimum by one vehicle. It is also worth mentioning that the results reported are obtained with constant and fixed parameter settings compared to other methodologies. Finally, ReGRVNTS was coded in C++ and run on a 1.5 GHz Pentium IV.

Table 1. Comparison of results obtained by ReGRVNTS compare to the best performing metaheuristics proposed recently with limited computational effort. CNV stands for Cumulative Number of Vehicles and CTD stands for Cumulative Total Distance.

Set	B[13]	BBB[21]	IB[11]	HG[22]	BvH[8]	BCK[9]	BHD[7]	ReGRVNTS
R1	1222,12	1221,1	1217,36	1211,67	1203,84	1214,2	1214,69	1220,97
	11,92	11,92	11,92	12,08	12,17	11,92	12	12.00
	975,12	975,43	959,11	950,72	980,31	954,32	960,44	974,44
R2	2,73	2,73	2,73	2,82	2,73	2,73	2,73	2,73
	828,38	828,38	828,38	828,45	828,38	828,38	828,38	828,38
C1	10	10	10	10	10	10	10	10
	589,86	589,86	589,86	589,96	589,86	589,86	589,86	589,86
C2	3	3	3	3	3	3	3	3
	1389,58	1389,89	1391,03	1395,93	1379,03	1385,3	1389,2	1396,68
RC1	11,5	11,5	11,5	11,5	11,63	11,5	11,5	11,5
	1128,38	1159,37	1122,79	1135,09	1158,91	1129,43	1124,14	1160,97
RC2	3,25	3,25	3,25	3,25	3,25	3,25	3,25	3,25
CTD	57710	57952	57192	57422	57707	57360	57422	58006
CNV	405	405	405	408	409	405	406	406

5 Conclusions

This paper presented an efficient and robust multi-start solution methodology to tackle the VRPTW. The suggested method utilized the basic structure of Greedy Randomized Adaptive Search Procedures equipped with a long term memory structure for the better strategic sampling of the solution space. The GRASP local search phase employed a Variable Neighborhood Tabu Search hybrid metaheuristic for intensification search. VNTS exploited the systematic changing of neighborhood structures and the shaking mechanism offered by the basic VNS scheme, to guide a TS which performed a trajectory local search. The resulting method proved efficient in terms of solution space exploration. The latter justified on the all computational experiments conducted on well known Solomon's benchmark data sets. In terms of further research, memory structures that exploit information gathered during search, is a worth pursuing research direction.

References

1. Tarantilis, C.D.: Solving the vehicle routing problem with adaptive memory programming methodology. *Comput. Oper. Res.* **32**(2005) 2309–2327
2. Cordeau, J-F., Desautniers, G., Desrosiers, J., Solomon, M., and Soumis, F.: The Vehicle Routing Problem with Time Windows. In: Toth P. and Vigo D. (eds), *The Vehicle Routing Problem*, SIAM Publishing: Philadelphia, (2002) 157–193
3. Bräysy, O., Dullaert, W., Gendreau, M.: Evolutionary algorithms for the Vehicle Routing Problem with Time Windows. *J. Heuristics* **10**(2004) 587–611
4. Bräysy, O., Gendreau, M.: Vehicle Routing Problem with Time Windows Part I: Route construction and local search algorithms. *Trans. Sci.* **39**(2005) 104–118

5. Bräysy, O., Gendreau, M.: Vehicle Routing Problem with Time Windows Part II: Metaheuristics. *Trans. Sci.* **39**(2005) 119–139
6. Gendreau, M., Potvin, J-Y.: Metaheuristics in Combinatorial Optimization, *Anns. Opns. Res.* **140**(2005) 189–213
7. Bräysy, O., Hasle, G., Dullaert, W.: A multi start local search algorithm for the vehicle routing problem with time windows. *Eur. J. Opl. Res.* **159**(2005) 586–605
8. Bent, R., Van Hentenryck, P.: A two-stage hybrid local search for the vehicle routing problem with time windows. *Trans. Sci.* **38**(2004) 515–530
9. Le Bouthillier, A., Crainic, T.G., Kropf, P.: A Guided Cooperative Search for the Vehicle Routing Problem with Time Windows. *IEEE Intelligent Sys.* **20**(2005) 36–42
10. Russell, R.A., Chiang, W-C.: Scatter search for the vehicle routing problem with time windows. *Eur. J. Opl. Res.* **169**(2006) 606–622
11. Ibaraki, T., Imahori, S., Kudo, M., Masuda, T., Uno, T., Yagiura, M.: Effective local search algorithms for routing and scheduling problems with general time window constraints. *Trans. Sci.* **39**(2005) 206–232
12. Feo, T., Resende, M.: Greedy randomized adaptive search procedures. *J. Glb. Opt.* **6**(1995) 109–154
13. Bräysy, O.: A Reactive Variable Neighborhood Search for the Vehicle Routing Problem with Time Windows. *INFORMS J. Comp.* **15**(2003) 347–368
14. Hansen, P., Mladenović, N.: Variable neighborhood search: Principles and applications, *Eur. J. Opl. Res.* **130**(2002) 449–467
15. Glover, F.: Future paths for integer programming and links to artificial intelligence. *Comput. Oper. Res.* **13**(1986) 533–549
16. Ioannou, G., Kritikos, M., Prastacos, G.: A greedy look-ahead heuristic for the vehicle routing problem with time windows. *J. Oper. Res. Soc.* **52**(2001) 523–537
17. Kontoravdis, G., Bard, J.F.: A GRASP for the vehicle routing problem with time windows. *ORSA J. Comp.* **7**(1995) 10–23
18. Solomon, M.M.: Algorithms for the vehicle routing and scheduling problems with time window constraints. *Opns. Res.* **35**(1987) 254–265
19. Bräysy, O.: Local search and variable neighborhood Search Algorithms for the Vehicle Routing Problem with Time Windows, Technical Report, Acta Wasaensia 87, University Wasaenis, Vaasa, 2001
20. Prais, M., Rideiro, C.C.: Parameter variation in GRASP procedures. *Investigación Operativa* **9**(2000) 1–20
21. Berger, J., Barkaoui, M., Braysy, O.: A route directed hybrid genetic approach for the vehicle routing problem with time windows. *Inform. Systems Oper. Res.* **41**(2003) 179–194
22. Homberger, J., Gehring, H.: A two phase hybrid metaheuristic for the vehicle routing problem with time windows. *Eur. J. Opl. Res.* **162**(2005) 449–467.

Incorporating Inference into Evolutionary Algorithms for Max-CSP

Madalina Ionita, Cornelius Croitoru, and Mihaela Breaban

“Al.I.Cuza” University of Iasi, Romania
{mionita, croitoru, pmihaela} @info.uaic.ro

Abstract. This paper presents a simple way of combining inference with stochastic search for solving constraint satisfaction problems. The approach makes use of an evolutionary algorithm for search assisted by an inference algorithm, the variable elimination procedure. The hybrid algorithm obtained is adapted in such way that a balance between exploitation and exploration is preserved. The results are presented for the Max-CSP optimization task.

1 Introduction

There are two major ways to solve constraint satisfaction problems (CSPs) : inference approaches and search algorithms [1]. Inference approaches derive and record new information in order to make the problem easier to solve. Search algorithms seek for a solution in the space of partial instantiations. Efficient algorithms for CSPs combine search with inference for more accurate results. Moreover, since most of the real world problems that can be expressed as CSPs are intractable, approximation schemes are preferred to deterministic ones.

In the realm of evolutionary computing there have been numerous attempts to use evolutionary algorithms for CSPs solving. Because the application of operators cannot guarantee the feasibility of offspring, constraint handling is not straightforward in an evolutionary algorithm. However, special methods for dealing with constraints have been developed [2], [3], [4], [5]. In [6] a comparison of the best evolutionary algorithms is given. Among the method types used in this area, the direct and indirect constraint handling must be mentioned [2]. The direct constraint handling includes methods like repairing infeasible solution candidates, use of special representations and operators, and decoding. The indirect approach refers to including penalties into the fitness function.

It was observed that the use of some heuristics inside an evolutionary algorithm or a method for adapting the penalties can be useful for solving CSPs. For example, Eiben in [7] proposes to incorporate some heuristics into the genetic operators. The mutation operator selects a number of variables to be mutated and assigns them new values. The selected variables are those appearing in constraints that are most often violated. The new values are the ones that maximize the number of satisfied constraints. Another way of incorporating heuristic information in an evolutionary algorithm is described in [8]. The heuristics are not

incorporated into operators, but as a standalone module. Individual solutions are improved by calling a local optimization procedure for each of them and then blind genetic operators are applied.

As previously mentioned, most of the real world problems are over-constrained and do not have an exact solution. The goal became then to find a solution that satisfies the most constraints. This paper proposes a way of using the information returned by an inference algorithm inside an evolutionary algorithm to solve the Max-CPS problem, the optimization version of constraint satisfaction. The inference algorithm employed is Bucket-Elimination, an algorithm which stores new information as a new constraint. This constraint summarizes the effect of the variable that has been processed and will replace the variable. The main drawback is that the new constraints may have large arities, so it takes an exponential time and space to process and store. To limit this drawback we have used an approximated version, the Mini-bucket Elimination algorithm. In some cases the mini-bucket scheme cannot find the optimal solution, even when it uses a higher level of accuracy. This scheme has been extended in [9] with a way of automatically generating some heuristic functions that can be used in solving optimization tasks. The heuristic function returns a lower bound on the minimum number of constraints that are violated by the best extension of a partial assignment. Our approach uses these functions inside the genetic operators.

The next section gives a short introduction to the constraint satisfaction problems and the used inference algorithm. The new approach is presented in section 3. Section 4 presents the tests and results for randomly generated binary CSPs. Section 5 concludes the paper with a discussion of our work and directions for future research.

2 Constraint Satisfaction

2.1 Constraint Satisfaction Problems

Definition 1. A *Constraint Satisfaction Problem (CSP)* is defined by a set of variables $X = \{X_1, \dots, X_n\}$, associated with a set of discrete-valued domains, $D = \{D_1, \dots, D_n\}$, and a set of constraints $C = \{C_1, \dots, C_m\}$. Each constraint C_i is a pair (S_i, R_i) , where R_i is a relation $R_i \subseteq D_{S_i}$ defined on a subset of variables $S_i \subseteq X$ called the scope of C_i . The relation denotes all compatible tuples of D_{S_i} allowed by the constraint.

A solution is an assignment of values to variables $x = (x_1, \dots, x_n)$, $x_i \in D_i$, such that each constraint is satisfied. If a solution is found, then the problem is named satisfiable or consistent. The problem may ask for one solution, all solutions, or - when a solution does not exist - a partial solution that optimizes some criteria is desired. In the following, our discussion will focus on the last case, that is, the Max-CSP problem. The task consists in finding an assignment that satisfies a maximum number of constraints. For this problem the relation R_i is given as a cost function $C_i(X_{i1} = x_{i1}, \dots, X_{ik} = x_{ik}) = 0$ if $(x_{i1}, \dots, x_{ik}) \in R_i$ and 1 otherwise.

2.2 Bucket Elimination

In this section, a brief description of the variable elimination framework (also known as bucket elimination [10]) with some extensions will be given.

The Bucket Elimination algorithm (BE) takes as input an ordering of variables and the cost functions. The method partitions the functions into buckets. Each function is placed in the bucket corresponding to the variable which appears latest in the ordering. After this step, two phases take place then. In the first phase the buckets are processed from last to first. The processing consists in a variable elimination procedure that computes a new function which is placed in a lower bucket. For the Max-CSP problem this procedure will compute the sum of all constraint matrices and will minimize over the bucket's variable. In the second phase, the algorithm considers the variables in increasing order. It builds a solution by assigning a value to each variable, consulting the functions created during the first phase.

Mini-bucket Elimination (MBE) [11] is an approximation of the previous algorithm which tries to reduce space and time complexity. The buckets are partitioned into smaller subsets, called mini-buckets which are processed separately, in the same way as in BE. The number of variables from each mini-bucket is upper bounded by a parameter, i . The time and space complexity of the algorithm is $O(exp(i))$. Also this parameter controls the trade-off between the quality of the approximation and the computational complexity.

The mini-bucket algorithm is expanded in [9] with a mechanism to generate some heuristic functions. The essential idea is to use the functions recorded by the mini-bucket in a more efficient way. Given a partial assignment $x^p = (x_1, \dots, x_p)$, the number of constraints violated by the best extension of x^p is:

$$f^*(x^p) = \min_{x_{p+1}, \dots, x_n} \sum_{k=1}^n C_k$$

for the variable ordering $d = (X_1, \dots, X_n)$.

The previous sum can be computed as:

$$f^*(x^p) = \left(\sum_{C_i \in \text{buckets}(1 \dots p)} C_i \right)(x^p) + h^*(x^p)$$

where $h^*(x^p)$ can be estimated by a heuristic function $h(x^p)$, derived from the functions recorded by the MBE algorithm. $h(x^p)$ is defined as the sum of all the h_j^k functions that satisfy the following properties:

- they are generated in buckets $p + 1$ through n , and
- they reside in buckets 1 through p .

$$h(x^p) = \sum_{i=1}^p \sum_{h_j^k \in \text{buckets}_i} h_j^k, \text{ where } k > p$$

h_j^k represents the function created by processing the j -th mini-bucket in bucket_k .

These functions can be used as heuristic evaluation functions in search. In [9] two deterministic search methods, Branch-and-Bound and Best First have been proposed to take advantage of such functions. In the next section we will describe a possible way of incorporating these functions into an evolutionary algorithm.

3 Hybridization

Genetic algorithms [12] are powerful search heuristics which work with a population of chromosomes, potential solutions of the problem. The individuals evolve according to rules of selection and genetic operators. To obtain good results for a problem we have to incorporate knowledge about the problem into the evolutionary algorithm. A possible way of achieving that is by hybridizing the evolutionary algorithm with some standard procedures. Evolutionary algorithms are flexible and can be easily extended by incorporating alternative approaches. The heuristic information introduced in an evolutionary algorithm can enhance the exploitation but will reduce the exploration. A good balance between exploitation and exploration is important.

Our new method includes the constraint processing information into the evolutionary algorithm in order to improve the search results. The basic idea is to use the functions returned by the mini-bucket algorithm as heuristic evaluation functions. The selected genetic algorithm is a simple one, with a classical scheme. The special particularity is that the algorithm uses the inferred information in a genetic operator and an adaptive mechanism for escaping from the local minima.

A candidate solution is represented by a vector with the dimension the number of variables. The value at position i represents the value of the corresponding variable, x_i . The algorithm works with complete solutions, i.e. all variables are instantiated. Each individual in the population has associated a measure of its fitness in the environment. The fitness function counts the number of violated constraints by the candidate solution.

In an EA the search for better individuals is conducted by the crossover operator, while the diversity in the population is maintained by the mutation operator.

The recombination operator is a fitness based scanning crossover. The scanning operator takes as input a number of chromosomes and returns one child. It chooses one of the i -th genes of the n parents to be the i -th gene of the child. For creating the new solution, the best genes are preserved. Our crossover makes use of the pre-processing information gathered with the inference process. It uses the functions returned by the mini-bucket algorithm, $f^*(x^p)$ to decide the values of the child. The variables are instantiated in a given order, the same as the one used in the mini-bucket algorithm. The order is determined with a deterministic heuristic. A new value to the next variable is assigned by choosing the best value from the parents according to the evaluation functions f^* . As stated before, these heuristic functions provide an upper bound on the cost of the best extension of a given partial assignment.

Algorithm 1. `multiparent_crossover($P(t), k$)`

```

for each set of  $k$  parents,  $p_1, \dots, p_k$  do
  for each position  $i$  in the ordering do
     $child_i \leftarrow best(p_{1i}, \dots, p_{ki})$ 
    /* use  $f^*(p_1^i), \dots, f^*(p_k^i)$  in  $best$  */
     $parent \leftarrow get\_worst\_parent(p_1, \dots, p_k)$ 
     $replace(parent, child)$ 
  end for
end for

```

This recombination operator intensifies the exploitation of the search space. It will generate new solutions for evaluations if there is sufficient diversity in the population. An operator to preserve variation is necessary. The mutation operator has this function, i.e. it serves for exploration. The operator assigns a new random value for a given variable.

After the application of the operators, the new individuals will replace the parents. Selection will take place next to ensure the preservation of fittest individuals. A fitness-based selection was chosen for experiments.

Because the crossover and the selection direct the search to most fit individuals, there is a chance of getting stuck in local minima. There is a need to leave the local minima and to explore different parts of the search space. A way of escaping from this minima must be included in the approach. From the available techniques, we have chosen the adaptive parameter control model which takes feedback from the search [13]. A possible way is to increase the mutation probability each time a local minima is reached. For a number of iterations the mutation probability was increased with a parameter that satisfies a Gaussian distribution. Mutating a gene means perturbing their value with a random number drawn from a Gaussian distribution $N(0, \sigma)$.

Another mode which was also tested in our approach is the earliest breakout mechanism [14]. When the algorithm is trapped in a local minimum point, a breakout is created for each nogood that appears in this current optimum. The weight for each newly created breakout is equal with one. If the breakout already exists, its weight is incremented by one. A predefined percent of the total weights (penalties) for an individual that violates these breakouts are added to the fitness function. In this manner the search is forced to put more emphasis on the constraints that are hard to satisfy. The evaluation function is an adaptive function because it is changed during the execution of the algorithm.

4 Tests and Results

4.1 Experimental Settings

We have considered binary CSPs where each constraint can not have more than two variables. The approach was tested on two well-known models for generating CSPs.

Algorithm 2. GA-MBE-Breakouts(i)

```

apply MBE(i)
 $t \leftarrow 0$ 
initialize  $P(t)$ 
evaluate  $P(t)$ 
while termination condition not meet do
   $t \leftarrow t + 1$ 
  select  $P(t)$  from  $P(t - 1)$ 
  multiparent_crossover ( $P(t), k$ )
  mutation( $P(t)$ )
  evaluate  $P(t)$ 
  if no diversity then
    get the list of breakouts from the best individual
    introduce the breakouts in the fitness
  end if
end while

```

The four parameter model [15], called **model B** did not allow the repetition of the constraints. A random CSP is given by four parameters (N, K, C, T) where N represents the number of variables, K the domain size, C the number of constraints and T the constraint tightness. The tightness represents the number of tuples not allowed. C constraints are selected uniformly at random from the available $N(N - 1)/2$ ones and for each constraint T nogoods are selected from the available K^2 tuples. The problems were first solved using a complete algorithm PFC-MRDAC [16]. This algorithm is an improved branch-and-bound algorithm, specifically designed for the Max-CSP problem. Because the networks generated are not necessarily solvable, the optimal solution needed for computing the accuracy was the solution found by the branch-and-bound algorithm. We have tested the approach on some over-constrained classes of binary CSPs. The selected classes are sparse $\langle 25, 10, 37, T \rangle$, with medium density $\langle 15, 10, 50, T \rangle$ and complete graphs $\langle 10, 10, 45, T \rangle$. For each class of problem the algorithms were tested on 50 instances.

