Lecture Notes in Computer Science 2264
Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Tokyo

Kathleen Steinhöfel (Ed.)

Stochastic Algorithms: Foundations and Applications

International Symposium, SAGA 2001
Berlin, Germany, December 13-14, 2001
Proceedings

Springer

Series Editors

Gerhard Goos, Karlsruhe University, Germany
Juris Hartmanis, Cornell University, NY, USA
Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editor

Kathleen Steinhöfel
GMD - National Research Center for Information Technology
Institute for Computer Architecture and Software Engineering
Kekuléstr. 7, 12489 Berlin-Adlershof, Germany
E-mail: kathleen.steinhoefel@gmd.de

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Stochastic algorithms: foundations and applications : international symposium ; proceedings / SAGA 2001, Berlin, Germany, December 13 - 14, 2001 / Kathleen Steinhöfel (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Tokyo : Springer, 2001
(Lecture notes in computer science ; Vol. 2264)
ISBN 3-540-43025-3

CR Subject Classification (1998):F.2, F.1.2, G.1.2, G.1.6, G.2, G.3

ISSN 0302-9743
ISBN 3-540-43025-3 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de

Typesetting: Camera-ready by author, data conversion by Steingräber Satztechnik GmbH, Heidelberg
Printed on acid-free paper SPIN 10846149 06/3142 5 4 3 2 1 0

Preface

SAGA 2001, the first Symposium on Stochastic Algorithms, Foundations and Applications, took place on December 13–14, 2001 in Berlin, Germany. The present volume comprises contributed papers and four invited talks that were included in the final program of the symposium.

Stochastic algorithms constitute a general approach to finding approximate solutions to a wide variety of problems. Although there is no formal proof that stochastic algorithms perform better than deterministic ones, there is evidence by empirical observations that stochastic algorithms produce for a broad range of applications near-optimal solutions in a reasonable run-time.

The symposium aims to provide a forum for presentation of original research in the design and analysis, experimental evaluation, and real-world application of stochastic algorithms. It focuses, in particular, on new algorithmic ideas involving stochastic decisions and exploiting probabilistic properties of the underlying problem domain. The program of the symposium reflects the effort to promote cooperation among practitioners and theoreticians and among algorithmic and complexity researchers of the field. In this context, we would like to express our special gratitude to DaimlerChrysler AG for supporting SAGA 2001.

The contributed papers included in the proceedings present results in the following areas: Network and distributed algorithms; local search methods for combinatorial optimization with application to constraint satisfaction problems, manufacturing systems, motor control unit calibration, and packing flexible objects; and computational learning theory.

The invited talk by Juraj Hromkovič surveys fundamental results about randomized communication complexity. In the talk, the efficiency of randomized communication is related to deterministic and nondeterministic communication models. Martin Sauerhoff discusses randomized variants of branching programs which allow the relative power of deterministic, nondeterministic, and randomized algorithms to be studied. Recent results on random 3-SAT formulas are summarized by Gregory Sorkin. The focus is on bounds for their density and shows how to tune so-called myopic algorithms optimally. Thomas Zeugmann gives an overview on stochastic finite learning that connects concepts from PAC learning and models of inductive inference learning.

Our special thanks go to all who supported SAGA 2001, to all authors who submitted papers, to all members of the program committee who provided very detailed referee reports, to the invited speakers, to the organizing committee, and to the sponsoring institutions.

December 2001 Kathleen Steinhöfel

Organization

SAGA 2001 was organized by the GMD - National Research Center for Information Technology, Institute for Computer Architecture and Software Engineering FIRST, Berlin.

Organization Committee

Andreas Jakoby
Babette Neumann
Kathleen Steinhöfel
Peggy Krüger
Friedrich Wilhelm Schröer
Armin Wolf

Program Committee

Andreas Albrecht (University of Hertfordshire, UK)
Juraj Hromkovič (RWTH Aachen, Germany)
Oktay Kasim-Zade (Moscow State University, Russia)
Frieder Lohnert (DaimlerChrysler AG, Germany)
Evelyne Lutton (INRIA, France)
Dirk Mattfeld (University of Bremen, Germany)
Heinz Mühlenbein (GMD AIS, Germany)
Christian Scheideler (Johns Hopkins University, USA)
Gregory Sorkin (IBM Research, NY, USA)
Kathleen Steinhöfel (Chair, GMD FIRST, Germany)
Toby Walsh (University of York, UK)
Peter Widmayer (ETH Zurich, Switzerland)
CK Wong (The Chinese University, Hong Kong)
Thomas Zeugmann (Medical University of Lübeck, Germany)

Sponsoring Institutions

DaimlerChrysler AG, Germany
IT Service Omikron GmbH

Table of Contents

Invited Talk
Randomized Communication Protocols (A Survey) 1
Juraj Hromkovič

Optimal Mutation Rate Using Bayesian Priors
for Estimation of Distribution Algorithms 33
Thilo Mahnig, Heinz Mühlenbein

An Experimental Assessment of a Stochastic, Anytime, Decentralized,
Soft Colourer for Sparse Graphs 49
Stephen Fitzpatrick, Lambert Meertens

Invited Talk
Randomized Branching Programs 65
Martin Sauerhoff

Yet Another Local Search Method for Constraint Solving 73
Philippe Codognet, Daniel Diaz

An Evolutionary Algorithm for the Sequence Coordination
in Furniture Production 91
Carlo Meloni

Evolutionary Search for Smooth Maps
in Motor Control Unit Calibration 107
Jan Poland, Kosmas Knödler, Alexander Mitterer,
Thomas Fleischhauer, Frank Zuber-Goos, Andreas Zell

Invited Talk
Some Notes on Random Satisfiability 117
Gregory B. Sorkin

Prospects for Simulated Annealing Algorithms
in Automatic Differentiation 131
Uwe Naumann, Peter Gottschling

Optimization and Simulation: Sequential Packing of Flexible Objects
Using Evolutionary Algorithms 145
Henning Behnke, Michael Kolonko, Ulrich Mertins, Stefan Schnitter

Invited Talk
Stochastic Finite Learning 155
Thomas Zeugmann

Sequential Sampling Algorithms: Unified Analysis and Lower Bounds 173
Ricard Gavaldà, Osamu Watanabe

Approximate Location of Relevant Variables
under the Crossover Distribution 189
Peter Damaschke

Author Index 203

Randomized Communication Protocols (A Survey)

Juraj Hromkovič*

Lehrstuhl für Informatik I, RWTH Aachen,
Ahornstraße 55, 52074 Aachen, Germany

Abstract. There are very few computing models for which the power of randomized computing is as well understood as for communication protocols and their communication complexity. Since the communication complexity is strongly related to several complexity measures of distinct basic models of computation, there exist possibilities to transform some results about randomized communication protocols to other computing models, and so communication complexity has established itself as a powerful instrument for the study of randomization in complexity theory. The aim of this work is to survey the fundamental results about randomized communication complexity with the focus on the comparison of the efficiency of deterministic, nondeterministic and randomized communication.

Keywords: Randomized computing, communication complexity, two-party protocols.

1 Introduction

The communication complexity of two-party protocols was introduced by Yao [1] in 1978-79. (Note that communication complexity was implicitly considered by Abelson [2], too.) The initial goal was to develop a method for proving lower bounds on the complexity of distributed and parallel computations, with a special emphasis on VLSI computations [3,4,5,6,7,8,9,10,11].

But the study of communication complexity contributed to complexity theory much more than one had expected at the beginning in the early eighties and recently its relation to VLSI theory does not belong to the most important applications of communication protocols. The communication complexity theory established itself as a subarea of complexity theory and the main reasons for this successful story are the following ones:

(i) Communication complexity (similarly as Kolmogorov complexity) became a powerful method in proving lower bounds on fundamental complexity measures of concrete problems (see [3,4] for some surveys). In some applications it caused a breakthrough in the long efforts in proving lower bounds (see, for

* Supported by the DFG Project Hr.

K. Steinhöfel (Ed.): SAGA 2001, LNCS 2264, pp. 1–32, 2001.

instance, [12,13,14,15,16,17]). This succeeded especially due to the development of a powerful nontrivial mathematical machinery for determining the communication complexity of concrete problems, while the relation of communication complexity to other complexity measures is typically transparent and usually easy to get.

(ii) Communication complexity essentially contributed to our understanding of randomized computation. There are very few computing models for which the power of randomized computing is as well understood as for two-party communication protocols. Moreover, due to (i) one can extend the results about randomized communication complexity to some other basic computing models.