In order to compare our results with the performance of other evolutionary algorithms we investigate the approach against the set of CSP instances made available by Craenen et al. on the Web ¹. These instances are generated using the **model E** [17]. We have experimented with 175 solvable problem instances: 25 instances for different values of p in model $E(20, 20, p, 2)$. Parameter p takes the following values: $\{0.24, 0.25, 0.26, 0.27, 0.28, 0.29, 0.30\}$. All instances considered were solvable.

The variable ordering used in MBE was determined using the min-induced-width heuristic. This method places the variable with the minimum degree last in the ordering. It connects then all of the variable neighbors, removes the node and all its adjacent edges and next repeats the procedure.

¹ http://www.xs4all.nl/~bcraenen/resources/csps_modelE_v20_d20.tar.gz

To measure the performance of the approach we have used the accuracy ratio $opt = F_{alg}/F_{Max_CSP}$, where F_{alg} represents the value of the solution found by the test algorithm and F_{Max_CSP} is the optimal solution.

The number of parents for the crossover operator was established to five. The genetic algorithm stops after a certain number of iterations or when a solution is found.

4.2 Results

Model B. The results for model B are given in Table 1, 2, and 3 for the version of the evolutionary algorithm which uses the breakouts mechanism. Similar results are obtained using the adaptive mutation model. We included the computations only for the first model of GA.

For each instance we perform five independent runs for GA and the best solution was taken. The percent of the problems that were solved was recorded in Table 1.

Table 1. Results for model B: the percent of solved problems, with the accuracy greater than 0.95

Instance	MBE	MBE	MBE
	GA-MBE i=2 solved	GA-MBE i=4 solved	GA-MBE i=6 solved
N=10, K=10, C=45, T=84	4	10	36
	96	92	98
N=10, K=10, C=45, T=85	4	16	56
	92	96	100
N=15, K=10, C=50, T=84	4	8	48
	76	92	100
N=10, K=10, C=45, T=85	2	10	36
	76	80	88
N=25, K=10, C=37, T=84	16	90	100
	94	98	100
N=10, K=10, C=45, T=85	18	84	100
	86	100	100

Each block from Table 1 corresponds to a class of CSPs, with different constraint tightness. The results are given for different values of parameter i , the level of inference. For example, for the class of problems $N = 15, K = 10, C = 50, T = 84$ the MBE with $i = 4$ solved only 8% of the generated problems. Our approach solved 92% of the problems. The Mini-Bucket algorithm solves more problems when the bound i increases. However, the time increases too.

It can be observed that the hybrid genetic algorithm increases the performance of the Mini-bucket. The performance of the genetic algorithm is higher when using a higher i -bound. This proves that the genetic algorithm uses efficiently

Table 2. Average constraint checks for the GA-MBE algorithm

Instance	T	i=2	i=4	i=6
N=10, K=10, C=45	84	$3.6 \cdot 10^6$	$5.1 \cdot 10^6$	$2.8 \cdot 10^6$
	85	$3.9 \cdot 10^6$	$3.8 \cdot 10^6$	$2.7 \cdot 10^6$
N=15, K=10, C=50	84	$5.7 \cdot 10^6$	$6.6 \cdot 10^6$	$4.2 \cdot 10^6$
	85	$6.1 \cdot 10^6$	$6.4 \cdot 10^6$	$5.1 \cdot 10^6$
N=25, K=10, C=37	84	$8.7 \cdot 10^6$	$1.7 \cdot 10^6$	$7.1 \cdot 10^5$
	85	$9.1 \cdot 10^6$	$3.4 \cdot 10^6$	$8.8 \cdot 10^5$

Table 3. Average time for MBE and GA-MBE algorithms

Instance	Algorithm	i=2	i=4	i=6
N=10, K=10, C=45, T=84	MBE	12	752	61440
	GA-MBE	2300	2541	1393
N=15, K=10, C=50, T=84	MBE	15	1154	91597
	GA-MBE	15782	14673	2550
N=25, K=10, C=37, T=84	MBE	51	1293	2628
	GA-MBE	7205	2015	357

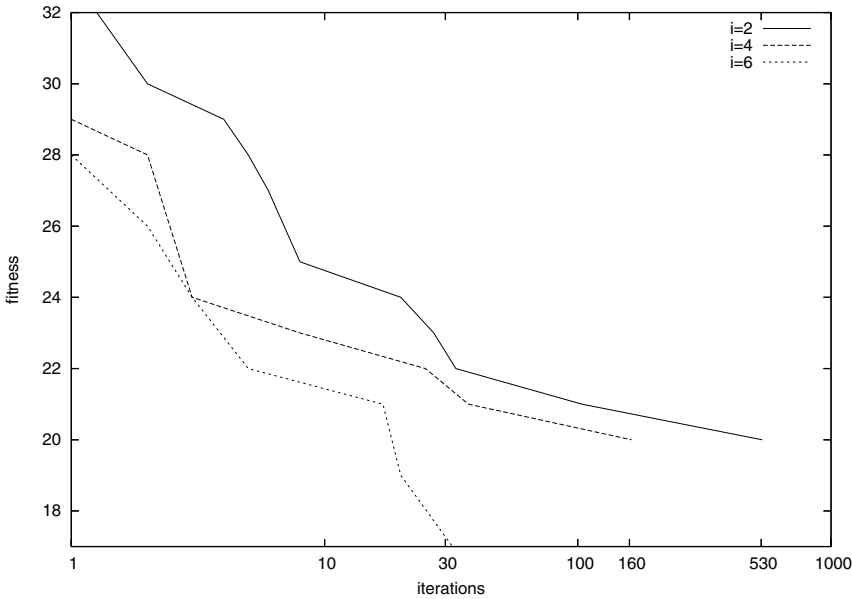


Fig. 1. Best fitness for a run on class $N = 15, K = 10, C = 50, T = 84$

the information gained by pre-processing. But there must be a trade-off between the preprocessing and the search realized by the evolutionary algorithm. This remark also results from the next tables.

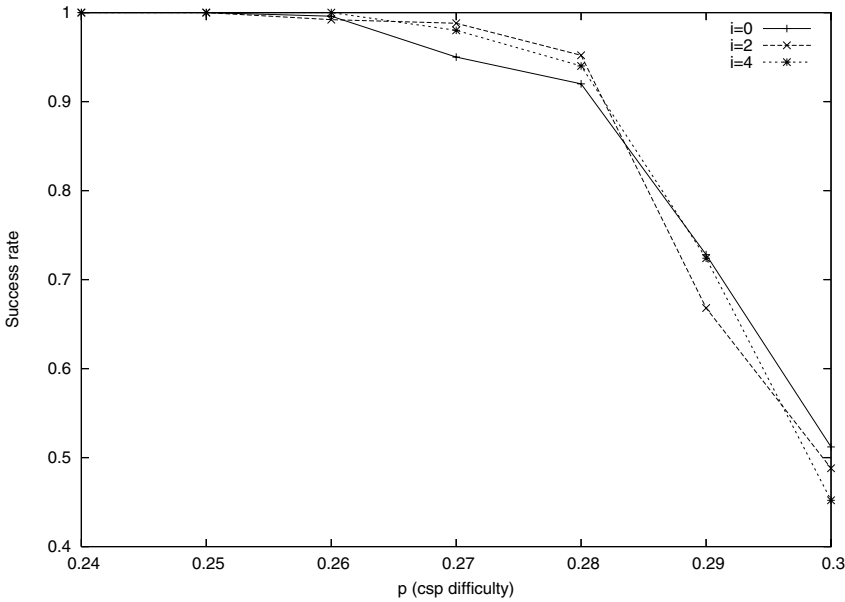


Fig. 2. Success rate for GA-MBE on instances of model E

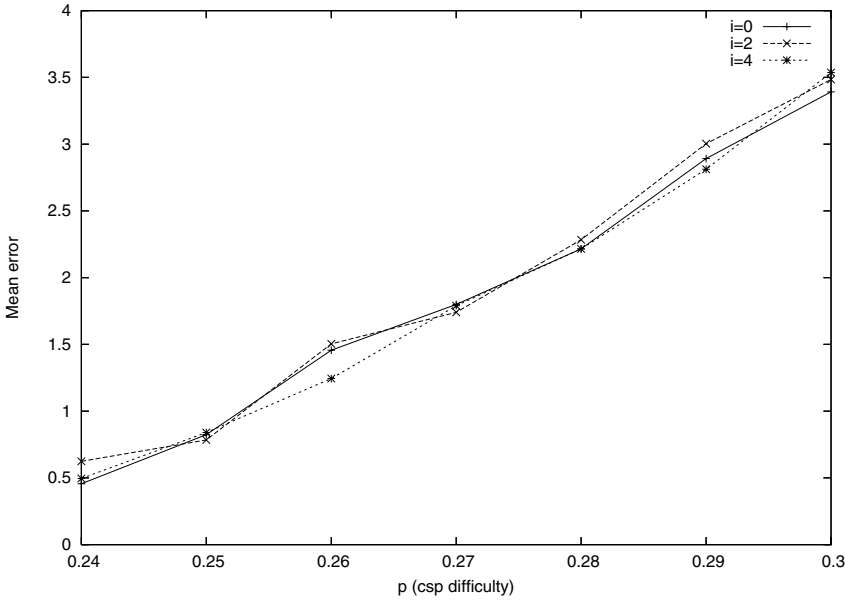


Fig. 3. Mean fitness for GA-MBE on instances of model E

For the evaluation we have also used an additional criterium. The standard measure of the efficiency of an evolutionary algorithm, the number of fitness evaluations is not very useful in this context. The use of heuristics implies more computation that is invisible for this metric. Therefore we have calculated the average number of conflict checks (Table 2).

In general, it is expected that the number of constraint checks increases with the obtaining of better results. The use of more conflict checks means learning more about the problem. The situation is a little different here. In Table 2 on the last column ($i = 6$) the number of constraint checks is smaller than the one in the column with $i = 2$. This happens because more information about the search space is gained in the MBE heuristic. It seems that the number of constraint checks decreases with the producing of more inference. In the case of a medium level of inference ($i = 4$) for the complete and medium classes of problems, the number of constraint checks reaches a maximum value.

The time needed for solving the binary CSPs is shown in Table 3 for some instances. GA-MBE with small inference can be preferred to MBE with greater inference, because the precision and the time are better for the first one.

Comparing with the results from [9], the gain from inference is bigger with the evolutionary algorithm than with a deterministic one (a Branch-and-Bound or a Best-First). Also, from Figure 1 it can be observed that with a stronger inference less iterations are necessary.

Model E. The results for model E are given in Figure 2 and 3. As measures of effectiveness, the success rate and the mean error at termination was used. The success rate represents the percentage of runs that find a good solution. As in the previous case, we consider only the solutions that have a ratio greater than 0.98. The error at termination for a run is equal with the number of constraints that are violated by the best solution, at the end of the algorithm.

The performance of the algorithm decreases with the difficulty of the problem. The results are not so clear as in the case of model B, regarding the level of inference used. We can also observe that the mean error is small, meaning that the algorithm is stable (Figure 3). This feature is very important for such kind of problems.

Using the results from the comparative study of several genetic algorithms made by Craenen et al. [6] we can conclude that the performance of our algorithm is comparable with that of the best GAs in the CSP field: SAW and Glass-Box GA. Low levels of inference slightly improve the performance of our algorithm on difficult CSP instances; higher levels of inference are needed.

5 Conclusions

A new approach for solving binary CSPs was presented. The approach uses the heuristics extracted from an approximation inference scheme inside an evolutionary algorithm. The evolutionary algorithm is adapted to exploit the heuristic information. This new approach is tested and compared with the Mini-Bucket

algorithm and with previous state of the art GAs for CSPs. Our results demonstrate the effectiveness of this approach. Of course, future evaluation work on real world CSPs is necessary. Moreover, some improvements can be obtained by choosing an evolutionary technique for determining the order of the variables.

References

1. Dechter, R.: Constraint Processing. Morgan Kaufmann Publishers (2003)
2. Michalewicz, Z.: A survey of constraint handling techniques in evolutionary computation methods. Proceedings of the 4th Annual Conference on Evolutionary Programming (1995) 135-155
3. Dozier, G., Bowen, J., Bahler, D.: Solving small and large constraint satisfaction problems using a heuristic-based microgenetic algorithm. Proceedings of the 1st IEEE Conference on Evolutionary Computation (1994) 306-311
4. Paredis, J.: Coevolutionary constraint satisfaction. Proceedings of the 3rd Conference on Parallel Problem Solving from Nature, 866 (1994) 46-55
5. Eiben, A.E., Ruttkay, Zs.: Self-adaptivity for constraint satisfaction: Learning penalty functions. In Proceedings of the 3rd IEEE Conference on Evolutionary Computation (1996) 258-261
6. Craenen, B.G.W., Eiben, A.E., van Hemert, J.I.: Comparing Evolutionary Algorithms on Binary Constraint Satisfaction Problems. IEEE Transactions on Evolutionary Computation, **7(5)** (2003) 424-444
7. Eiben, A.E., Raue, P.-E., Ruttkay, Zs.: Solving constraint satisfaction problems using genetic algorithms. Proceedings of the 1st IEEE Conference on Evolutionary Computation (1994) 542-547
8. Marchiori, E., Steenbeek, A.: A Genetic Local Search Algorithm for Random Binary Constraint Satisfaction Problems. Proceedings of the 14th Annual Symposium on Applied Computing (2000) 458-462
9. Kask, K., Dechter, R.: New Search Heuristics for Max-CSP. Principles and Practice of Constraint Programming (2000) 262-277
10. Dechter, R.: Bucket elimination: A unifying framework for reasoning. Artificial Intelligence, **113** (1999) 41-85
11. Dechter, R., Rish, I.: Mini-buckets: A general scheme for bounded inference. Journal of the ACM, **50(2)** (2003) 107-153
12. Michalewicz, Z.: Genetic Algorithms + Data structures = Evolution programs. Springer Berlin 3rd edition (1996)
13. Hinterding, R., Michalewicz, Z., Eiben, A.E.: Adaptation in evolutionary computation: a survey. Proceedings of the 4th IEEE Conference on Evolutionary Computation (1997) 65-69
14. Morris, P.: The breakout method for escaping from local minima. Proceedings of the 11th National Conference on Artificial Intelligence, AAAI (1993) 40-45
15. Smith, B.: Phase transition and the mushy region in constraint satisfaction. Proceedings of the 11th ECAI (1994) 100-104
16. Larossa, J., Meseguer, P.: Partial Lazy Forward Checking for MAX-CSP. Proceedings of the 13th European Conference on Artificial Intelligence (1998) 229-233
17. Achlioptas, D., Kirovski, L.M., Kranakis, E., Krizanc, D., Molloy, M.S.O. and Stamatiou, Y.C.: Random constraint satisfaction: A more accurate picture. Constraints, **4(6)** (2001) 329-344

Scheduling Social Golfers with Memetic Evolutionary Programming

Carlos Cotta¹, Iván Dotú², Antonio J. Fernández¹,
and Pascal Van Hentenryck³

¹ Dpto. de Lenguajes y Ciencias de la Computación, Universidad de Málaga, Spain

² Dpto. de Ingeniería Informática, Universidad Autónoma de Madrid, Spain

³ Brown University, Box 1910, Providence, RI 02912, USA

Abstract. The social golfer problem (SGP) has attracted significant attention in recent years because of its highly symmetrical, constrained, and combinatorial nature. Nowadays, it constitutes one of the standard benchmarks in the area of constraint programming. This paper presents the first evolutionary approach to the SGP. We propose a memetic algorithm (MA) that combines ideas from evolutionary programming and tabu search. In order to lessen the influence of the high number of symmetries present in the problem, the MA does not make use of recombination operators. The search is thus propelled by selection, mutation, and local search. In connection with the latter, we analyze the effect of baldwinian and lamarckian learning in the performance of the MA. An experimental study shows that the MA is capable of improving results reported in the literature, and supports the superiority of lamarckian strategies in this problem.

1 Introduction

The social golfer problem has attracted significant interest since it was first posted on `sci.op-research` in May 1998. It consists of scheduling $n = g \cdot s$ golfers into g groups of s players every week for w weeks so that no two golfers play in the same group more than once. The problem can be regarded as an optimization problem if for two given values for g and s , we ask for the maximum number of weeks w the golfers can play together.

It can be easily inferred from the informal definition of the SGP given before that it constitutes a highly combinatorial, constrained, and symmetric problem. Not surprisingly, a lot of attention has been devoted to the SGP in the constraint programming community (e.g., see [1,2,3] among others). Indeed, it raises fundamentally interesting issues in modelling and symmetry breaking, and has become one of the standard benchmarks in the area. Notice in this sense that symmetry is manifold in this problem, e.g., players can be permuted within groups, groups can be ordered arbitrarily within every week, and even the weeks themselves can be permuted. Recent developments (e.g., [4]) approach the scheduling of social golfers using innovative, elegant, but also complex, symmetry-breaking schemes.

To the best of our knowledge, no evolutionary approach has been reported in the literature to handle this problem. Here, we present a memetic algorithm (MA) that is based on the hybridization of evolutionary programming and tabu search, and that constitutes the first attempt of tackling the SGP by evolutionary techniques. While deterministic techniques such as constraint programming have addressed the SGP by detecting and breaking symmetries (e.g., [5,6,7,8]), the flexibility of MAs eases the handling of these symmetries. To be precise, their influence is mostly confined to sexual-reproduction operators such as recombination. However, as shown in this work, a MA based on selection, mutation and local search still constitutes a powerful tool for optimization, capable of performing at a state-of-the-art level for this problem.

2 The Social Golfer Problem

As mentioned in previous section, the Social Golfer Problem (SGP) consists of scheduling $n = g \cdot s$ golfers into g groups of s players every week for w weeks, so that no two golfers play in the same group more than once. An instance of the social golfer is thus specified by a triplet $\langle g, s, w \rangle$. A (potentially infeasible) solution for such an instance is given by a schedule $\sigma : \mathbb{N}_g \times \mathbb{N}_w \longrightarrow 2^{\mathbb{N}_n}$, where $\mathbb{N}_i = \{1, 2, \dots, i\}$, and $|\sigma(i, j)| = s$ for all $i \in \mathbb{N}_g, j \in \mathbb{N}_w$, that is, a function that on input (i, j) returns the set of s players that constitute the i -th group of the j -th week.