This survey focuses on the study of randomized communication protocols. It is organized as follows. Section 2 presents the basic model of two-party communication protocols and its nondeterministic and randomized extensions. Here, we consider Las Vegas, one-sided-error and two-sided-error (bounded-error) randomization. Section 3 gives a short overview about methods for proving lower bounds on communication complexity. The central parts of this paper are Section 4 and 5. Section 4 relates the efficiency of randomized communication to the deterministic and nondeterministic communication models. Section 5 investigates whether one can restrict the number of random bits without restricting the power of randomized communication protocols. Because of the lack of space we do not present any survey of applications of results presented in Sections 4 and 5 to other computing models.

2 Definition of Communication Protocols

Informally, a **two-party (communication) protocol** consists of two computers C_I and C_{II} and a communication link between them. A protocol computes a finite function $f : U \times V \to Z$ in the following way. At the beginning C_I obtains an input $\alpha \in U$ and C_{II} obtains an input $\beta \in V$. Then C_I and C_{II} communicate according to the rules of the protocol by exchanging binary messages until one of them knows $f(\alpha, \beta)$. The **communication complexity of the computation on an input (α, β)** is the sum of the lengths of messages exchanged. The communication complexity of the protocol is the maximum of the complexities over all inputs from $U \times V$. The **communication complexity of f**, **cc(f)**, is the complexity of the best protocol for f.

Typically, one considers the situation that $U = \{0,1\}^n$, $V = \{0,1\}^n$ and $Z = \{0,1\}$, i.e. f is a Boolean function of $2n$ variables. Since this restriction is sufficient for our purposes we give the formal definition for protocols computing Boolean functions only. In what follows we describe the computation of a protocol on an specific input by a string $c_1\$c_2\$\ldots\$c_k\$c_{k+1}$, where $c_i \in \{0,1\}^+$ for $i = 1,\ldots,k$ are the messages exchanged (c_1 is the first message sent from C_I to C_{II}, c_2 is the second message submitted from C_{II} to C_I, etc.) and c_{k+1} is the result of the computation. The part $c_1\$c_2\$\ldots\$c_k$ is called the **history of communication**.

Definition 1. *Let $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ be a Boolean function over a set $X = \{x_1, x_2, \ldots, x_{2n}\}$ of Boolean variables. A* **protocol P over X** *(or* **over $\{0,1\}^n \times \{0,1\}^n$***) is a function from $\{0,1\}^n \times \{0,1,\$\}^*$ to $\{0,1\}^+ \cup \{\text{accept}, \text{reject}\}$ such that*

(i) P has the **prefix-freeness property***:*
For each $c \in \{0,1,\$\}^*$ and any two different $\alpha, \beta \in \{0,1\}^n$, $P(\alpha, c)$ is no proper prefix of $P(\beta, c)$.
{The next message $P(\alpha, c)$ is computed either by C_I or by C_{II} in the dependence of the input α of C_I (C_{II}) and the history c of communication. The prefix-freeness property assures that the messages exchanged between the two computers are self-delimiting, and no extra "end of transmission" symbol is required. To visualize the end of messages in the history of communication c, we write the special symbol \$ at the end of every message.}

(ii) If $P(\alpha, c) \in \{\text{accept}, \text{reject}\}$ for an $\alpha \in \{0,1\}^m$, and $c \in (\{0,1\}^+\$)^{2p}$ for some $p \in \mathbb{N}$ [for $c \in (\{0,1\}^+\$)^{2p+1}$], then for all $q \in \mathbb{N}$, $\gamma \in \{0,1\}^m$, $d \in (\{0,1\}^+\$)^{2q+1}$ [$d \in (\{0,1\}^+\$)^{2q}$], $P(\gamma, d) \notin \{\text{accept}, \text{reject}\}$ for every communication history $d \in \{0,1,\$\}^*$.
{This property assures that the output value is always computed by the same computer independently of the input assignment.}

(iii) For every $c \in \{0,1,\$\}^*$, if $P(\alpha, c) \in \{\text{accept}, \text{reject}\}$ for some $\alpha \in \{0,1\}^n$, then $P(\beta, c) \in \{\text{accept}, \text{reject}\}$ for every $\beta \in \{0,1\}^n$.
{This property assures that if the computer C_I (C_{II}) computes the output for an input, then the other computer C_{II} (C_I) knows that C_I (C_{II}) knows the result, and so it does not wait for further communication.}

A **computation of P on an input $(\alpha, \beta) \in \{0,1\}^n \times \{0,1\}^n$** *is a string $c = c_1\$c_2\$\ldots\$c_k\$c_{k+1}$, where*

(1) $k \geq 0$, $c_1, \ldots, c_k \in \{0,1\}^+$, $c_{k+1} \in \{\text{accept}, \text{reject}\}$, and
(2) for every integer l, $0 \leq l \leq k$,
(2.1) if l is even, then $c_{l+1} = P(\alpha, c_1\$c_2\$\ldots\$c_l\$)$, and
{The message c_{l+1} is sent by C_I, and C_I computes c_{l+1} in dependence of its input part α and of the whole current communication history $c_1\$c_2\$\ldots\$c_l\$$.}
(2.2) if l is odd, then $c_{l+1} = P(\beta, c_1\$c_2\$\ldots\$c_l\$)$.
{The message c_{l+1} is sent from C_{II} to C_I, and C_{II} computes C_{l+1} in the dependence of its input β and the communication history $c_1\$c_2\$\ldots\$c_l\$$.}

For every computation $c = c_1\$c_2\$\ldots\$c_k\$c_{k+1}$,

$$\mathbf{Com}(c) = c_1\$c_2\$\ldots\$c_k\$$$

is the **communication** *(or* **communication history***) of c.*

We say that P **computes** *f if, for every $(\alpha, \beta) \in \{0,1\}^n \times \{0,1\}^n$, the computation of P on (α, β) is finite and ends with "*accept*" if and only if $f(\alpha, \beta) = 1$. In what follows we also say that a computation is* **accepting** *(***rejecting***) if it ends with* accept*(*reject*).*

The **length of a computation c** *is the total length of all messages in c (ignoring $\$$'s and the final* accept/reject*). The* **communication complexity of the protocol P, $cc(P)$**, *is the maximum of all computation lengths over all inputs $(\alpha, \beta) \in \{0,1\}^n \times \{0,1\}^n$.*

The **communication complexity of f** *is*

$$\mathbf{cc}(f) = \min\{\mathrm{cc}(P) \mid P \textit{ computes } f\}.$$

□

We use the notation $P(\alpha, \beta)$ for the output $\in \{\text{accept}, \text{reject}\}$ of the computation of the protocol P on the input (α, β). Note that "accept" is used to denote the result "1", and "reject" is used to denote the result "0". We use the notations "accept" and "reject" instead of "1" and "0" in order to distinguish between the communication bits and the results. Besides the reason above it will be convenient to be able to speak about accepting and rejecting computations in what follows.

To illustrate the above definition we consider the following simple examples. Let

$$\mathrm{Eq}_n(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n) = \bigwedge_{i=1}^{n} x_i \equiv y_i$$

be the Boolean function of $2n$ variables from $\{0,1\}^n \times \{0,1\}^n$ to $\{0,1\}$ that takes the value 1 iff the first half of the input is equal to the second half of the input. A protocol P computing Eq_n can simply work as follows. C_I sends its whole input $\alpha \in \{0,1\}^n$ to C_{II} and C_{II} compares whether α is identical with its input $\beta \in \{0,1\}^n$. Formally,

$P(\alpha, \lambda) = \alpha$, and[1]

$$P(\beta, \alpha\$) = \begin{cases} \text{accept} & \text{if } \alpha \equiv \beta. \\ \text{reject} & \text{if } \alpha \not\equiv \beta. \end{cases}$$

We observe that the above described strategy (C_I sends its whole input to C_{II}) works for every function and so

$$\mathrm{cc}(f) \leq n$$

for every Boolean function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ of $2n$ variables.

Now, consider the symmetric function $s_{2n} : \{0,1\}^{2n} \to \{0,1\}$ that takes the value 1 if and only if the number of 1's in the input is equal to the number of 0's in the input. A simple way to compute s_{2n} by a protocol follows. C_I sends the binary representation of the number $\#_1(\alpha)$ of 1's in its input $\alpha \in \{0,1\}^n$. C_{II} checks whether $\#_1(\alpha) + \#_1(\beta) = n$, where $\beta \in \{0,1\}^n$ is the input of C_{II}. Obviously, the communication complexity of this protocol is $\lceil \log_2(n+1) \rceil$.