2.1 Modelling the SGP

There are many possible modelings for the social golfer problem, which is one of the reasons why it is so interesting. In a generalized way, this problem can be modelled as a constraint satisfaction problem (CSP) defined by the following constraints:

- A golfer plays exactly once a week, i.e.,

$$\forall p \in \mathbb{N}_n : \forall j \in \mathbb{N}_w : \exists! i \in \mathbb{N}_g : p \in \sigma(i, j). \quad (1)$$

We will use the notation $\gamma(p, j)$ to denote the index of the group in which golfer p plays during the j -th week. This constraint can be also formalized by claiming that no two groups in the same week intersect, i.e.,

$$\forall j \in \mathbb{N}_w : \forall i, i' \in \mathbb{N}_g, i \neq i' : \sigma(i, j) \cap \sigma(i', j) = \emptyset. \quad (2)$$

- No two golfers play together more than once, i.e.,

$$\forall j, j' \in \mathbb{N}_w : \forall i, i' \in \mathbb{N}_g, i \neq i' : |\sigma(i, j) \cap \sigma(i', j')| \leq 1. \quad (3)$$

Let $\#_\sigma(a, b)$ be the number of times golfers a and b play together in schedule σ , i.e.,

$$\#_\sigma(a, b) = \sum_{i \in \mathbb{N}_g} \sum_{j \in \mathbb{N}_w} [\{a, b\} \subseteq \sigma(i, j)], \quad (4)$$

where $[\cdot]$ is the Iverson bracket, namely $[\mathbf{true}] = 1$ and $[\mathbf{false}] = 0$. We define the degree of violation of a constraint a -and- b -play-together-at-most-once $v_\sigma(a, b) = \max(0, \#_\sigma(a, b) - 1)$.

In addition to the tightly constrained structure of feasible solutions, this problem is of the foremost interest due to its high degree of symmetry. Symmetries can appear in this problem because:

- Golfers are interchangeable inside groups. This means $(s!)^{g \cdot w}$ symmetries. Notice that this symmetry arises in naive formulations of the problem in which the schedule function is defined to return a *list* of golfers for each group and week, rather than a *set* of golfers.
- Groups within a week can be exchanged. This amounts to the fact that group indexes bear no absolute meaning within a week, and implies $(g!)^w$ symmetries.
- Weeks can be arbitrarily reordered, that is, given a schedule σ , we can obtain another one by simply permutating the weekly schedules. Therefore, this means $w!$ symmetries.
- Golfers can be renumbered. This means exactly $n!$ –i.e., $(g \cdot s)!$ – symmetries.

As a consequence, a very naive formulation $\langle g, s, w \rangle$ has $(s!)^{g \cdot w} (g!)^w w! (gs)!$ symmetries. Observe that the symmetries grow very rapidly as the size of the problem grows, and this may be problematic for search algorithms that ignored them (they might be mislead in the case of heuristic approaches, and/or waste computational resources in the case of complete techniques).

The symmetry problem can be remedied in different ways. For instance the first kind of symmetry is implicitly removed by using sets for modelling groups, as mentioned before. As to the symmetry inside weeks, it can be removed by ordering groups using some pre-defined total order \prec (e.g., the lower the smallest element in a group is, lower the group in a week is, i.e., for any week j and $i \in \mathbb{N}_{g-1}$,

$$\sigma(i, j) \prec \sigma(i+1, j) \Leftrightarrow \min(\sigma(i, j)) < \min(\sigma(i+1, j)). \quad (5)$$

Observe that with this symmetry breaking procedure, golfer 1 is always in the first group in every week. The third symmetry can be handled in roughly the same way, that is, ordering weeks with respect to the first group in each week, using for this purpose the second lowest element in the first group, i.e., let w_i and w_{i+1} two weeks ($i \in \mathbb{N}_{w-1}$); then

$$w_i \prec w_{i+1} \Leftrightarrow \min(\sigma(1, i) \setminus \{1\}) < \min(\sigma(1, i+1) \setminus \{1\}). \quad (6)$$

Finally, symmetries among golfers are harder to handle: they can be fully removed only by using advanced techniques for dynamic symmetry breaking (see [4] for more details).

2.2 Related Work

Due to the interest that the SGP has attracted in the constraint satisfaction community, it has been extensively attacked using different techniques. Here, we

mention just some of the most recent advances in solving the SGP. To begin with, Harvey and Winterer [9] have proposed to construct solutions to the SGP by using sets of mutually orthogonal latin squares. Also, Gent and Lynce [10] have recently introduced a satisfiability (SAT) encoding for the SGP. Barnier and Brisset [4] have presented a combination of techniques to efficiently find solutions to a specific instance of SGP, the Kirkman's schoolgirl problem. Global constraints for lexicographic ordering have been proposed by Frisch *et al.* [11], being used for breaking symmetries in the SGP. Also, a tabu-based local search algorithm for the SGP is described by Dotú and Van Hentenryck [12].

The SGP problem also admits a number of possible variants; for instance finding a w -week schedule with “maximum socialization” (i.e., as few repeated pairs as possible), or finding a schedule of minimum length such that each golfer plays with every other golfer at least once (“full socialization”) [13]. In either case, and to the best of our knowledge, no evolutionary approach has been reported in the literature to handle this problem in any of these variants. Next section tackles this gap.

3 A Memetic Approach to the Social Golfer Problem

The application of standard population-based metaheuristics to the SGP is, if not thwarted, at least challenged by the presence of the manifold symmetries detailed in the previous section. To be precise, these symmetries are specifically relevant with respect to the performance of recombination operators (and generalizations thereof, that is, any reproductive operator constructing new solutions on the basis of two or more *parents*). If these symmetries are not implicitly broken by means of a wise representation of solutions, or the operators are not explicitly designed to take them into account, recombination attempts are doomed to fail: two similar solutions (even two identical solutions) can be considered as completely different solutions due to disregarded symmetries. As a consequence, it cannot be expected in general that the relevant features of these solutions (i.e., those information pieces ultimately responsible for the quality of the solutions) be processed in an adequate way. In this scenario, recombination is likely to behave as a highly disruptive, macromutation process. It turns out that this is precisely the general interpretation that is made of recombination in the realm of Evolutionary Programming (EP) [14]. For this reason, we have chosen an EP model as the base of our memetic approach, as shown next.

3.1 General Algorithmic Model

Following the philosophy of EP, our MA only uses mutation as the primary means to diversify the search. This relieves the need for performing symmetry breakages, and subsequently allows a simpler representation of solutions. Let σ be a w -week assignment for g groups of s players each, as described in Sect. 2. This assignment is encoded as a string $u = t_{11} :: t_{12} :: \dots :: t_{1g} :: \dots :: t_{wg}$, where $t_{ij} \in \mathbb{N}_{g \cdot s}^s$ is a permutation of the elements in the set $\sigma(i, j)$, and the operator

```

1:  for  $i \in [1 : \text{popsize}]$  do
2:    let  $\text{pop}[i] \leftarrow \text{GENERATESOLUTION}(g, s, w)$ 
3:    let  $f[i] \leftarrow \text{VIOLATEDCONSTRAINTS}(\text{pop}[i])$ 
4:  endfor
5:  let  $\text{iter} \leftarrow 0$ 
6:  do
7:    let  $u \leftarrow \text{SELECT}(\text{pop}, f)$ 
8:    let  $u' \leftarrow \text{MUTATE}(u)$ 
9:    let  $(u'', f') \leftarrow \text{LEARNING}(u')$ 
10:   let  $i \leftarrow \max_{i \in \{1, \text{popsize}\}}^{-1}(f[i])$ 
11:   let  $(\text{pop}[i], f[i]) \leftarrow (u'', f')$ 
12:   let  $\text{iter} \leftarrow \text{iter} + 1$ 
13: until  $\text{TERMINATIONCRITERION}(\text{pop}, f, \text{iter})$ 
14: return  $\text{pop} \left[ \min_{i \in \{1, \text{popsize}\}}^{-1}(f[i]) \right]$ 

```

Fig. 1. Pseudocode of the memetic EP approach

:: indicates string concatenation. Conversely, a string $u \in \mathbb{N}_{g \cdot s}^{s \cdot g \cdot w}$ is decoded into an assignment σ by dividing it into s -element chunks, taking the elements in each of them as the components of a set $\sigma(i, j)$, $i \in \mathbb{N}_g$, $j \in \mathbb{N}_w$. Although no symmetries are considered in this encoding, we do have considered a basic constraint in it, namely the fact that a certain golfer cannot be scheduled into two different groups in the same week. To do so, strings are initially generated from $\mathbb{P}_{g \cdot s}^w$, that is, as the concatenation of w permutations of the elements in $\{1, \dots, g \cdot s\}$. This structure of solutions is respected by all operators involved in the algorithm, whose overall pseudocode is shown in Fig. 1.

As it can be seen, our MA follows a steady-state evolution model, in which a single solution is selected, mutated, and subjected to a learning process. Regarding mutation, it is done by selecting two players from different groups in the same week, and switching their positions. The set of possible swaps is then

$$\mathcal{S}(\sigma) = \{(\langle w, p_1 \rangle, \langle w, p_2 \rangle) \mid \gamma(p_1, w) \neq \gamma(p_2, w)\}. \quad (7)$$

Each selected solution is subjected to a number of swaps that is Poisson-distributed with parameter $(g \cdot s)^{-1}$, that is, w swaps are performed on average. This procedure is respectful with the permutational structure of solutions, as mentioned before. As to the learning (i.e., improvement) process, it is done by means of an embedded tabu-search procedure (described in next subsection). Two strategies for conducting the learning have been considered: baldwinian and lamarckian. Firstly explored by Hinton and Nolan [15], baldwinian learning consist of performing a local improvement procedure, retaining the fitness value thus obtained, but discarding the phenotypical changes discovered. In some sense, this amounts to evaluating a solution on the basis of how good it could become, and bears some resemblance to natural evolution in that traits acquired during one's lifetime are not transmitted to the offspring. On the contrary, lamarckian learning does keep the improved phenotype as well, and injects the changes back to the

genotype. This kind of learning is akin to cultural (i.e., memetic) evolution, and provides faster convergence rates (e.g., see [16]). Nevertheless, it may be also prone to premature convergence to suboptimal solutions in some cases. Determining whether this is the case in the SGP has been one of the issues considered in the experimentation.

3.2 The Tabu Search Strategy

The local improvement strategy is based on the tabu-search (TS) template, and explores the neighborhood arising from swapping golfers from different groups in the same week. This is the same neighborhood used in individual mutations, but notice that the latter consists of the iterated application of a number of swaps; hence, mutation represents a long jump in the search space, as regarded by the TS algorithm. Furthermore, from the point of view of TS, it is more effective to restrict attention just to swaps involving at least one golfer in conflict with another golfer in the same group. This ensures that the algorithm focuses on swaps which may decrease the number of violations. More formally, a pair $\langle w, p \rangle$ is said to be in conflict in schedule σ (denoted by $v_\sigma(\langle w, p \rangle) = \text{true}$), if

$$\exists p' \in \sigma(\gamma(p, w), w), p' \neq p : v_\sigma(p, p') > 1. \quad (8)$$

With this restriction in mind, the set of swaps $\mathcal{S}^-(\sigma)$ considered for a schedule σ becomes

$$\mathcal{S}^-(\sigma) = \{(\langle w, p_1 \rangle, \langle w, p_2 \rangle) \in S(\sigma) \mid v_\sigma(\langle w, p_1 \rangle)\}. \quad (9)$$

The tabu component of the algorithm is based on three main ideas. First, the tabu list is distributed across the various weeks, which is natural since the swaps only consider golfers in the same week. The tabu component thus consists of an array *tabu*, where *tabu*[*w*] represents the tabu list associated with week *w*. Second, for a given week *w*, the tabu list maintains triplets $\langle a, b, i \rangle$, where *a* and *b* are two golfers, and *i* represents the first iteration where golfers *a* and *b* can be swapped again in week *w*. Third, the tabu tenure, i.e., the time a pair of golfers (*a*, *b*) stays in the list, is dynamic: it is randomly generated in the interval [4, 100]. In other words, each time a pair of golfers (*a*, *b*) is swapped, a random value ρ is drawn uniformly from the interval [4, 100] and the pair (*a*, *b*) is tabu for the next ρ iterations. As a consequence, for schedule σ and iteration *k*, the neighborhood consists of the set of moves $\mathcal{S}^t(\sigma, k)$ defined as

$$\mathcal{S}^t(\sigma, k) = \{(\langle w, p_1 \rangle, \langle w, p_2 \rangle) \in \mathcal{S}^-(\sigma) \mid \nexists k' > k : \langle p_1, p_2, k' \rangle \in \text{tabu}[w]\}. \quad (10)$$

In addition to the non-tabu moves, the neighborhood also considers moves that improve the best solution found so far, i.e., the set $\mathcal{S}^*(\sigma, \sigma^*)$ defined as

$$\mathcal{S}^*(\sigma, \sigma^*) = \{(t_1, t_2) \in \mathcal{S}^-(\sigma) \mid f(\sigma[t_1 \leftrightarrow t_2]) < f(\sigma^*)\}, \quad (11)$$

where $\sigma[(w, p_1) \leftrightarrow (w, p_2)]$ denotes the schedule σ where golfers p_1 and p_2 switch their groups in week *w*, and σ^* denotes the best solution found so far. Observe

```

1:  for  $i \in [1 : w]$  do let  $\text{tabu}[i] \leftarrow \emptyset$  endfor
2:  let  $\sigma^* \leftarrow \sigma$ 
3:  let  $k \leftarrow 0$ 
4:  while  $(k \leq \text{maxIter}) \wedge [f(\sigma) > 0]$  do
5:    let  $f^* \leftarrow \infty$ 
6:    for  $(t_1, t_2) \in \mathcal{S}^t(\sigma, k) \cup \mathcal{S}^*(\sigma, \sigma^*)$  do
7:      let  $f' \leftarrow f(\sigma[t_1 \leftrightarrow t_2])$ 
8:      if  $f' < f^*$  then
9:        let  $(t_1^*, t_2^*) \leftarrow (t_1, t_2)$ ; let  $f^* \leftarrow f'$ 
10:      endif
11:    endfor
12:    let  $\tau \leftarrow \text{RANDOM}([4, 100])$ 
13:    let  $\text{tabu}[\omega(t_1^*)] \leftarrow \text{tabu}[\omega(t_1^*)] \cup \{(\pi(t_1^*), \pi(t_2^*), k + \tau)\}$ 
14:    let  $\sigma \leftarrow \sigma[t_1^* \leftrightarrow t_2^*]$ 
15:    if  $f^* < f(\sigma^*)$  then
16:      let  $\sigma^* \leftarrow \sigma$ 
17:    endif
18:    let  $k \leftarrow k + 1$ 
19:  endwhile
20:  return  $(\sigma^*, f(\sigma^*))$ 

```

Fig. 2. Pseudocode of the tabu-search component of the MA

that the expression $f(\sigma[t_1 \leftrightarrow t_2])$ represents the number of violations obtained after performing the corresponding swap.

With the so-defined neighborhoods, the TS algorithm is described in Fig. 2. The core of the algorithm is given in lines 4-19, where local moves are iterated for a maximum number of iterations or until a solution is found. The local move is selected in line 9. The key idea is to select the best swaps in the neighborhood $\mathcal{S}^t(\sigma, k) \cup \mathcal{S}^*(\sigma, \sigma^*)$, i.e., the non-tabu swaps and those improving the best schedule. The tabu list is updated in line 13, where $\omega(\langle w, p \rangle) = w$, and $\pi(\langle w, p \rangle) = p$. The algorithm returns the best solution found and its quality.

4 Experimental Results

The experiments have been done with the steady-state MA described in previous section, using a population size of 25 individuals, binary tournament selection, and a maximum number of 2,500 evaluations. Each invocation to the TS algorithm uses $\text{maxIter} = g \cdot w$, so that there exists the possibility that the solution resulting from learning had no group in common with the original one. For each pair (g, s) , we have performed series of 20 runs per increasing values of w , to determine the maximum number of weeks for which a feasible schedule can be found. The experiments have been done both with the lamarckian and the baldwinian variants of the MA.

The results are shown in Fig. 3 and Fig. 4. For comparison purposes, we include the results reported in [12] corresponding to the stand-alone application

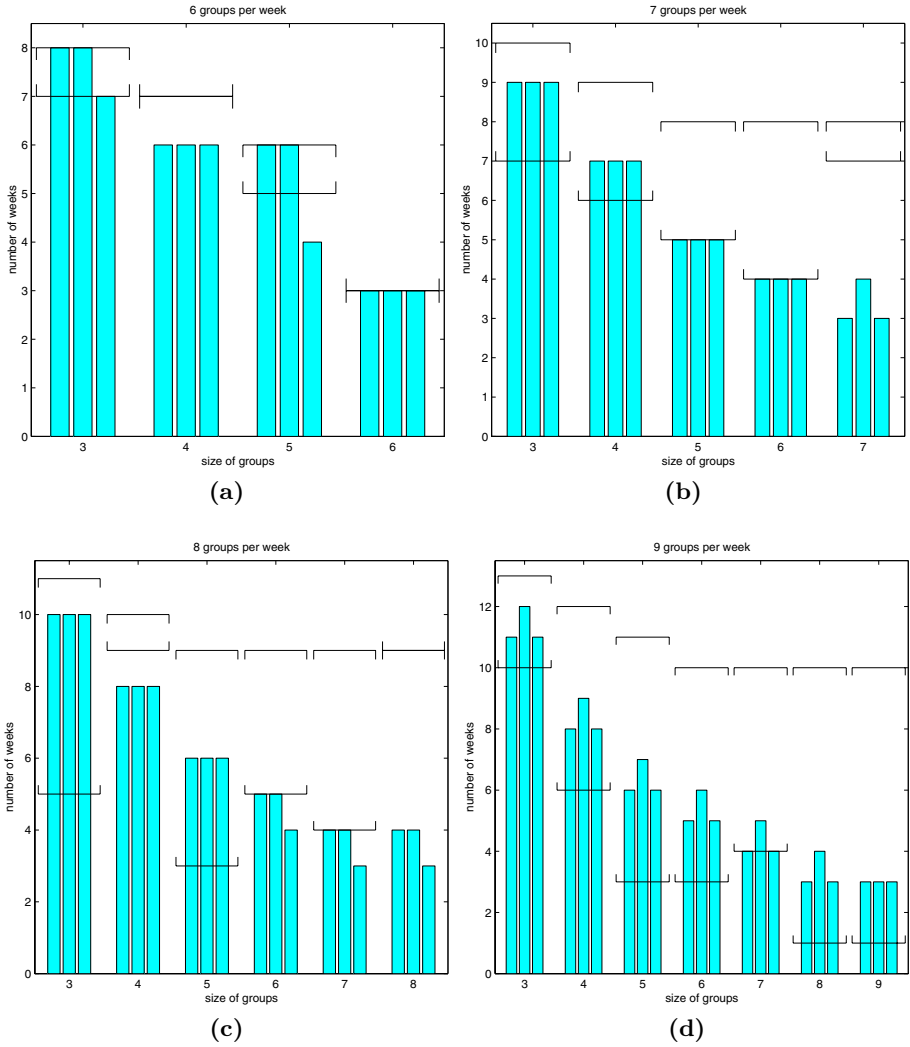


Fig. 3. Results for six groups per week (a), seven groups per week (b), eight groups per week (c), and nine groups per week (d). Each group of three bars represent (from left to right) the maximum number of weeks solved by stand-alone *TS*, by the lamarckian MA, and by the baldwinian MA. The horizontal brackets indicate the lower and upper bounds for the corresponding instances.

of an extended version of the TS strategy used within our MA, incorporating reinitialization mechanisms. We also indicate the upper and lower bounds for the corresponding problem instances, as reported in [17]. Notice that many of these lower bounds (i.e., best known solutions) are actually superseded by the plain application of TS. Moreover, the lamarckian MA is capable of further improving these solutions, providing better solutions for 12 problem instances,

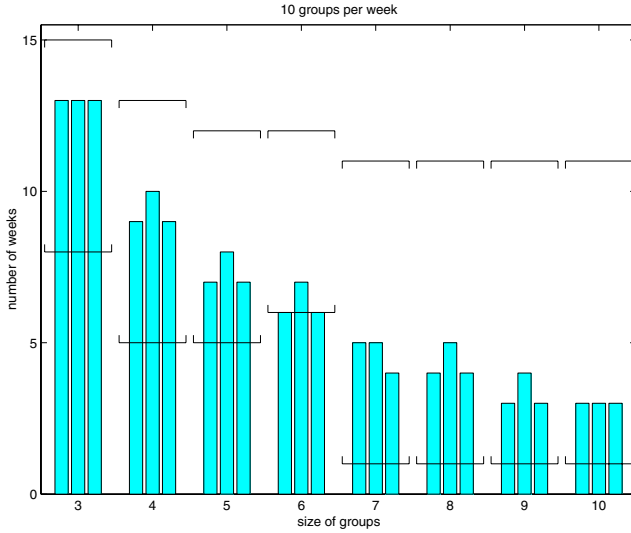


Fig. 4. Results for ten groups per week. Each group of three bars represent (from left to right) the maximum number of weeks solved by stand-alone *TS*, by the lamarckian MA, and by the baldwinian MA. The horizontal brackets indicate the lower and upper bounds for the corresponding instances.

and achieving the same results of TS in the remaining ones. Notice also that the performance of the baldwinian MA is markedly inferior to that of its lamarckian counterpart. Several factors may be responsible for this behavior. On one hand, the search space can have a very complex topology, thus making the baldwinian information harder to use (the improved solution found by the TS component may involve a convoluted path in the search space, very difficult to trace by means of mutation and selection). On the other hand, and related to the previous point, longer run times may be necessary for the baldwinian MA to achieve the performance level of the lamarckian MA (in these experiments, each run took from a few seconds up to around twenty minutes –in a P4 3GHz 1GB winXP computer– for the larger instances). Then again, this indicates the superiority of the latter strategy in this problem.