Note that we shall later show that the above protocols for Eq_n and s_{2n} are optimal. We observe that these protocols are very simple because the whole

[1] λ denotes the empty word.

communication consists of the submission of one message. The protocols whose computations contains at most one message are called **one-way protocols** in what follows. For every Boolean function f

$$\mathbf{cc_1(}\boldsymbol{f}\mathbf{)} = \min\{\mathrm{cc}(P) \mid P \text{ is a one-way protocol computing } f\}$$

is the **one-way communication complexity of f**.

In Section 3 we shall see that there can be an exponential gap between $\mathrm{cc}_1(f)$ and $\mathrm{cc}(f)$. The following function $f_{\mathrm{ind}(n)}(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n)$ for $n = 2^k$, $k \in I\!N - \{0\}$, is an example of a computing problem where one can profit from the exchange of more than one message between C_I and C_{II}. Let, for every binary string $\alpha = \alpha_0\alpha_1 \ldots \alpha_k$,

$$\mathbf{Number(}\boldsymbol{\alpha}\mathbf{)} = \sum_{i=0}^{k} \alpha_{k-i} \cdot 2^i$$

be the number with the binary representation α. The function $f_{\mathrm{ind}(n)}$ takes the value 1 iff

$$x_{\mathrm{Number}(y_{\mathrm{Number}(x_1x_2\ldots x_{\lceil \log_2 n\rceil})+1}\cdots y_{(\mathrm{Number}(x_1x_2\ldots x_{\lceil \log_2 n\rceil})+\lceil \log_2 n\rceil) \bmod n})+1} = 1.$$

Informally, the first $\log_2 n$ values of the variables $x_1, x_2, \ldots, x_{\log_2 n}$ determine a position (an index) $a = \mathrm{Number}(x_1, \ldots, x_{\log_2 n})+1$. y_a and the following $\log_2 n - 1$ values of y variables determine an index b, and one requires $x_b = 1$ for the result 1. We describe a protocol P that computes $f_{\mathrm{ind}(n)}$. C_I sends $x_1x_2 \ldots x_{\lceil \log_2 n\rceil}$ to C_{II}. After that C_{II} sends the message $y_a y_{(a+1) \bmod n} \cdots y_{(a+\log_2 n-1) \bmod n}$ to C_I, where $a = \mathrm{Number}(x_1 \ldots x_{\lceil \log_2 n\rceil}) + 1$. Now, C_I accepts iff

$$x_{\mathrm{Number}(y_a y_{(a+1) \bmod n}\cdots y_{(a+\lceil \log_2 n\rceil-1) \bmod n})+1} = 1.$$

The communication complexity of this protocol is $2 \cdot \lceil \log_2 n \rceil$.

In what follows we use, for all nonnegative integers l, k, $l \geq \lceil \log_2 k \rceil$, $l \geq 1$, $\mathbf{BIN_l(}\boldsymbol{k}\mathbf{)}$ to denote the binary representation of k by a binary string of length l. This means that if $l > \lceil \log_2 k \rceil$, $l - \lceil \log_2 k \rceil$ leading 0's are added to the representation.

One can introduce nondeterminism for protocols in the usual way. Because of this we prefer to give an informal description of nondeterministic protocols rather than an exact formal definition.

Let $f : U \times V \to \{0, 1\}$ be a finite function. A **nondeterministic protocol** P computing on inputs from $U \times V$ consists of two nondeterministic computers C_I and C_{II}. At the beginning C_I obtains an input $\alpha \in U$, and C_{II} obtains an input $\beta \in V$. As in the deterministic case, the computation consists of a number of communication rounds, where in one round one computer sends a message to the other one. The computation finishes when one of the computers decides to accept or to reject the input (α, β). In contrast to the deterministic case, C_I can be viewed as a relation on

$$(U \times \{0, 1, \$\}^*) \times (\{0, 1\}^+ \cup \{\text{accept,reject}\})$$

and C_{II} can be viewed as a relation on

$$(V \times \{0,1,\$\}^*) \times (\{0,1\}^+ \cup \{\text{accept,reject}\}).$$

This means that, for every argument (α, c), C_I (C_{II}) nondeterministically chooses a message from a finite set of possible messages determined by the argument (α, c).