Further details on the performance of the two MA variants are provided in Table 1. The data correspond to the average results in the largest problem instances they could solve, thus offering a glimpse of their behavior at the edge of solvability. Obviously, these limits were shown to be farther for the lamarckian MA in Fig. 3 and Fig. 4, and hence the tables must be studied with caution. Although entries do not always correspond to homogeneous instance sizes (number of weeks in this case), we do know that the success rate for the next larger instance size is 0%. If we couple this fact with the observation that the lower part of Table 1 (i.e., the baldwinian MA) has a larger number of 100%-success entries than the upper part, we can conclude that the lamarckian MA does not simply perform better, but it also has a more gradual decline in performance

Table 1. Detailed results of the lamarckian MA (top) and the baldwinian MA (bottom) in the largest solved instance for each combination of s (size of groups) and g (groups per week). Each triplet of numbers indicate from left to right the success rate in $n = 20$ runs, the mean number of violated constraints in the best solutions found in these runs, and the standard error of the mean (σ/\sqrt{n}).

Lamarckian MA												
size (s)	groups per week (g)											
	6			7			8			9		
3	.25	.75	.10	1.00	.00	.00	1.00	.00	.00	.10	1.75	.14
4	.20	1.05	.15	.50	.65	.16	.50	.65	.16	.70	.50	.18
5	.10	5.90	.48	1.00	.00	.00	1.00	.00	.00	1.00	.00	.00
6	1.00	.00	.00	1.00	.00	.00	1.00	.00	.00	1.00	.00	.00
7	—	—	—	.05	1.95	.11	1.00	.00	.00	1.00	.00	.00
8	—	—	—	—	—	—	.05	2.50	.19	1.00	.00	.00
9	—	—	—	—	—	—	—	—	—	1.00	.00	.00
10	—	—	—	—	—	—	—	—	—	—	1.00	.00

Baldwinian MA												
size (s)	groups per week (g)											
	6			7			8			9		
3	1.00	.00	.00	1.00	.00	.00	1.00	.00	.00	1.00	.00	.00
4	.05	1.70	.12	.05	2.00	.14	1.00	.00	.00	1.00	.00	.00
5	1.00	.00	.00	1.00	.00	.00	1.00	.00	.00	1.00	.00	.00
6	1.00	.00	.00	1.00	.00	.00	1.00	.00	.00	1.00	.00	.00
7	—	—	—	1.00	.00	.00	1.00	.00	.00	1.00	.00	.00
8	—	—	—	—	—	—	1.00	.00	.00	1.00	.00	.00
9	—	—	—	—	—	—	—	—	—	1.00	.00	.00
10	—	—	—	—	—	—	—	—	—	—	.15	2.40

for increasing instance sizes. Therefore, it seems to be more scalable (or at least less sensitive to the curse of dimensionality) than the baldwinian MA. The latter exhibits an abrupt performance drop from full solvability capacity to null such capacity.

5 Conclusions and Future Work

We have presented here the first evolutionary approach to the Social Golfer Problem. Combining ideas from the realm of evolutionary programming and tabu search, we have devised a memetic algorithm capable of improving results reported in the literature. In this sense, we believe that the incorporation of intensification mechanisms such as *ad hoc* local searchers is essential to tackle this problem. Indeed, given the fact that a less-intensive strategy based in baldwinian learning performs worse than a pure lamarckian version, we hypothesize that lesser-intensive algorithms based on unbiased variation plus selection are not adequate for this problem either.

As mentioned in previous sections, symmetries play a major role in this problem. Although we have opted for not using recombination mechanisms, hence diminishing the impact of these symmetries, their consideration is an important line for future developments. We intend to approach the breakage of symmetries by smart representations and/or by problem-aware recombination operators, subsequently examining whether this symmetry-free approach results in a significant performance change.

We also plan to introduce further problem knowledge in other components of the algorithm. In this sense, Dotú and Van Hentenryck [12] have devised a constructive approach that can be shown to provide feasible solutions for certain values of w when g and s are equal and odd. In other cases, this constructive heuristic can provide a good starting point for local search. The overall results of TS endowed with this constructive heuristic are still similar to those of the lamarckian MA (despite the latter starts from a purely random initial population). Injecting the solutions provided by this constructive heuristic into the initial population may boost the performance of the MA. This issue will be dealt in the future as well.

Acknowledgements

This work was partially supported by Spanish MCyT under contracts TIN2004-7943-C04-01 and TIN2005-08818-C04-01.

References

1. Fahle, T., Schamberger, S., Sellmann, M.: Symmetry breaking. In Walsh, T., ed.: 7th International Conference on Principles and Practice of Constraint Programming. Volume 2239 of Lecture Notes in Computer Science., Paphos, Cyprus, Springer (2001) 93–107
2. Smith, B.M.: Reducing symmetry in a combinatorial design problem. In: Third International Workshop on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. (2001) 351–359
3. Sellmann, M., Harvey, W.: Heuristic constraint propagation. In Hentenryck, P.V., ed.: 8th International Conference on Principles and Practice of Constraint Programming. Volume 2470 of Lecture Notes in Computer Science., Ithaca, NY, USA, Springer (2002) 738–743
4. Barnier, N., Brisset, P.: Solving kirkman's schoolgirl problem in a few seconds. *Constraints* **10** (2005) 7–21
5. Ramani, A., Markov, I.: Automatically exploiting symmetries in constraint programming. In Faltings, B., Petcu, A., Fages, F., Rossi, F., eds.: Recent Advances in Constraints, Joint ERCIM/CoLogNet International Workshop on Constraint Solving and Constraint Logic Programming, CSCLP 2004. Volume 3419 of Lecture Notes in Computer Science., Lausanne, Switzerland, Springer (2005) 98–112 Revised Selected and Invited Papers.

6. Prestwich, S., Roli, A.: Symmetry breaking and local search spaces. In Barták, R., Milano, M., eds.: Second International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Volume 3524 of Lecture Notes in Computer Science., Prague, Czech Republic, Springer (2005) 273–287
7. Mancini, T., Cadoli, M.: Detecting and breaking symmetries by reasoning on problem specifications. In Zucker, J.D., Saitta, L., eds.: International Symposium on Abstraction, Reformulation and Approximation (SARA 2005). Volume 3607 of Lecture Notes in Computer Science., Airth Castle, Scotland, UK, Springer (2005) 165–181
8. Sellmann, M., Hentenryck, P.V.: Structural symmetry breaking. In Kaelbling, L.P., Saffioti, A., eds.: Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05), Edinburgh, Scotland, Professional Book Center (2005) 298–303
9. Harvey, W., Winterer, T.: Solving the MOLR and social golfers problems. In van Beek, P., ed.: 11th International Conference on Principles and Practice of Constraint Programming. Volume 3709 of Lecture Notes in Computer Science., Sitges, Spain, Springer (2005) 286–300
10. Gent, I., Lynce, I.: A SAT encoding for the social golfer problem. In: IJCAI'05 workshop on Modelling and Solving Problems with Constraints, Edinburgh, Scotland (2005)
11. Frisch, A., Hnich, B., Kiziltan, Z., Miguel, I., Walsh, T.: Global constraints for lexicographic orderings. In Hentenryck, P.V., ed.: 8th International Conference on Principles and Practice of Constraint Programming. Volume 2470 of Lecture Notes in Computer Science., Ithaca, NY, USA, Springer (2002) 93–108
12. Dotú, I., Hentenryck, P.V.: Scheduling social golfers locally. In Barták, R., Milano, M., eds.: International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems 2005. Volume 3524 of Lecture Notes in Computer Science., Prague, Czech Republic, Springer-Verlag (2005) 155–167
13. Gent, I., Walsh, T.: CSPLIB: A benchmark library for constraints. In Jaffar, J., ed.: 5th International Conference on Principles and Practice of Constraint Programming (CP'99). Volume 1713 of Lecture Notes in Computer Science., Alexandria, Virginia, USA, Springer (1999) 480–481
14. Fogel, L., Owens, A., Walsh, M.: Artificial Intelligence Through Simulated Evolution. Wiley, New York NY (1966)
15. Hinton, G., Nolan, S.: How learning can guide evolution. *Complex Systems* **1** (1987) 495–502
16. Whitley, D., Gordon, S., Mathias, K.: Lamarckian evolution, the baldwin effect and function optimization. In Davidor, Y., Schwefel, H.P., Männer, R., eds.: Parallel Problem Solving from Nature III. Volume 866 of Lecture Notes in Computer Science., Springer-Verlag (1994) 6–15
17. Sellmann, M.: The social golfer problem. (Web site available at <http://www.cs.brown.edu/people/sello/golf.html>)

Colour Reassignment in Tabu Search for the Graph Set T-Colouring Problem

Marco Chiarandini¹, Thomas Stützle², and Kim S. Larsen¹

¹ University of Southern Denmark, IMADA, Odense, Denmark
`marco,kslarsen@imada.sdu.dk`

² Université Libre de Bruxelles, CoDe, IRIDIA, Brussels, Belgium
`stuetzle@ulb.ac.be`

Abstract. The graph set T -colouring problem (GSTCP) is a generalisation of the classical graph colouring problem and it is used to model, for example, the assignment of frequencies in mobile networks. The GSTCP asks for the assignment of sets of nonnegative integers to the vertices of a graph so that constraints on the separation of any two numbers assigned to a single vertex or to adjacent vertices are satisfied and some objective function is optimised. Among the various objective functions of interest, we focus on the minimisation of the span, that is, the difference between the largest and the smallest integers used.

In practical applications large size instances of the GSTCP are to be solved and heuristic algorithms become necessary. In this article, we propose a new hybrid procedure for the solution of the GSTCP that combines a known tabu search algorithm with an algorithm for the enumeration of all feasible re-assignments of colours to a vertex. We compare the new algorithm with the basic tabu search algorithm and for both we study possible variants. The experimental comparison, supported by statistical analysis, establishes that the new hybrid algorithm performs better on a variety of instance classes.

1 Introduction

The *Graph Set T-Colouring Problem* (GSTCP) generalises the concept of graph colouring which is used to model a variety of real-world combinatorial optimisation scenarios [1]. A *set T-colouring* of a graph is an assignment of sets of nonnegative integers (colours) to the vertices of the graph such that (i) every vertex receives exactly the number of colours it requires, (ii) each pair of integers assigned to a single vertex satisfies the vertex separation constraints, and (iii) every pair of numbers assigned to two adjacent vertices satisfies the edge separation constraints. The edge and vertex separation constraints are expressed by sets of integers that represent the forbidden differences between the integers assigned to the vertices. In the special case of the sets being composed of only consecutive integers starting from zero, we speak of *separation distance* constraints and the largest integer in each set suffices to represent them.

There exist various objective functions for the optimisation version of the GSTCP. Common objective functions to be minimised are the *span*, that is, the

difference between the largest and the smallest integers assigned to any vertex, or the *order*, that is, the number of integers effectively used [2,3]. In the frequency assignment application, more often, it is required to minimise the constraint violations by keeping the number of colours fixed. However, in the literature on this problem the focus has been mainly on the minimisation of the span because most solution approaches are suitable also for the needs in frequency assignment. Here we also address the minimal span objective.

The GSTCP is a generalisation of the vertex colouring problem and it arises in the modelling of various real-life problems, the most important being the assignment of frequencies to radio transmitters when designing mobile phone networks. In this case, vertices represent transmitters and colours the frequencies to be assigned to the transmitters subject to certain interference constraints, the *T*-constraints [2]. Other applications arise in traffic phasing and fleet maintenance [4] or in the task assignment problem, where a large task is divided into incompatible subtasks (for example, due to resource conflicts) and the problem is to assign a set of time periods to each subtask so that incompatible subtasks are in different time periods [3].

Due to its practical interest, the GSTCP has received significant attention both in graph theory [3,4,5,6] as well as for its algorithmic solution. The development of solution algorithms has mainly focused on approximate methods because exact methods cannot efficiently solve large size instances, as those arising in frequency assignment [7,8]. Among the approximate algorithms, the overwhelming part of the literature focuses on stochastic local search (SLS) algorithms [9]. Among the first of these approaches, Dorne and Hao apply the Tabu Search method [10]. Further research on the GSTCP has been inspired by the *Computational Challenge on Graph Colouring and its Generalisations* organised by Johnson, Mehrotra and Trick (see <http://mat.gsia.cmu.edu/COLORING02/>).¹ Phan and Skiena devise an SLS algorithm based on swap operations and the Simulated Annealing method within their general-purpose platform *Discrept* [11], while Prestwich proposes a randomised backtracking algorithm [12]. In later research, Lim, Zhang and Zhu designed a Squeaky Wheel algorithm for this problem [13].

In this article, we present a new tabu search algorithm for the GSTCP. This algorithm is very similar to that of Dorne and Hao, but it uses a procedure for the reassignment of all colours to a vertex; this procedure first enumerates all feasible reassignments and then chooses one of these uniformly at random. The randomised choice of the reassignment is done to avoid cycling behavior. We also performed an extensive experimental comparison of the new and known algorithms on GSTCP instances with separation distance constraints. This study is corroborated by statistical analysis and it shows that our new Tabu Search search algorithm performs better than previously known versions on specific GSTCP instance classes.

The paper is organised as follows. Section 2 introduces some formalism, transformations of the problem and the benchmark instances. Section 3 describes the

¹ Note that in this challenge, the separation distance GSTCP is called bandwidth multi-colouring problem.

SLS algorithms that are experimentally compared in Section 4. We end with some concluding remarks in Section 5.

2 Definitions, Transformations and Benchmark Instances

A GSTCP instance is defined by (i) an undirected graph $G = (V, E)$, with V being the set of $n = |V|$ vertices and E being the set of edges, (ii) a set of nonnegative integer numbers (called colours) Γ , (iii) a number $r(v)$ of required colours at each vertex $v \in V$, and (iv) a collection T (called T-set) of nonnegative integers including zero, such that there is an integer t_{uv} for each edge $uv \in E$ and an integer t_u for each vertex $u \in V$ representing the allowed separation distances of colours between and within vertices. The decision version of the GSTCP asks for a mapping $\varphi : V \mapsto \mathcal{P}(\Gamma)$ such that the three groups of constraints

$$|\varphi(v)| = r(v) \quad \forall v \in V \quad (1)$$

$$|x - y| \geq t_u \quad \forall u \in V, \forall x, y \in \varphi(u), x \neq y \quad (2)$$

$$|x - y| \geq t_{uv} \quad \forall uv \in E, \forall x \in \varphi(v), \forall y \in \varphi(u) \quad (3)$$

are satisfied. We call such a multi-valued function φ a *proper set T-colouring*, if all these constraints are satisfied and *improper*, otherwise. The three groups of constraints to be satisfied are called *requirement constraints*, *vertex constraints*, and *edge constraints*, respectively. Various objective functions can be defined for the GSTCP. Here, we consider the span, that is, $\max_{u,v \in V} \{|x - y| : x \in \varphi(u), y \in \varphi(v)\}$, the maximal difference between the colours used. With $\min_{u \in V} \{x : x \in \varphi(v)\}$ fixed to 1, the span corresponds to $k - 1$ if $\Gamma = \{1, 2, \dots, k\}$. Hence, in the optimisation version, we are searching for the minimal span, that is, for the minimal value of k such that a proper set T-colouring exists.

The special case of the GSTCP with $r(v) = 1, \forall v \in V$ is called *graph T-colouring problem*. In the T-colouring problem there are no particular requirement and vertex constraints. Both, graph set T-colouring and T-colouring problems, are obviously in \mathcal{NP} and, thus, \mathcal{NP} -complete because they are generalisations of the *k-colouring problem*.

An instance of the GSTCP problem, defined by a graph G , a T-set T and vertex requirements $r(v), \forall v \in V$, is equivalent to an instance of the T-colouring problem on a graph $G^S(V^S, E^S)$. The graph G^S is obtained from G by creating a vertex u for each requirement of a vertex $v \in V(G)$ so that at the end $|V^S| = \sum_{v \in V} r(v)$. The vertices $u \in V^S$ derived from a vertex $v \in V$ form a clique of order $r(v)$ in which each edge uv receives the distance constraints t_v . Every such vertex is then connected with each vertex of the clique induced by another vertex $w \in V$ if $vw \in E$. The colour separation associated with these edges is t_{vw} . The graph G^S is called *split graph* and there is a bijective correspondence between solutions for G and for G^S .

The set of benchmark instances that we use to test algorithms for the GSTCP is composed by three classes. The first class consists of the instances introduced by Michael Trick for the ‘‘Computational Challenge on graph colouring and its

Table 1. Statistics on the benchmark instances. t_u and t_{uv} , are used to indicate that the range of values differs among vertex and edge constraints. $\bar{\rho}$ and p are the average edge density of the graph.

$ V $	$\bar{\rho}$	r	t_u	t_{uv}	# instances
20, 30, ..., 120	0.1	3	10	4.5	—
			10	10	4.5 11 (GEOMn)
	0.2	3	10	4.5	11 (GEOMna)
			10	10	4.5 11 (GEOMnb)

(a) Random Geometric instances

$ V $	p	r	t	# instances
60	0.1	5	5	10
			10	10
			10	5
	0.5	5	5	10
			10	10
			10	5
	0.9	5	5	10
			10	10
			10	5

(b) New Random Uniform instances

Inst	$\bar{\rho}$	$[r^{min}; r^{max}]$	t_u	$[t_{uv}^{min}; t_{uv}^{max}]$
P1	0.73	[8; 77]	5	[1; 2]
P2	0.49	[8; 77]	5	[1; 2]
P3	0.73	[5; 45]	5	[1; 2]
P4	0.49	[5; 45]	5	[1; 2]
P5	0.73	[20; 20]	5	[1; 2]
P6	0.49	[20; 20]	5	[1; 2]
P7	0.73	[16; 154]	5	[1; 2]
P8	0.73	[8; 77]	5	[1; 2]
P9	0.73	[32; 308]	5	[1; 2]

(c) FAP instances

generalisations;” the second class comprises random uniform graphs; the third class is derived from well known instances from frequency assignment.

Random geometric instances (DIMACS). The graphs are formed by vertices that bijectively correspond to points with coordinates in a $[10,000 \times 10,000]$ square that are generated uniformly at random. Each vertex is connected by an edge to another one, if the points are close enough. Separation distances associated to edges are inversely proportional to the distances between points. Vertex requirements are chosen uniformly from the set $\{1, \dots, r\}$ and vertex separation distances are fixed to 10. The instance size ranges from 20 to 120 vertices. We denote sparse instances as **GEOMn** and denser instances as **GEOMna** and **GEOMnb**. The instances **GEOMnb** have higher requirements per node than **GEOMna**. Statistics of this class of instances are summarised in Table 1a.

Random uniform instances. Uniform graphs are typically identified as G_{np} , where n is the number of vertices and each of the $\binom{n}{2}$ possible edges is present with a probability p . These instances were generated by the algorithm of [14], modified to produce set T -colouring instances. Vertex requirements are chosen uniformly from the set $\{1, \dots, r\}$ and vertex and edge separation distances uniformly from the set $\{1, \dots, t\}$. The values assigned to the parameters and the number of instances are reported in Table 1b. We denote these graphs as **T-G.r.t-n.p**. Note that the instances in [10] were generated in the same manner but due to their size they result in very high computation times, making an extensive experimental study, as reported below, impractical.

Philadelphia instances. These instances are characterised by 21 hexagons representing the cells of a cellular phone network around Philadelphia [15]. For each cell, a demand $r(v)$ is given. In case the mutual distance between the centre of two cells is less than d (normalised by the radius of the cells), it is not allowed to assign the same frequency to both cells. This case is generalised by replacing the reuse distance d by a series of non-increasing values d^0, \dots, d^k . For more on these instances we refer to the FAP web repository.² As in the frequency assignment literature, we denote these instances by P1-P9.

3 SLS Algorithms for the GSTCP

In this section, we describe the main components of the SLS algorithms that we examine: the construction heuristic to generate the initial solution, the local search schemes to improve the candidate solution, and the high-level meta-heuristic to guide the search beyond local optima.

All SLS algorithms that we describe solve the optimisation version of the GSTCP. This is done by starting with some large value of k and successively trying to reduce the number of colours used, which directly minimises also the span. The best solution returned is the minimum value of k for which a proper set T-colouring is found.

As far as the meta-heuristic component is concerned, we restrict ourselves to consider only Tabu Search which is the common choice on the GSTCP [10,13].

3.1 Construction Heuristic

We use a *generalised DSATUR* heuristic for constructing the initial solution. This heuristic arose as the best one in a study reported in [16]. The heuristic works on the split graph and for each colour assignment, first a next vertex is chosen and then this vertex is assigned a colour. Generalised DSATUR chooses the vertex to colour next based on the *saturation degree*, i.e., the number of colours forbidden by the assignment of colours in the adjacent vertices. For every vertex $u \in G^S$ receiving a colour c , the list of forbidden colours of the vertices v adjacent to u is updated with the colours in the interval $(c - t_{uv}, c + t_{uv})$. The order of vertices is recomputed by assigning higher priority to vertices with higher saturation degree. In case of ties, priority is given to vertices with the largest adjusted vertex degree, which is defined as $d(v) = \sum_{u \in V^S, uv \in E^S} t_{uv}$. The application of the generalised DSATUR heuristic generates a proper set T-colouring and, hence, an upper bound k .

3.2 Local Search Schemes

In designing a local search for the GSTCP we may follow different approaches. In a first approach the problem is solved as a sequence of decision problems,

² A. Eisenblätter and A. Koster. “FAP web – A website devoted to frequency assignment”. September 2005. <http://fap.zib.de>. (January 2006).

where for each current value k of available colours a proper set T -colouring is searched. In an alternative approach, the value of k is left free to increase and decrease during the search. A further decision can be taken on the problem representation, *i.e.*, whether to transform or not the problem in a T -colouring problem. The combination of these choices gives rise to different local search schemes. In the following, we describe three promising schemes. They form the basis of the Tabu Search algorithms that are explained later.

Scheme 1: Split Graph, k Fixed. In this case, solutions are represented as complete assignments, *i.e.*, one colour is assigned to each vertex. The advantage of this solution representation is that the requirement constraints of the GSTCP are always satisfied. The evaluation function f is defined as the number of vertex and edge constraints broken and, hence, the goal becomes to minimise f to zero. The neighbourhood in this scheme is defined through the well-known one-exchange neighbourhood, where the solutions s and s' are neighbours if they differ in the colour assignment of one single vertex. Often, it is useful to restrict the neighbourhood examination of one-exchanges to only vertices that are involved in constraint violations. We call this restricted neighbourhood N_E .

Scheme 2: Original Graph, k Fixed. A variant to the first scheme uses the original graph. The main implication of this choice is on the representation of a solution, which is now given naturally by a set of $r(v)$ colours for each vertex $v \in V$. The effective search space may be reduced to only those candidate assignments that satisfy the vertex constraints. Hence, requirement constraints and vertex constraints are always satisfied and the evaluation function needs only to count the number of unsatisfied edge constraints.

The one-exchange neighbourhood N'_E is defined, similarly to N_E , by the collection of single colour changes at a vertex but with the further restriction of maintaining the vertex constraints satisfied. This is reflected in a restriction of the sets of possible new colours in the exchanges.

In addition, the vertex exact colour reassignment neighbourhood N_R is defined, which is composed by the collection of all possible reassignments of colours at one single vertex. We restrict these reassignments to those that satisfy the requirement, vertex and edge constraints acting on that vertex. As such, the application of the operator defined by N_R can be seen as an exact solution to a subproblem, where the assignment of $\{1, \dots, k\}$ colours to one vertex is searched under the condition that no other vertex changes its colour assignment. Clearly, it is possible that, given a current configuration, a reassignment of colours that satisfies all edge constraints at a vertex does not exist. In this case, if only vertices involved in at least one conflict are considered, the neighbourhood N_R is empty. Alternatively a change in the colours of a vertex not involved in any conflict may propagate favourably.

Scheme 3: Split Graph, k Variable. In this scheme, a first solution and an initial k_I is provided by a construction heuristic but the number of colours is left free to vary at run time. As in the first scheme, solutions can be represented as complete colourings. The difference is that solutions can be proper and improper

set T-colourings and that the number of colours is bounded to k_I . An evaluation function to guide the search towards proper colourings and towards colourings with smaller span was defined in [17] as

$$f(s) = k_{max} + k_I \cdot \left(\sum_{uv \in E} I_e(uv) + \sum_{v \in V} I_v(v) \right) + (k_{max} - k_{min}) + \sum_{i=1}^{k_I} I_g(i) \quad (4)$$

where k_{max} is the maximal colour, $I_e(uv)$ and $I_v(v)$ are indicator functions that return one if the corresponding edge or vertex constraint is broken, $k_{max} - k_{min}$ is the span of the colouring, and $I_g(i)$ is an indicator function to determine whether any vertex has colour i ; this last term computes the order. The edge conflicts are weighted by the largest number of colours, thus a solution which reduces the number of violations will always be preferred with respect to those that modify the other terms of the sum. The inclusion of the order in the sum contributes to break ties. The term k_{max} is the least important and contributes to use the first colours, avoiding to move with the same span over and over through the interval $[1, k_I]$. As in Scheme 1, the same one-exchange neighbourhood may be used.

3.3 Vertex Exact Colour Reassignment Neighbourhood

We now describe the procedure for the exploration of the neighbourhood N_R used by the local search of Scheme 2. The effect of this neighbourhood operator is to modify the colour assignment to one vertex such that the requirement, vertex and edge constraints, in which the vertex under concern is involved, are all satisfied. Once a vertex is chosen, a set F is determined comprising the colours which are proper given the edge constraints and the colours assigned to the adjacent vertices. The construction of this set can be done in $\mathcal{O}(|V|k)$ if the usual speed-up techniques known from the graph colouring problem are implemented [18]. If one simply looked for $r(v)$ colours from F such that the vertex constraints are satisfied, this would be easy: it suffices indeed to order the values in F and scan the set once, skipping the values that are not sufficiently distant from the previous ones. Yet, this procedure is deterministic and the search would yield the same reassignment of colours if a vertex is visited twice and nothing has changed in its adjacent vertices. In order to avoid this cycling behaviour some randomisation in the reassignment should be introduced, such that, visiting the vertex a second time, a different configuration is obtained which can be profitably propagated. Implementing this strategy corresponds to determining all subsets of F of size $r(v)$ that satisfy the vertex constraints, and pick one at random. More formally, one can formulate the problem via finding a *Subsequence of length L of H integers with mutual distance not smaller than D* : Given an arbitrary sequence of integers $s = \{s_1, \dots, s_H\}$, find a subsequence $l = \{l_1 \dots l_L\}$ of length L with $l_i \in s$, $\forall i = 1, \dots, L$ and mutual distance not smaller than D , that is, satisfying: $|l_i - l_j| \geq D$, $\forall i, j$. The determination of all desired subsets of F then corresponds to the problem of *enumerating all subsequences of length L of H integers with mutual distance not smaller than D* .

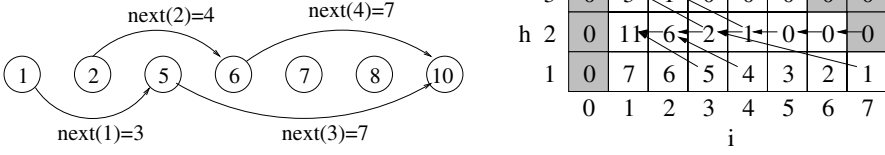


Fig. 1. An example of vertex exact colour reassignment for a case in which $F = \{1, 2, 5, 6, 7, 8, 10\}$, $L = |F| = 7$, $D = t_v = 4$ and $H = r(v) = 3$. On the left is given the vector s of 7 integers. For each integer in the sequence the pointer $next()$ is computed; where it is not indicated, it is set to 0. On the right, we have the table of $N_h[i]$ values. Its construction starts from the low right corner. Arrows indicate the stored values that are used to compute the entries. The grey cells indicate the values without a proper meaning and hence assigned by convention.

We solve the problem of determining all the proper subsets of F in a dynamic-programming fashion by solving subproblems and saving their answers in a table. Given the ordered sequence of integers in F , a proper colouring is an ordered subsequence of integers composed by other subsequences, each allowing a number of proper solutions, corresponding to the different ways the subsequence can be extended to a sequence of length $r(v)$ by adding elements from F . The total number of such solutions can be defined recursively and computed in a bottom-up fashion. Afterwards it is possible to choose one solution randomly by selecting among all the existing solutions.

More specifically, let s be the ordered vector of integers in F , $L = |F|$, $D = t_v$ and $H = r(v)$. Then for each position i of the vector s we define $next(i) = \min_j \{j | j > i, s[j] - s[i] \geq D\}$. For each subsequence l of s , the number $N_H[i]$ of proper subsequences of s of length $h \in \{1, \dots, H\}$ containing $l = \{s[1], \dots, s[i]\}$, can be determined by the recursion

$$N_h[i] = \begin{cases} L - i + 1 & \text{if } h = 1 \\ N_{h-1}[next(i)] + N_h[i + 1] & \text{if } L - i - 1 > h \geq 2 \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

if $i \geq 1$ and $N_h[0] = 0$ by convention.

Hence, the total number of proper subsequences of length $r(v)$ corresponds to $N_H[1]$. To select one solution at random among the $N_h[i]$ solutions one has then simply to scan the sequence s and select each element with probability $N_{h-1}[next(i)]/N_h[i]$, since this is the fraction of extensions that contain the element in question. Whenever an element is chosen, the scanning moves to $next(i)$. Scanning the sequence of numbers takes linear time, but computing each time the recursion 5 takes exponential time. However, this can be done much more efficiently if all values $N_h[i]$ are computed at the beginning and recorded in a table. Referring to Figure 1, right, if the table is filled from bottom to top and from right to left within each row, each new entry needs the values of $N_{h-1}[next(i)]$ and $N_h[i + 1]$ which are already determined and stored. The

next function and the table can be computed in $\mathcal{O}(\max(D, H, \log L)L)$ while for choosing a subsequence randomly a further scan of the sequence s is needed.

3.4 Tabu Search Algorithms

As said, all the SLS algorithms developed use the Tabu Search technique. We give here the details of how Tabu Search is applied in the three local search schemes. Under Scheme 2 we devise three different variants thus giving rise to a total of 5 algorithms that we tested experimentally.

Scheme 1: Split Graph, k Fixed. We use a standard Tabu Search procedure that chooses at each iteration a best non-tabu move or a tabu but “aspired” neighbouring solution from the restricted one-exchange neighbourhood (N_E). The tabu list forbids to reverse a move and the tabu tenure is chosen as $tt = \text{random}(10) + 2\delta|V^c|$, where V^c is the set of vertices which are involved in at least one conflict, δ is a parameter, and $\text{random}(10)$ is an integer random number uniformly distributed in $[0, 10]$; this choice follows that of a successful tabu search algorithm for the graph colouring problem [19]. We denote this algorithm SF-TS. SF-TS is very similar to the Tabu Search algorithms proposed in [20], [21], and [17]. In those papers, Tabu Search was shown to perform better than Simulated Annealing and Genetic Algorithms.

Scheme 2: Original Graph, k Fixed. An application of Tabu Search using the one-exchange neighbourhood N'_E on the original graph was designed by [10]. Other versions of this algorithm for frequency assignment [22] differ only in the management of the tabu length or are more rudimentary [23]. Our version uses the same tabu tenure definition as SF-TS; this results in an algorithm analogous to that proposed in [10]. We denote this algorithm OF-TS.

We also include two enhanced versions of OF-TS that make use of the newly introduced vertex reassignment neighbourhood N_R . Since the exploration of the union of N'_E and N_R would be computationally expensive, we adopt a heuristic rule for choosing the next move to apply. For short, first the best non tabu move in the neighbourhood N'_E is determined. If it improves on the current solution, it is accepted. If it leaves the evaluation function value unchanged or worsens it, a move is searched in the neighbourhood N_R , restricted to vertices involved in at least one conflict. If a proper reassignment is found, it is applied; otherwise the best non-tabu move in N'_E is applied. We call the overall algorithm OF-TS+R.

A variant of OF-TS+R considers a random vertex from V if no move is found in N_R restricted to conflicting vertices. The motivation for this is that a random reassignment of colours to vertices where no conflict is present may produce a change that can propagate profitably. We denote this variant OF-TS+R*.³

The Tabu Search mechanism applied to moves in N'_E is the same used in OF-TS and [10] (aspiration criterion included). No tabu search mechanism is

³ By chance a randomly chosen vertex can happen to be in conflict; in this case the best non tabu move in N_E is chosen as in OF-TS+R. Clearly, this case can be avoided in another implementation.

instead applied to moves in N_R . In this case, repetitions in the search are avoided by the randomisation of the reassignment. This is the reason why preliminary experiments clearly indicated that the use of a randomised reassignment instead of a deterministic one is preferable.

Scheme 3: Split Graph, k Variable. We test a tabu search algorithm based on the same framework as in the neighbourhood N_E , but using the evaluation function of Equation 4. We denote this algorithm as **SV-TS**.

Parameter Settings. SLS algorithms require a number of parameters to be adapted to the class of problem instances under consideration. To accomplish this task we used the racing algorithm of Birattari [24], which is a fully automatic procedure based on sequential testing. In the Tabu Search algorithms introduced the only parameter to be decided is δ . For each of the algorithms we used as candidates the set of numbers $\{0.5; 1; 10; 20; 30; 40; 50; 60; 70; 100\}$ and the best values found were 10 for the uniform, 20 for geometric, and 40 for Philadelphia instances. In the following, for each algorithm we only consider the version with the best set of parameters for the instance class.

4 Experimental Analysis

We evaluate experimentally the five versions of Tabu Search. We maintain the classes of instances separated as we expect to see differences in performance. On the 27 random geometric instances (disconnected graphs were removed) and on the 90 random uniform instances, we collected three runs per algorithm per instance; on each of the nine Philadelphia instances five runs per algorithm.

In a preliminary test we verified on the instances by Dorne and Hao [10] that our re-implementation of **OF-TS** gives results comparable to the earlier published ones [16]. To compare the five algorithms under roughly fair conditions, we imposed a same computation time limit for all of them. This is necessary, since the single iterations of **OF-TS** and **OF-TS+R** have different computation time requirements. To determine a time limit, we run **OF-TS** for $I_{max} = 10^5 \times \sum_{v \in V} r(v)$ iterations. Note that the termination time is a stochastic variable and moreover it varies among instances. We used, therefore, a multiple regression model from the termination times collected in 5 runs per instance to define the time limits. The details of this model are reported in [16]. Here we limit ourselves to report in the tables that follow the time limits adopted which refer to a machine 2GHz AMD Athlon MP 2400 Processor with 256 KB cache and 1 GB RAM, running Debian Linux.