We say that $\boldsymbol{P}$ **computes** $\boldsymbol{f}$ if for every $(\alpha, \beta) \in U \times V$,

(i) if $f(\alpha, \beta) = 1$, then there exists an accepting computation of P on the input (α, β), and
(ii) if $f(\alpha, \beta) = 0$, then all computations of P on (α, β) are rejecting ones.

Again, we require that the prefix-freeness property and the property that exactly one computer takes the final decision for all inputs are satisfied.

The **nondeterministic communication complexity of** $\boldsymbol{P}$, denoted by $\mathbf{ncc}(\boldsymbol{P})$, is the maximum of the lengths of all accepting computations of P. The **nondeterministic communication complexity of** $\boldsymbol{f}$ is

$$\mathbf{ncc}(\boldsymbol{f}) = \min\{\text{ncc}(P) \mid P \text{ is a nondeterministic protocol computing } f\}.$$

To show the power of nondeterminism in communication consider, for every positive integer n, the function $\text{Ineq}_n : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ defined by

$$\text{Ineq}_n(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n) = \bigvee_{i=1}^{n} (x_i \oplus y_i),$$

where $\oplus$ denotes the exclusive or (plus mod 2).

A nondeterministic protocol P that accepts all inputs $(\alpha, \beta) \in \{0,1\}^n \times \{0,1\}^n$ with $\alpha \neq \beta$ can work as follows. For every input $\alpha = \alpha_1\alpha_2 \ldots \alpha_n$, C_I nondeterministically chooses a number $i \in \{1, \ldots, n\}$ and sends the message $\alpha_i \text{BIN}_{\lceil \log_2 n \rceil}(i)$ to C_{II}, where $\text{BIN}_{\lceil \log_2 n \rceil}(i)$ is the binary representation of i of the length[2] $\lceil \log_2 n \rceil$. Now, for every input $\beta = \beta_1 \ldots \beta_n$ of C_{II}, C_{II} accepts if and only if $\alpha_i \neq \beta_i$. In the case $\alpha_i = \beta_i$, C_{II} rejects the input. Obviously, for all $\alpha, \beta \in \{0,1\}^n$ with $\alpha \neq \beta$, there exists a j such that $\alpha_j \neq \beta_j$ and so there exists an accepting computation of P on (α, β). On the other hand if $\alpha = \beta$, then all n different computations of P on (α, β) are rejecting. So, P computes Ineq_n within the communication complexity $\lceil \log_2 n \rceil + 1$, i.e.,

$$\text{ncc}(\text{Ineq}_n) \leq \lceil \log_2 n \rceil + 1.$$

Similarly as in the deterministic case, one can consider **one-way nondeterministic protocols** whose computations contain at most one message. Let $\mathbf{ncc_1}(\boldsymbol{f})$ denote the **one-way nondeterministic communication complexity of** $\boldsymbol{f}$. In contrast to the deterministic case, we show that there is no difference in the power of nondeterministic protocols and one-way nondeterministic protocols.

[2] This means, that additional 0's are taken to achieve the required length, if necessary.

Theorem 1. *For every finite function f,*

$$\mathrm{ncc}(f) = \mathrm{ncc}_1(f).$$

Proof. Let f be a function from $U \times V$ to $\{0,1\}$, and let $D = (C_I, C_{II})$ be a nondeterministic protocol computing f. We construct a one-way nondeterministic protocol D_1 that computes f within the communication complexity $\mathrm{ncc}(D)$. In what follows we say that a computation $C = c_1\$c_2\$\ldots\$c_k\$c_{k+1}$ is **consistent** for C_I with an input $\alpha \in U$ (or from the point of view of C_I and α) if C_I with the input α has the possibility to send the messages $c_1, c_3, c_5, \ldots$ when receiving the messages $c_2, c_4, c_6, \ldots$ from C_{II}. Similarly, one can define the consistency of C according to C_{II} with an input $\beta \in V$. Obviously, if C is a consistent computation for C_I on an input α and for C_{II} on an input β, then C is a possible computation of (C_I, C_{II}) on the input (α, β).