4.1 Tabu Search Comparison

The analysis of results is carried out through simultaneous confidence intervals for multiple comparisons. In particular the Friedman rank-based statistical procedure is used to infer the simultaneous confidence intervals of the average rank

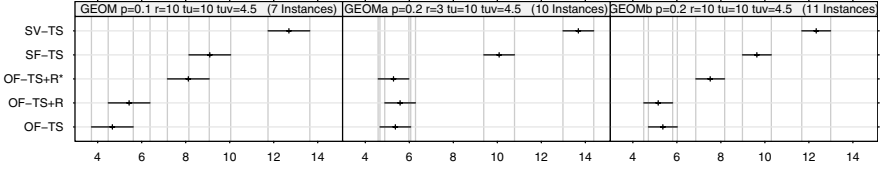


Fig. 2. Multiple comparisons through simultaneous confidence intervals on the random geometric instances. The x -axis indicates the average rank while the confidence intervals are derived from the Friedman test.

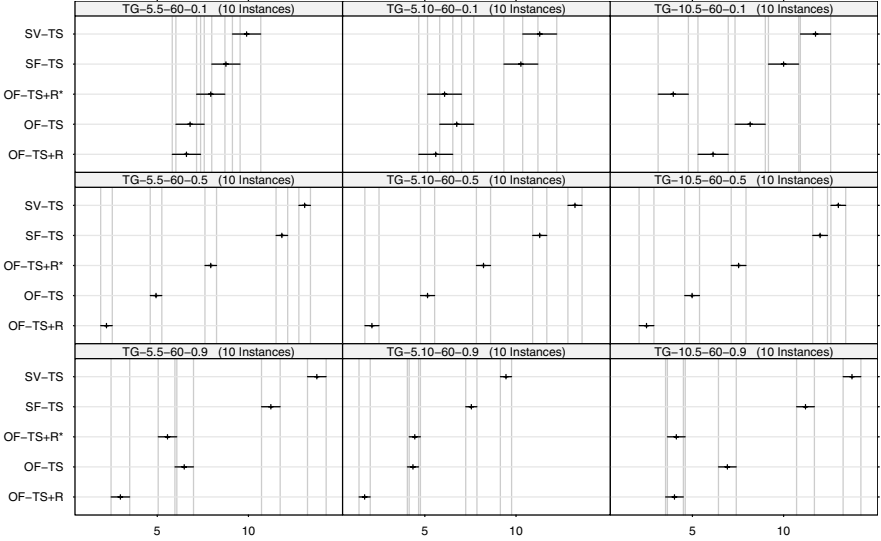


Fig. 3. Confidence intervals for the all pairwise comparisons of SLS algorithms on aggregated uniform random instances of the GSTCP. The x -axis indicates the average rank while the confidence intervals are derived from the Friedman test.

performance of each algorithm [25,26]. In this procedure, each result in terms of colour span is ranked with all other results for the same instance, thus removing the problem of different scale of results among the instances and allowing an aggregate analysis (within a given instance class). We report the results in Figure 2, 3 and 4, where two algorithms are significantly different if the confidence intervals of the corresponding average rank do not overlap. The more the interval is shifted towards the left the better is the algorithm performance.

The first observation is that, as expected, results vary among the instances. On the geometric instances the exact colour reassignment does not introduce any significant improvement and, in fact, it may even worsen the basic OF-TS slightly. These instances are, however, representative of only a restricted portion

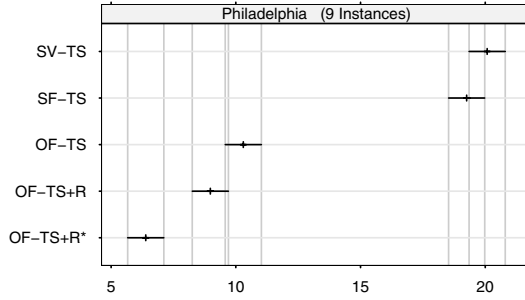


Fig. 4. Confidence intervals for the all pairwise comparisons of SLS algorithms on Philadelphia instances of the GSTCP. The x -axis indicates the average rank while the confidence intervals are derived from the Friedman test.

of the space of all possible instances: they represent only graphs with very small edge density and fixed separation constraints.

More informative in this sense may be the analysis on the random uniform graphs (Figure 3). Here the space of instances is better sampled as graphs with different edge density and vertex requirement are present. This allows to conjecture on performance variations due to instance features. The most important result is that exact colour reassignment becomes worthwhile when the edge density is at least 0.5. In these cases, indeed, the performance of OF-TS+R becomes clearly better than that of OF-TS. The vertex requirements appear also to have an influence as seen by the better performance of OF-TS+R* for high vertex requirements. This latter tendency seems to be confirmed on the Philadelphia instances, where OF-TS+R* outperforms all other versions of tabu search.

Finally a note on SV-TS and SF-TS. Given the much worse performance of these two algorithms compared to the other three, we conclude that the use of the split graph is not a good choice for local search algorithms on the GSTCP. (This seems to be different for constructive heuristics [16].) Additionally, the worse performance of SV-TS compared to SF-TS suggests that the usage of a variable k is not advisable for the GSTCP.

4.2 Comparison to the State-of-the-Art Algorithms

In Tables 2 and 3 we report the numerical results for comparison with previously published results. On the random geometric graphs previous results on the span are due to [11,12,13]. On the Philadelphia instances results on lower bounds and upper bounds are available at <http://fap.zib.de/problems/Philadelphia/>. In all the instances the best upper bounds of approximate algorithms went over time to coincide with the lower bound thus proving optimality for these instances. Note however that no algorithm has solved all the instances alone and we refer to the FAP web repository for a complete list of references for each result (the most robust seems to be the genetic algorithm by [27]). The same reasoning holds for

Table 2. Numerical results on the random geometric instances. Given are the instance identifier, a lower bound, the best solution known so far, the results of our generalised DSATUR, our computation time limits, and the results of our five tabu search algorithms.

Instance	LWB	Best Heur.	DSATUR	max time	OF-TS	OF-TS+R	OF-TS+R*	SF-TS	SV-TS
GEOM60	230	258	258 258	710	258 258	258 258	258 258	258 258	258 258
GEOM70	260	273	283 287.5	1060	269 270	270 271	271 272	270 278	274 285
GEOM80	365	383	392 395	1490	384 385	386 386	388 390	389 389	393 394
GEOM90	313	332	335 338.5	1810	332 333	332 332	335 337	333 334	335 340
GEOM100	378	404	412 416	2170	411 414	412 414	411 411	411 414	414 415
GEOM110	348	383	400 410	2510	383 383	381 382	387 389	381 389	403 404
GEOM120	343	402	412 419	2730	402 404	404 405	404 405	409 417	416 419
GEOM30a	182	209	238 238	380	212 213	212 212	212 212	222 228	234 235
GEOM40a	160	213	229 229	500	214 215	215 216	217 217	220 220	225 226
GEOM50a	199	318	335 345	1080	318 318	319 321	321 322	329 337	337 340
GEOM60a	290	358	369 373	1420	361 361	361 362	362 364	363 363	373 373
GEOM70a	425	469	487 487	1470	484 484	484 484	481 484	483 486	487 487
GEOM80a	241	379	388 396	1510	371 372	371 371	368 369	378 383	391 394
GEOM90a	285	377	398 405	1910	398 401	398 401	389 389	398 402	398 405
GEOM100a	302	459	462 471	2500	444 448	445 446	443 447	454 459	462 465
GEOM110a	385	494	523 523	3120	506 506	504 505	498 500	507 509	516 518
GEOM120a	514	556	571 578	3690	550 556	553 556	558 560	556 558	578 578
GEOM20b	39	44	45 45	30	44 44	44 44	44 44	44 44	44 44
GEOM30b	38	77	78 78	80	77 77	77 77	77 77	77 77	77 77
GEOM40b	74	74	79 86	140	74 74	74 74	74 74	75 75	76 76
GEOM50b	67	87	92 94	200	84 85	85 85	84 85	86 87	87 88
GEOM60b	79	116	123 132	300	120 120	118 119	119 119	120 120	121 122
GEOM70b	94	121	133 135	380	121 121	120 122	122 122	122 123	125 125
GEOM80b	110	141	148 149	490	140 140	139 140	141 142	140 141	140 142
GEOM90b	112	157	160 161	590	150 150	149 149	149 150	152 153	153 155
GEOM100b	133	170	173 179	690	162 163	164 164	162 165	169 170	172 172
GEOM110b	182	206	221 225.5	790	208 209	206 209	213 213	212 213	214 215
GEOM120b	172	199	206 219.5	910	195 198	197 198	201 201	200 201	202 203

Table 3. Numerical results on the Philadelphia instances. Given are the instance identifier, the known optimum, the best heuristic solution of the FASoft system, the results of our generalised DSATUR, our computation time limits, and the results of our five tabu search algorithms.

Instance	OPT	Known Heur.	DSATUR	max time	OF-TS+R*	OF-TS+R	OF-TS	SF-TS	SV-TS
P1	427	448	480 485	490	427 427	427 427	427 427	479 480	480 480
P2	427	476	458 464	712	427 427	427 448	428 459	482 483	484 484
P3	258	285	268 268	333	258 258	258 258	258 258	266 266	262 263
P4	253	269	260 266	225	253 254	253 253	253 253	264 264	264 264
P5	240	251	250 255	327	240 240	240 240	240 240	240 240	240 240
P6	180	231	195 199	279	183 184	185 185	185 186	195 197	195 197
P7	856	895	969 973	2439	856 857	871 877	860 871	967 969	967 969
P8	525	593	539 539	610	525 525	525 525	527 528	548 548	549 549
P9	1714	1801	1938 1946	10381	1715 1717	1756 1758	1755 1759	1938 1938	1938 1938

the construction heuristics whose results we also report for comparison. On both instance classes the results concern the best and median value for the generalised DSATUR heuristic and the Tabu Search algorithms. The lower bounds given on the geometric instances are obtained by the procedure described in [16]. Results are in terms of maximal number of colours used, hence to derive the span one has to subtract one.

On the random geometric graphs our results improve the best known colourings on 11 instances, are worse on 9 instances and reach the same performance on 8 instances. There seems to be therefore no significant difference. However,

such kind of comparison is not reliable as our data are based on aggregated best values from more than one algorithm and more than one run. Nevertheless, the results prove the high quality of the results here discussed.

On the Philadelphia instances OF-TS+R* produces 7 times out of 9 the optimal solution. Only the algorithm by [27] can reach similar performance with 8 out of 9 optimal results. The results of the generalised DSATUR construction heuristic are worse than those attained by a portfolio of heuristics implemented in the system FASoft [17]. However a comparison with the results of [10] shows that our generalised DSATUR gives much better results than those reported for their implementation of DSATUR. No result for construction heuristics is instead reported in the literature on the random geometric instances.

5 Summary

In this article, we studied a hybrid Tabu Search algorithm for the GSTCP problem. This algorithm uses a canonical tabu search algorithm based on a one-exchange neighbourhood operator and enhances it by an exact reassignment of colours to vertices when opportune. The exact reassignment would be trivial if implemented in a deterministic manner but it would yield a cycling behaviour in the search. Instead we proposed an algorithm which is capable of returning a random exact colour reassignment. This feature favours the diversification of the search, which is often a key mechanism for making SLS algorithms successful. A major contribution of this article is, hence, the recursive formula with polynomial time-complexity for the random reassignment.

Another contribution is the experimental analysis of various tabu search algorithms on 3 classes of instances. The results of this comparisons are that (i) Tabu Search algorithms working on a split-graph representation are less efficient than the tabu search algorithms working on the “natural” problem representation, (ii) on instances characterised by a low edge density in the graph and low vertex requirements, the occasional exact reassignment does not improve the underlying tabu search algorithms, (iii) on the other instances and above all on the Philadelphia instances, which stem from the literature on frequency assignment, the hybrid algorithm performs, often by quite a large margin, better than the basic tabu search algorithms.

Acknowledgements. Thomas Stützle acknowledges support of the Belgian FNRS, of which he is a research associate. Kim S. Larsen was supported in part by the Danish Natural Science Research Council (SNF).

References

1. Jensen, T.R., Toft, B.: Graph coloring problems. Wiley Interscience, New York, USA (1995)
2. Hale, W.K.: Frequency assignment: Theory and applications. *Proceedings of the IEEE* **68** (1980) 1497–1514

3. Tesman, B.A.: Set T -colorings. *Congressus Numerantium* **77** (1990) 229–242
4. Roberts, F.S.: T -colorings of graphs: Recent results and open problems. *Discrete Mathematics* **93** (1991) 229–245
5. Tesman, B.A.: List T -colorings. *Discrete Applied Mathematics* **45** (1993) 277–289
6. Giaro, K., Janczewski, R., Malafiejski, M.: The complexity of the T -coloring problem for graphs with small degree. *Discrete Applied Mathematics* **129** (2003) 361–369
7. Eisenblätter, A., Grötschel, M., Koster, A.M.C.A.: Frequency assignment and ramifications of coloring. *Discussiones Mathematicae Graph Theory* **22** (2002) 51–88
8. Aardal, K.I., van Hoesel, C.P.M., Koster, A.M.C.A., Mannino, C., Sassano, A.: Models and solution techniques for the frequency assignment problem. ZIB-report 01–40, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Germany (2001)
9. Hoos, H., Stützle, T.: *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann Publishers, San Francisco, CA, USA (2004)
10. Dorne, R., Hao, J.: Tabu search for graph coloring, T -colorings and set T -colorings. In: *Meta-heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Kluwer Academic Publishers (1998) 77–92
11. Phan, V., Skiena, S.: Coloring graphs with a general heuristic search engine. In Johnson, D.S., Mehrotra, A., Trick, M., eds.: *Proceedings of the Computational Symposium on Graph Coloring and its Generalizations*, Ithaca, New York, USA (2002) 92–99
12. Prestwich, S.: Hybrid local search on two multicolouring models. In: *International Symposium on Mathematical Programming*, Copenhagen, Denmark (2003)
13. Lim, A., Zhu, Y., Lou, Q., Rodrigues, B.: Heuristic methods for graph coloring problems. In: *SAC '05: Proceedings of the 2005 ACM Symposium on Applied Computing*, New York, NY, USA, ACM Press (2005) 933–939
14. Culberson, J., Beacham, A., Papp, D.: Hiding our colors. In: *Proceedings of the CP'95 Workshop on Studying and Solving Really Hard Problems*, Cassis, France (1995) 31–42
15. Anderson, L.G.: A simulation study of some dynamic channel assignment algorithms in a high capacity mobile telecommunications system. *IEEE Transactions on Communications* **21** (1973) 1294–1301
16. Chiarandini, M.: *Stochastic Local Search Methods for Highly Constrained Combinatorial Optimisation Problems*. PhD thesis, Computer Science Department, Darmstadt University of Technology, Darmstadt, Germany (2005)
17. Hurley, S., Smith, D.H., Thiel, S.U.: FASoft: A system for discrete channel frequency assignment. *Radio Science* **32** (1997) 1921–1939
18. Fleurent, C., Ferland, J.: Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research* **63** (1996) 437–464
19. Galinier, P., Hao, J.: Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization* **3** (1999) 379–397
20. Costa, D.: On the use of some known methods for T -colorings of graphs. *Annals of Operations Research* **41** (1993) 343–358
21. Castelino, D., Hurley, S., Stephens, N.: A tabu search algorithm for frequency assignment. *Annals of Operations Research* **63** (1996) 301–320
22. Hao, J.K., Dorne, R., Galinier, P.: Tabu search for frequency assignment in mobile radio networks. *Journal of Heuristics* **4** (1998) 47–62
23. Hao, J.K., Perrier, L.: Tabu search for the frequency assignment problem in cellular radio networks. Technical Report LGI2P, EMA-EERIE, Parc Scientifique Georges Besse, Nîmes, France (1999)

24. Birattari, M.: The `race` package for R. Racing methods for the selection of the best. Technical Report TR/IRIDIA/2003-37, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium (2003)
25. Conover, W.: Practical Nonparametric Statistics. third edn. John Wiley & Sons, New York, NY, USA (1999)
26. Chiarandini, M., Basso, D., Stützle, T.: Statistical methods for the comparison of stochastic optimizers. In Doerner, K.F., et al., eds.: MIC2005: The Sixth Metaheuristics International Conference, Vienna, Austria (2005) 189–196
27. Matsui, S., Tokoro, K.: Improving the performance of a genetic algorithm for minimum span frequency assignment problem with an adaptive mutation rate and a new initialization method. In: Proc. of GECCO-2001 (Genetic and Evolutionary Computation Conference), Morgan Kaufmann Publishers (2001) 1359–1366

Investigation of One-Go Evolution Strategy/Quasi-Newton Hybridizations

Thomas Bartz-Beielstein*, Mike Preuss, and Günter Rudolph

Dortmund University, 44221 Dortmund, Germany

Thomas.Bartz-Beielstein@udo.edu

<http://ls11-www.cs.uni-dortmund.de/people/>

Abstract. It is general knowledge that hybrid approaches can improve the performance of search heuristics. The first phase, exploration, should detect regions of good solutions, whereas the second phase, exploitation, shall tune these solutions locally. Therefore a combination (hybridization) of global and local optimization techniques is recommended. Although plausible at the first sight, it remains unclear how to implement the hybridization, e.g., to distribute the resources, i.e., number of function evaluations or CPU time, to the global and local search optimization algorithm. This budget allocation becomes important if the available resources are very limited. We present an approach to analyze hybridization in this case. An evolution strategy and a quasi-Newton method are combined and tested on standard test functions.

1 Introduction

Hybridizing *evolutionary algorithms* (EA) with *local search* techniques (LS) is not exactly a new idea. In fact, several such approaches exist (e.g. genetic local search, hybrid genetic algorithms) and nowadays, they are subsumed under the term *memetic algorithms* (MA) that was invented by [13]. A recent overview is given by [10], together with a suggested taxonomy.

However, MA usually do not treat EA and LS as coequal techniques. Instead, the local search methods are integrated into the evolutionary algorithm framework. This is straightforward as EAs are considered to have global search capabilities whereas LSs are prone to get stuck in the first local optimum they approach. Nevertheless, we follow a different path by simply applying an *evolution strategy* (ES) and a *quasi-Newton* (QN) method consecutively, without any other information exchange besides communicating the best solution found by the former into the latter. This scheme resembles the simplest possible of such combinations, thereby implying that the EA is able to detect a region near the global optimizer in one go which is then approximated by the local search method. The working hypothesis of commonly used MA differs insofar as the evolutionary algorithm is only required to step into the vicinity of *any* (possibly local) optimizer before the LS method takes over. In stark contrast to the situation investigated here, MAs mostly apply both techniques several times.

* Corresponding author.

The main motivation for hybridizing ES and QN by simply applying them consecutively only once is drawn from two sources:

- Combining a global and a local search technique is expected to result in better performance than either of them alone, especially if at least one of the techniques can be tailored to deliver what the other needs to proceed.
- Real-world applications as e.g. design problems enforce short runs: The available time poses hard limits onto the allowed number of evaluations, often less than 10^4 can be afforded. Consequently, frequent switching between global and local search techniques may be inappropriate for such problems.

In recent research, we observe two contradictory trends: (i) to develop more and more new algorithms or (ii) to analyze and understand existing heuristics and to add new features only when necessary. With this work, we lean against the second trend by taking two existing algorithms and combining them in a very simple fashion. However, we do not add new features but rather try to *adapt* an EA to work well in combination with a QN-algorithm by tuning its parameters and adjusting the fraction of shared resources it is allowed to consume.

The paper is organized as follows: Section 2 introduces the algorithms that will be hybridized: an evolution strategy and a QN-method, followed by a brief description of the resulting hybrid algorithm. The experimental methodology is introduced in Sect. 3. It relies on the sequential parameter optimization approach, which has been applied to several optimization tasks from industrial optimization and theoretical computer science. Experiments are presented in Sect. 4. Our focus lies on optimization problems with limited resources. Section 5 analyzes the experimental results, and Sect. 6 summarizes the conclusions drawn from this study.

2 Algorithms

2.1 Evolution Strategies

An ES-algorithm run may be characterized as follows: The parental population is *initialized* at time (generation) $g = 0$. Then λ offspring individuals are generated in the following manner: For each offspring individual, a parent family of size ρ is selected randomly from the parent population. *Recombination* is applied to the object variables and the strategy parameters. The mutation operator is applied to the resulting offspring vector. After evaluation, the next parent population is determined by means of a *selection* procedure. The populations created in the iterations of the algorithm are called *generations* or *reproduction cycles*. Unless a termination criterion is fulfilled, the generation counter (g) is incremented and the process continues with the generation of the next offspring. We consider the parameters or control variables from Table 1. This table shows typical parameter settings. Bäck does not recommend using “standard” without reflection. Considering the no-free lunch debate and current results from experimental research, it is obvious that problems exist where these “standards” fail. Thus it is necessary to adjust the parameters to the specific optimization problem. SPO,

Table 1. Default settings of exogenous parameters of a “standard” evolution strategy [1]. The ES parameters can be described as follows: The symbols μ and λ denote parent and offspring population sizes, respectively. The offspring-parent ratio is defined as $\nu = \lambda/\mu$, $\sigma^{(0)}$ denotes the initial standard deviation, which is used for mutation. Let d be the problem dimension. Then between $n_\sigma = 1$ and d different standard deviations can be used. c_0 and c_1 denote multipliers for the global and local learning rates, respectively, as described in Equation (27) in [4]. Note, [4] use the same c , i.e., $c_1 = c_2$ for global and local learning rates. The parameter ρ describes the number of parent individuals used in recombination and r_d and r_i denote discrete and intermediary recombination, respectively. Intermediate recombination has been used for both object and strategy parameters in our experiments. The symbol κ is the maximum lifespan of an individual, the so-called comma strategies use $\kappa = 1$, whereas plus strategies use $\kappa = \infty$. Parameters, that are tuned, are printed in *boldface*.

Symbol	Parameter	Range	Default
μ	Number of parent individuals	\mathbb{N}	15
ν	Offspring-parent ratio	\mathbb{R}_+	7
$\sigma_i^{(0)}$	Initial standard deviations	\mathbb{R}_+	3
n_σ	Number of standard deviations	$\{1, 2, \dots, d\}$	1
c_0	Multiplier for the global learning rate	\mathbb{R}_+	1
c_1	Multiplier for the local learning rate	\mathbb{R}_+	1
ρ	Mixing number	$\{1, 2, \dots, \mu\}$	2
r_x	Recombination operator for object variables	$\{r_i, r_d\}$	r_d
r_σ	Recombination operator for strategy variables	$\{r_i, r_d\}$	r_i
κ	Maximum age	\mathbb{N}	1

as described in Sect. 3.2, provides one possible technique to avoid poor results caused by wrongly specified parameters. The reader is referred to [2] and [4] for detailed descriptions of these parameters.

2.2 Quasi-Newton Methods

The variable metric method utilized for the experiments in this study is a QN-method. Quasi-Newton methods build up curvature information. Let H denote the Hessian, c a constant vector, and b a constant, then a quadratic model problem formulation of the form $\min_x \frac{1}{2}x^T Hx + c^T x + b$ is constructed. If the partial derivatives of x go to zero, i.e., $\nabla f(x^*) = Hx^* + c = 0$, the optimal solution for the quadratic problem occurs. Hence $x^* = -H^{-1}c$. Quasi-Newton methods avoid the numerical computation of the inverse Hessian H^{-1} by using information from function values and gradients. The MATLAB function `fminunc` uses the formula of [5], [7], [8], and [18] to approximate H^{-1} .

2.3 ES/QN-Hybrid

The ESQN algorithm combines the ES and QN by running them consecutively and initializing the latter with the result of the former. The parameter ES2QN distributes the available resources to the algorithms, i.e., it defines the percentage

of function evaluations for the ES: $\text{ES2QN} \in [0.0, 1.0]$. The remainder is assigned to the QN-strategy. For example, if ES2QN is 0, the ES receives no function evaluation and the ESQN is a canonical QN-method. Allotting more time for the ES cuts down resources available to the QN-algorithm and vice versa.

3 Experimental Methodology

3.1 Problem and Algorithm Designs

The concept of *experimental designs* is crucial for our approach. On the one hand, search heuristics such as the Nelder-Mead simplex strategy, genetic algorithms, or particle swarm optimization require the specification of *exogenous* parameters before the algorithm is started. On the other hand, *endogenous* parameters can evolve during the optimization process, e.g., in self-adaptive evolution strategies. We will consider exogenous parameters in the following. By varying the values of the exogenous parameters the experimenter can get some insight into the behavior of an algorithm. This procedure can be described as *active experimentation* in contrast to *passive experimentation*, where the experimenter only observes some phenomena. Passive experimentation predominated experimental research in evolutionary computation until recently. Nowadays, more and more active experimental approaches are developed.

Exogenous parameters will be referred to as *design variables* in the context of statistical design and analysis of experiments. The parameter values chosen for the experiments constitute an algorithm design X_A . Let \mathcal{D}_A denote the set of all possible parameter settings for one algorithm. A *design point* $x_a \in \mathcal{D}_A$ represents one specific parameter setting. *Algorithm tuning can be understood as the process of finding the optimal design point $x_a^* \in \mathcal{D}_A$ for a given problem design X_P .* Tuning leads to results that are tailored for one specific algorithm-optimization problem combination. To discuss the behavior of an algorithm the underlying problem has to be taken into account. A problem being GA easy may be ES hard, and vice versa. Tuning enables a fair comparison of two or more algorithms that should be performed prior to their comparison. This should provide an equivalent budget—for example, a number of function evaluations or an overall run time—for each algorithm.

It is crucial to formulate the goal of the tuning experiments precisely, because in many real-world situations, it is not possible or not desired to find the optimum. A good solution, i.e., a robust solution, is often preferred. This discussion is also relevant for the specification of *performance measures* (PM) in evolutionary computation. There are many different measures for the goodness of an algorithm, i.e., the quality of the best solution, the percentage of runs terminated successfully, or the number of iterations required to obtain the results.

3.2 Sequential Parameter Optimization

Sequential parameter optimization (SPO) is a methodology for the experimental analysis of optimization algorithms to determine improved algorithm de-

Algorithm 1. Sequential parameter optimization

```

1: procedure SPO( $\mathcal{D}_A, \mathcal{D}_P$ )                                ▷ Algorithm und problem design
2:   Select  $p \in \mathcal{D}_P$  and set  $t = 0$                                 ▷ Select problem instance
3:    $X_A(t) = \{x^{(1)}, x^{(2)}, \dots, x^{(k)}\}$                 ▷ Sample  $k$  initial points, e.g., LHS
4:   repeat
5:      $y_j^{(i)} = Y_j(x^{(i)}, p) \forall x^{(i)} \in X_A(t)$  and  $j = 1, \dots, r(t)$     ▷ Fitness evaluation
6:      $\bar{Y}^{(i)}(t) = \sum_{j=1}^{r(t)} y_j^{(i)}(t) / r(t)$                 ▷ Sample statistic for the  $i$ th design point
7:      $x_b$  with  $b = \arg \min_i (\bar{y}^{(i)})$                                 ▷ Determine best point
8:      $Y(\cdot) = \mathcal{F}(\beta, \cdot) + Z(\cdot)$                                 ▷ DACE model
9:      $X_S = \{x^{(k+1)}, \dots, x^{(k+s)}\}$                         ▷ Generate  $s$  sample points,  $s \gg k$ 
10:     $y(x^{(i)}), i = 1, \dots, k + s$                                 ▷ Predict fitness from the DACE model
11:     $I(x^{(i)})$  for  $i = 1, \dots, s + k$                 ▷ Determine expected improvement, cf. [17]
12:     $X_A(t + 1) = X_A(t) \cup \{x^{(k+i)}\}_{i=1}^m$     ▷ Add  $m$  points with the highest  $I(\cdot)$ 
13:    if  $x_b(t) = x_b(t + 1)$  then
14:       $r(t + 1) = 2r(t)$                                 ▷ Increase number of repeats
15:    end if
16:     $t = t + 1; k = k + m$                                 ▷ Increment counters
17:  until Budget exhausted
18: end procedure

```

signs and to learn, how the algorithm works. It employs computational statistic methods to investigate the interactions among optimization problems, algorithms, and environments. We consider each algorithm design with associated output as a realization of a stochastic process and use interpolation method to predict unknown values. Our presentation follows concepts introduced in [16], [9], and [11].

Consider a set of m design points $x = \{x^{(1)}, \dots, x^{(k)}\}$ with $x^{(i)} \in \mathbb{R}^d$. In the *design and analysis of computer experiments* (DACE) *stochastic process model*, a deterministic function is evaluated at these design points. The vector of the k responses is denoted as $y = (y^{(1)}, \dots, y^{(k)})$ with $y^{(i)} \in \mathbb{R}$. The process model proposed in [16] expresses the deterministic response $y(x^{(i)})$ for a d -dimensional input $x^{(i)}$ as a realization of a regression model \mathcal{F} and a stochastic process Z . Algorithm 1. describes the SPO in a formal manner. The selection of a suitable problem instance is done in the pre-experimental planning phase to avoid floor and ceiling effects (1.2). Latin hypercube sampling can be used to determine an initial set of design points (1.3). After the algorithm has been run with these k initial parameter settings (1.5), the DACE process model is used to discover promising design points (1.10). Note that other sample statistics than the mean, e.g., the median, can be used in 1.6. The m points with the highest expected improvement are added to the set of design points, where m should be small compared to s . The update rule for the number of reevaluations $r(t)$ (1.13-15) guarantees that the new best design point $x_b(t + 1)$ has been evaluated at least as many times as the previous best design point $x_b(t)$. Obviously, this is a very simple update rule and more elaborate rules are possible. Other termination criteria exist besides the budget based termination (1.17). Figure 1 illustrates

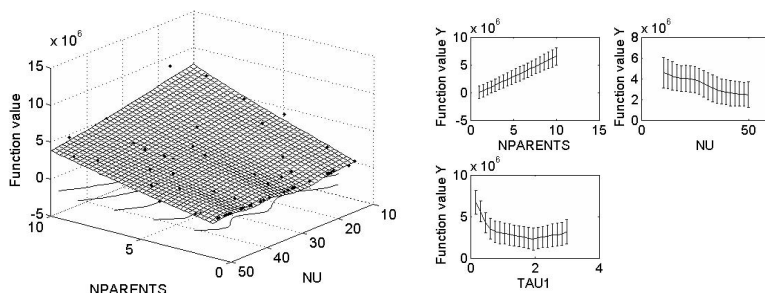


Fig. 1. Typical results from the sequential parameter optimization of the ES. *Left:* Interactions between population size and selective pressure *Right:* Plots of the single effects.

a typical situation from SPO. Here, small population sizes and high selective pressures are beneficial.

A toolbox that implements the sequential parameter optimization is available under the following link: <http://www.springer.com/3-540-32026-1>. Additional material, e.g., the implementation of the evolution strategy used in the following experiments can be downloaded, too. Furthermore, we will provide interfaces to SPO for commonly used search heuristics such as particle swarm optimization, genetic algorithms, or commercial optimization-software packages.

4 Experiments

4.1 Problem Design

We decided to use the deterministic initialization scheme DETEQ, see [3]. It uses one single starting point, i.e., x_0 , so that the same initial conditions are used by both algorithms and the hybrid approach.¹ This is a disadvantage for the population based ES because it is forced to spread search points of its initial population (by applying the mutation operator) within a tight cloud around the starting point, rather than distributing them throughout the whole search space. However, our main focus does not lie on a direct comparison of the algorithms, but on the effect of the hybridization.

To check for floor and ceiling effects, the number of function evaluations was varied during the pre-experimental planning phase. This ensures that the problem design is not too easy or too hard for the algorithms under consideration. Floor and ceiling effects are discussed in [6,3]. To enable a fair comparison, we have chosen t_{\max} , i.e., the maximum number of function evaluations, as $100 \times$ problem dimension. This value appears to be very small ES, because ES need

¹ Note, x_0 should not be confused with $x^{(0)}$ defined in Algorithm 1. The former describes the starting point for one algorithm run, the latter is one parameter set of the optimization algorithm.

a certain amount of function values to adapt they step sizes. However, our experiments reveal some interesting insight into the ES performance that might correct some typical prejudices against ES.

4.2 Algorithm Designs

A suitable algorithm design has to be determined. Clearly, for this specific situation, “standard” parameter settings from the literature are not adequate. Therefore, SPO was used to detect suitable algorithm designs for the ES. Due to the small number of function evaluations, population sizes between 1 and 10 individuals have been used. The selective pressure was chosen from the interval $[0, 10]$. The region of interest for the learning parameters c_0 and c_1 was defined as the interval $[0.1, 3]$. The related ES algorithm designs for the selected functions from [12] are summarized in Table 3. Note, that the parameters from the tuned algorithms show no directly observable patterns, so that no general recommendations can be given for an ES algorithm design that works equally well on every function from the [12] test set.²

4.3 Experiments on Moré’s Test Problems

Due to the limited space, function definitions are omitted. The reader is referred to [12] and [14] for a full description of these functions. We have included some plots to illustrate some characteristics.

Rosenrock. The Rosenbrock function is the first function from the collection described in [12] [15]. Minimum $x^* = (1, 1)$. Optimum $f^* = 0$. Starting point $x_0 = (-1.2, 1)$. This is the famous two-dimensional “banana valley” function.

Experiments with the canonical ES and QN for Rosenbrock’s function showed, that QN and ES are able to solve this problem in principle. Now we will tackle the central question from this paper: does it pay to hybridize ES and QN? Therefore, we have generated a series of ES2QN plots, e.g., in Fig. 2 (right). These plots enable a direct comparison of the canonical ES and QN algorithms: if ES2QN is 0 (0 % ES, but 100% QN), the average performance for $n = 10$ runs of the QN algorithm is shown. If ES2QN is 1 (100% ES), the performance of the ES can be seen. Intermediate ES2QN values, i.e., $\text{ESQN} \in]0, 1[$, show the performance of hybrid approaches. In addition to the mean value from ten runs, the minimum, maximum, and the **bestof** function values are plotted, because they have a great practical relevance. The **bestof** value is determined from n values as follows: determine the minimum value from m random draws (with replacement) out of n ($m < n$). This procedure is repeated very often, say 1,000,000 times, and the average value is reported. The **bestof** value is larger than the minimum, but smaller than the mean value. We have chosen $m = 5$, because 5 repeated runs represent a realistic situation in many real world optimization scenarios. We did not show plots with error bars (or confidence intervals), because for our

² The QN-method was not tuned, because MATLAB does not provide any interfaces to adjust exogenous parameters.

Table 2. Problem designs for the experiments performed on the [12] test suite. The DETEQ initialization method, the EXH termination criterion, and $n = 10$ repeats are used for all experiments. The experiment’s name, the maximum number of function evaluations t_{\max} , the problem’s dimension d , the starting point x_0 for the initialization of the object variables are reported.

Problem design	t_{\max}	d	x_0
$x_{\text{rosen}}^{(1)}$	200	2	$(-1.2, 1)$
$x_{\text{froth}}^{(1)}$	200	2	$(0.5, -2)$
$x_{\text{bscp}}^{(1)}$	200	2	$(0, 1)$
$x_{\text{bscb}}^{(1)}$	200	2	$(1, 1)$
$x_{\text{jensam}}^{(1)}$	200	2	$(0.3, 0.4)$
$x_{\text{osborne2}}^{(1)}$	200	11	see [12]
$x_{\text{meyer}}^{(1)}$	300	3	$(0.02, 4000, 250)$

Table 3. ES algorithm designs. Further ES parameters remained constant as described in Sect. 2.1. Rosenbrock: $x_{\text{ES}}^{(1)}$, Freudenstein and Roth: $x_{\text{ES}}^{(2)}$, Powell badly scaled: $x_{\text{ES}}^{(3)}$, Brown badly scaled: $x_{\text{ES}}^{(4)}$, Jenrich and Sampson: $x_{\text{ES}}^{(5)}$, Meyer: $x_{\text{ES}}^{(6)}$, and Osborne 2: $x_{\text{ES}}^{(7)}$.

Algo. design	μ	ν	c_1	c_2
$x_{\text{ES}}^{(1)}$	1	1.5646	0.315154	0.102151
$x_{\text{ES}}^{(2)}$	1	4.35957	0.215921	2.10074
$x_{\text{ES}}^{(3)}$	6	1.09798	2.93576	2.94653
$x_{\text{ES}}^{(4)}$	4	2.29763	1.66219	2.90905
$x_{\text{ES}}^{(5)}$	8	4.70181	1.87773	0.27439
$x_{\text{ES}}^{(6)}$	9	1.239	0.792342	1.93755
$x_{\text{ES}}^{(7)}$	2	6.94046	1.71284	0.537968

purpose, the mean, min, max, **bestof** (MMMB plots) provide more information. A comparison of both representations is shown in Fig. 7.

Figure 2 clearly indicates that QN outperforms ES and that hybridization worsens the performance for this setting. This result is in accordance with results reported in [14], where the QN algorithm reached a function value of $1.15e - 10$ with 150 function evaluations only.

Freudenstein and Roth. Minimum $x^* = (5, 4)$. Optimum $f^* = 0$. Starting point $x_0 = (0.5, -2)$. This is function 2 from the [12] test set. Figure 3 shows a 3 dimensional and contour plot. It indicates that hybridization is beneficial and that ES performs slightly better than QN. The ES generates solution candidates that “jump over the saddle ($y \equiv 2$)”, and QN can fine tune these solutions. Best results are obtained if approximately 3/4 of the budget is assigned to the ES.

Powell Badly Scaled. Minimum $x^* = (1.098 \dots 10^{-5}, 9.106 \dots)$. Optimum $f^* = 0$. Starting point $x_0 = (0, 1)$. This is function 3 from the [12] test set. Figure 4 shows a 3 dimensional and contour plot. This plot illustrates that

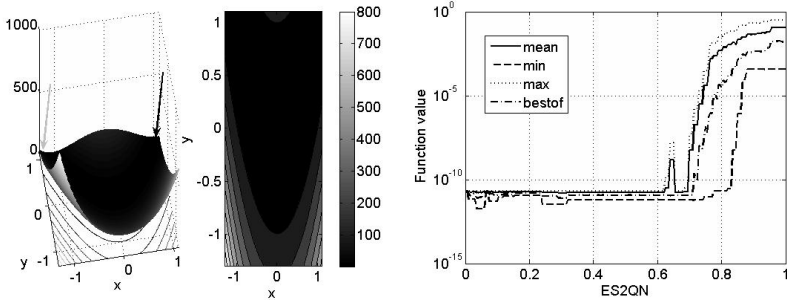


Fig. 2. Rosenbrock. *Left:* The gray arrow depicts the starting point $x_0 = (-1.2, 1)$, the black arrow the optimizer $x^* = (1, 1)$. *Right:* Results from the hybridization ($n = 10$ repeats, this value was used in the following plots, too) clearly demonstrate that hybridization does not improve the algorithm, because QN outperforms ES.

comparing mean values alone tells not the whole story. Hybridization can improve the performance, but this is not guaranteed. However, it might be a good strategy, if the user can select the best result from several runs (as modeled in the performance measure *bestof*). SPO proposed a 6+6-ES, see Table 3. The following situation could be observed in some runs: The ES was able to detect values close to the optimizer after 3 generations, which could be improved by QN.

Brown Badly Scaled. Minimum $x^* = (10^6, 2 \cdot 10^{-6})$. Optimum $f^* = 0$. Starting point $x_0 = (1, 1)$. This is function 4 from the [12] test set. Figure 5 shows a 3 dimensional and contour plot. A first look at the results leads to the conclusion that QN performs better than the ES, cf. the right graph in the first row. However, this result depends heavily on the number of available function evaluations, i.e., t_{\max} . If t_{\max} is increased, ES performs better than QN. Hybridization has no positive effect, it is better to use the canonical algorithms. The ES needs some time adapting the step width, but was able to detect the minimizer. QN

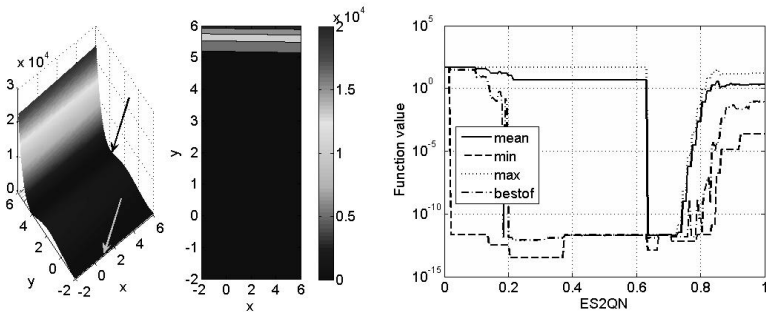


Fig. 3. Freudenstein and Roth. *Left:* The gray arrow depicts the starting point $x_0 = (0.5, 2)$, the black arrow the optimizer $x^* = (5, 4)$. *Right:* Experimental results indicate that hybridization can improve the algorithm's performance.

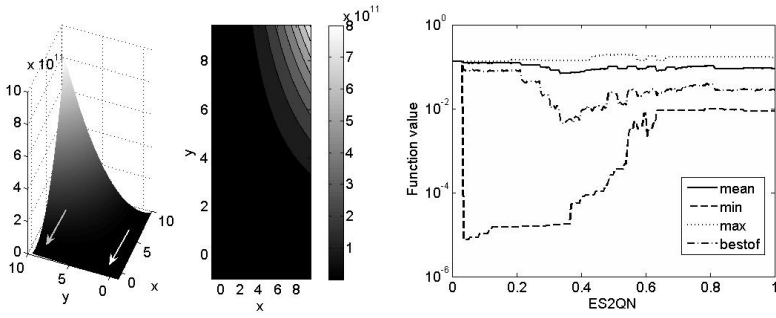


Fig. 4. Powell Badly Scaled. *Left:* The white arrow depicts the starting point $x_0 = (0, 1)$, the gray arrow the optimizer $x^* = (1.098 \dots 10^{-5}, 9.106 \dots)$. *Right:* Results from the hybridization.

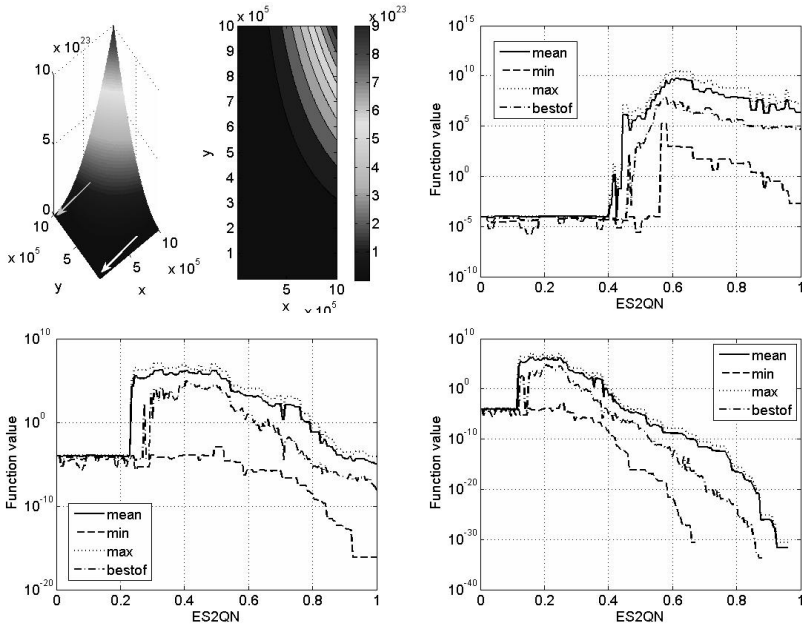


Fig. 5. Brown Badly Scaled. *First row, left:* The white arrow depicts the starting point $x_0 = (1, 1)$, the gray arrow the optimizer $x^* = (10^6, 2 \cdot 10^{-6})$. *Right:* Results from the hybridization, $t_{\max} = 200$. *Second row, left:* $t_{\max} = 400$, *right:* $t_{\max} = 800$. Some curves end abruptly, because the plotted values are zero, which is the known minimum.

finds suboptimal solutions with fewer function evaluations. QN could not detect the optimizer, even if t_{\max} was increased as can be seen from the graphs in the second row of Fig. 5.

Jenrich and Sampson. Minimum $x^* = (0.2578 \dots, 0.2578 \dots)$. Optimum $f^* = 124.362 \dots$. Starting point $x_0 = (0.3, 0.4)$. This is function 6 from the [12] test

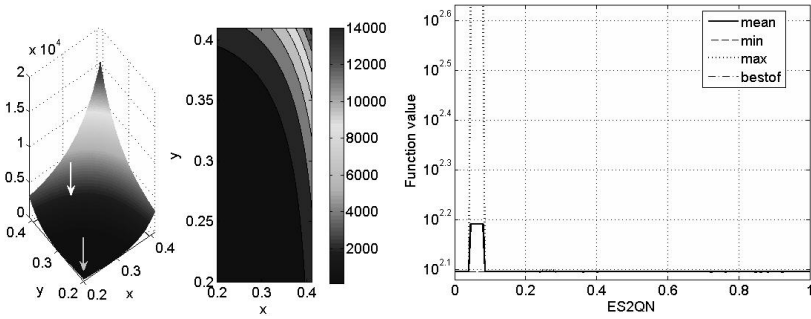


Fig. 6. Jenrich and Sampson. *Left:* The white arrow depicts the starting point $x_0 = (0.3, 0.4)$, the gray arrow the optimizer $x^* = (0.2578 \dots, 0.2578 \dots)$. *Right:* Results from the hybridization illustrate that hybridization worsens the algorithm’s performance.

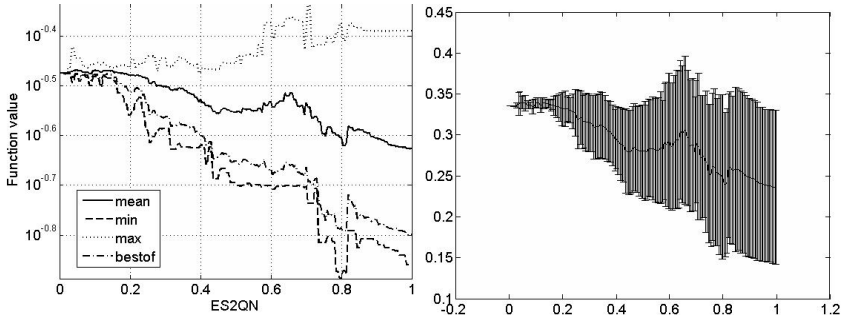


Fig. 7. Osborne 2. *Left:* Results from the hybridization. *Right:* Results from the hybridization illustrate that hybridization does not improve the algorithm’s performance.

set. Figure 6 shows a 3 dimensional plot and contour plot. Both algorithms perform equally well, there is no benefit in hybridization. Hybridization worsens the performance in some settings, see Fig. 6.

Osborne 2. Osborne 2 was included into the test function set, because it the 11-dimensional function. The test suite from [12] contains 6 two, three, and four dimensional, 1 five, six, and nine dimensional, 9 ten dimensional, and 1 eleven dimensional test function. [14] reports some results from optimization attempts with the MATLAB optimization toolbox: This 11 dimensional problem could not be solved by MATLAB’s BFGS without supplying gradient information. And, even with gradient information, more than 10,000 function evaluations were required for finding a point in the vicinity of the global optimizer.³ Osborne 2 is function 19 from the [12] test set. Figure 7 nicely illustrates the trade off between deterministic (QN) and stochastic (ES) search algorithms. If the user needs a

³ We have obtained slightly different, i.e., better, results, because we used a newer MATLAB release (R14).

good result with a high reliability, she should use the QN. If she can afford several runs, then ES is the correct choice. There is no guarantee that ES detects a better solution, but a high probability. Hybridization is not recommended. The plot on the right shows the same data as on the left, but uses error bars.

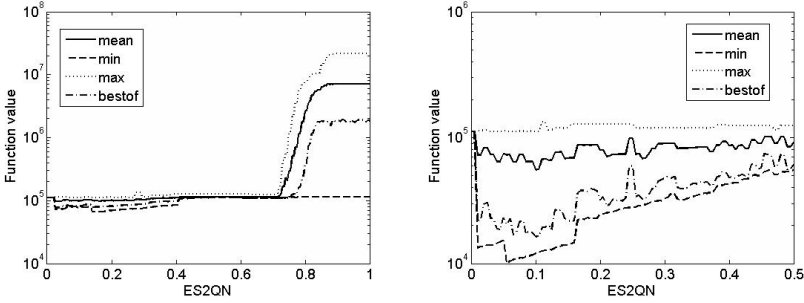


Fig. 8. Meyer. *Left:* Results from the hybridization with 400 function evaluations. *Right:* Results from the hybridization with 1000 function evaluations.

Meyer. Meyer’s function was added to our test set, because it is a three dimensional function on which MATLAB’s BFGS method failed. [14] reports a function value of $3.4675e+007$ at solution found after 10,002 function evaluations, whereas the best known minimum reads $f^* = 87.9458$. This is function 10 from the [12] test set. Figure 8 suggests that QN performs better than ES. But this is an artefact, because both algorithms failed. They did not find a value in the vicinity of the optimizer. Hence, the problem is too hard for both algorithms. Increasing the computational budget, i.e., t_{\max} , does not lead to better results. Therefore, the difference is statistical significant—but not scientifically relevant.

5 Analysis

The main research goal of our study addresses the question “Are there situations in which the hybridization of ES and QN methods improve their performance?” The analysis of the experiments produced no clear picture. QN is more robust than the ES in the traditional definition of robustness, i.e., low standard deviations. This robustness can be seen as an disadvantage, e.g., if the optimization practitioner can afford several runs from which she chooses the best.

Looking at local run properties reveals that the performance improvement is caused by the following effect: The ES explores the search space and detects a suitable starting point that is passed to the QN, which performs a local fine tuning. This is superior to the global search behavior of the ES alone and the local strategy of the QN-methods. However, we could not derive general guidelines, e.g., “choose an ES2QN value of 0.31415 to improve the algorithm’s

performance”. The hybrid approach has also some advantages compared to an approach that performs a sampling of the search space in the first phase and runs a QN method in the second phase, because the region of interest is not known in many situations. The ES jumps to a promising region in the first steps, so that the additional refinement with the QN can be performed efficiently. A comparison to multi-start techniques is of great interest and has not been done in our study.

As expected and mentioned in the abstract, hybridization can improve algorithm’s performance, even if the resources are very limited. Restricted resources are standard situations in industrial optimization, because function evaluations are very costly or results must be available immediately, i.e., in optimization via simulation or in real-time optimization scenarios, respectively. We observed the following results that might be transferable to other situations as well: Algorithms that are specialized for certain (simple) optimization scenarios cannot benefit from hybridization. This is understandable, because these algorithms need a certain budget to adapt their internal model, e.g., step sizes in ES or the gradient and Hessian approximation for QN-methods. Switching to another algorithm is costly, it might be beneficial only if no progress can be obtained with the current strategy. Results from Rosenbrock’s function support this assumption.

It is important to tune the ES, i.e., to determine suitable algorithm designs. SPO, or related tools, can provide a quick overview of suitable parameter settings. Evolution strategies with standard setting from the literature failed in our scenarios. Not only the algorithms have to be tuned before the experiment is started—it was crucial to find an experimental setup that is neither too hard nor too easy for the algorithms as can be seen from Meyer’s function.

6 Summary

No general recommendations—especially for real-world optimization problems—can be given here, because several factors influence the algorithm’s performance. Consider the computational budget: Modifications lead to different results. SPO or related techniques can be applied in this situation, because they can improve the performance significantly. There is no need for hybridization if well tuned algorithms on simple test functions are considered. Only if the problem structure is complex, the combination of global, stochastic search and local, gradient-based strategies is useful. The hybrid ESQN communicates only the best found solutions between its two parts. It may however be beneficial to take over the already learned internal model of the EA (mutation strengths) into the QN-method. Investigating this remains as a task for future research.

Acknowledgment. The research leading to this paper was supported by the DFG (Deutsche Forschungsgemeinschaft) as part of the collaborative research center “Computational Intelligence” (531) and by project grant no. 252441, “Mehrkriterielle Struktur- und Parameteroptimierung verfahrenstechnischer Prozesse mit evolutionären Algorithmen am Beispiel gewinnorientierter unscharfer destillativer Trennprozesse”.

References

1. T. Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York NY, 1996.
2. Thomas Bartz-Beielstein. Experimental analysis of evolution strategies—overview and comprehensive introduction. Interner Bericht des Sonderforschungsbereichs 531 Computational Intelligence CI-157/03, Universität Dortmund, Germany, November 2003.
3. Thomas Bartz-Beielstein. *Experimental Research in Evolutionary Computation—The New Experimentalism*. Springer, Berlin, Heidelberg, New York, 2006.
4. H.-G. Beyer and H.-P. Schwefel. Evolution strategies—A comprehensive introduction. *Natural Computing*, 1:3–52, 2002.
5. C. G. Broyden. The convergence of a class of double-rank minimization algorithms. *Journal of the Institute of Mathematics and Its Applications*, 6:76–90, 1970.
6. Paul R. Cohen. *Empirical Methods for Artificial Intelligence*. MIT Press, Cambridge MA, 1995.
7. R. Fletcher. A new approach to variable metric algorithms. *Computer Journal*, 13:317–322, 1970.
8. D. Goldfarb. A family of variable metric updates derived by variational means. *Mathematics of Computing*, 24:23–26, 1970.
9. D.R. Jones, M. Schonlau, and W.J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13:455–492, 1998.
10. Natalio Krasnogor and Jim E. Smith. A Tutorial for Competent Memetic Algorithms: Model, Taxonomy and Design Issues. *IEEE Transactions on Evolutionary Computation*, 5(9):474–488, 2005.
11. S.N. Lophaven, H.B. Nielsen, and J. Søndergaard. DACE—A Matlab Kriging Toolbox. Technical Report IMM-REP-2002-12, Informatics and Mathematical Modelling, Technical University of Denmark, Copenhagen, Denmark, 2002.
12. J. J. More, B. S. Garbow, and K. E. Hillstom. Testing unconstrained optimization software. *ACM Transactions on Mathematical Software*, 7(1):17–41, 1981.
13. Pablo Moscato. On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms. Technical Report Caltech Concurrent Computation Program, Report. 826, California Institute of Technology, Pasadena, California, USA, 1989.
14. Arnold Neumaier. “Results for moré/garbow/hillstom test problems”, 2006. <http://www.mat.univie.ac.at/~neum/glopt/results/more/moref.html>. Cited 19 Mai 2006.
15. H.H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *Computer Journal*, 3:175–184, 1960.
16. J. Sacks, W. J. Welch, T. J. Mitchell, and H. P. Wynn. Design and analysis of computer experiments. *Statistical Science*, 4(4):409–435, 1989.
17. T. J. Santner, B. J. Williams, and W. I. Notz. *The Design and Analysis of Computer Experiments*. Springer, Berlin, Heidelberg, New York, 2003.
18. D. F. Shanno. Conditioning of quasi-Newton methods for function minimization. *Mathematics of Computing*, 24:647–656, 1970.

Author Index

- Bartz-Beielstein, Thomas 178
Blum, Christian 94
Breaban, Mihaela 139
- Chiarandini, Marco 162
Cipriano, Raffaele 110
Cotta, Carlos 150
Croitoru, Cornelius 139
- delaOssa, Luis 42
Dhaenens, Clarisse 57
Di Gaspero, Luca 110
Dotú, Iván 150
Dovier, Agostino 110
- Fernández, Antonio J. 150
- Gámez, José A. 42
- Ibaraki, Toshihide 13
Ioannou, George 124
Ionita, Madalina 139
- Jourdan, Laetitia 57
- Kapanoglu, Muzafer 28
Koc, Ilker Ozan 28
- Larsen, Kim S. 162
- Melián, Belén 82
- Nakamura, Kouji 13
- Paraskevopoulos, Dimitris C. 124
Preuss, Mike 178
Puerta, José M. 42
- Raidl, Günther R. 1
Repoussis, Panagiotis P. 124
Rodríguez-Martín, Inmaculada 70
Rudolph, Günter 178
- Salazar-González, Juan-José 70
Stützle, Thomas 162
- Talbi, El-Ghazali 57
Tarantilis, Christos D. 124
- Van Hentenryck, Pascal 150
- Yábar Vallès, Mateu 94