Let $C_1, C_2, \ldots, C_k$ be all accepting computations of D over all inputs from $U \times V$. Clearly, $k \leq 2^{\mathrm{ncc}(D)}$. Let D_1 consist of the computers C_I^1 and C_{II}^1. For every $\alpha \in U$, C_I^1 nondeterministically chooses an $i \in \{1, \ldots, k\}$, and it sends the message $\mathrm{BIN}_{\mathrm{ncc}(D)}(i)$ to C_{II}^1 if C_i is a possible accepting computation of D from the point of view of C_I and α. In other words, C_I^1 can send the binary representation of i if and only if there exists an $\gamma \in V$ such that C_i is the accepting computation of D on (α, γ). For every input β of C_{II}^1, when C_{II}^1 receives the binary representation of a number i, then C_{II}^1 accepts if C_i is a consistent accepting computation from the C_{II} and β point of view.

So, D_1 has exactly the same number of accepting computations as D and $\mathrm{ncc}_1(D_1) = \mathrm{ncc}(D)$. □

There are two distinct ways to introduce randomized protocols. One possibility is to take a nondeterministic protocol and to consider a probability distribution for every possible nondeterministic guess. Such randomized protocols are called **private** because each of the computers takes its random bits from a separate source; i.e., C_I (C_{II}) does not know the random bits of C_{II} (C_I). If one of the computers wants to know the random bits influencing the choice of the message submitted by the other computer, then these bits must be communicated. Another possibility to define randomized protocols is to say that a randomized protocol is a probability distribution over a set of deterministic protocols. Such randomized protocols are called **public** randomized protocols because this model corresponds to the situation when both computers have the same source of random bits (i.e. everybody sees the random bits of the other one for free). This second approach represents the well-known paradigm "*foiling an adversary*" of the design of randomized algorithms. So, every efficient public randomized protocol can be viewed as a successful application of this paradigm.

Clearly, public randomized protocols are at least as powerful as private ones. Newman [18] has proved that the relations between the communication complexities of public randomized protocols and private ones are linear for every bounded-error model. We shall present this result in Section 6. Because of this and in order to simplify the matters we formally define the public versions of randomized protocols only.

In the following definitions we use also the notation $P(\alpha,\beta)=1$ (0) instead of $P(\alpha,\beta)=$ accept (reject).

Definition 2. *Let U and V be finite sets. A* **randomized protocol over $U\times V$** *is a pair $R=(Prob,S)$, where*

(i) $S=\{D_1,D_2,\ldots,D_k\}$ is a set of (deterministic) protocols over $U\times V$, and
(ii) $Prob$ is a probability distribution over the elements of S.

For $i=1,2,\ldots,k$, $\boldsymbol{Prob(D_i)}$ is the probability that the protocol D_i is randomly chosen to work on a given input.

For an input $(\alpha,\beta)\in U\times V$, the **probability that R computes an output z** *is*

$$\boldsymbol{Prob(R(\alpha,\beta)=z)}=\sum_{\substack{D\in\{D_1,\ldots,D_k\}\\ D(\alpha,\beta)=z}} Prob(D).$$

The **communication complexity of R** *is*

$$\mathbf{cc(R)}=\max\{\mathrm{cc}(D)\mid D\in S\}.$$

A randomized protocol $R=(Prob,S)$ is called **one-way** *if all elements of S are one-way protocols.*

For every randomized protocol $R=(Prob,\{D_1,D_2,\ldots,D_k\})$ with a uniform probability distribution $Prob$, the **degree of randomness** *of P is $\lceil\log_2 k\rceil$. Since one can unambiguously identify every D_i with a binary string of length $\lceil\log_2 k\rceil$, we call $\lceil\log_2 k\rceil$ the* **number of random bits of R**, *too.*

In what follows we consider only randomized protocols computing Boolean functions. In contrast to the previous protocol models we also allow a "**neutral**" output "**?**", whose meaning is that the protocol was not able to compute any final answer in the given computation (random attempt). We consider this possibility for Las Vegas protocols that never err but may produce the answer "?" with a bounded probability.

Note that in what follows we use also the notation $R(\alpha,\beta)=1$ ($R(\alpha,\beta)=0$) instead of $R(\alpha,\beta)=$ "accept" ($R(\alpha,\beta)=$ "reject"). This is convenient because we obtain the possibility to speak about the probability of the event $R(\alpha,\beta)=f(\alpha,\beta)$ in this way.

Definition 3. *Let $f: U\times V\to\{0,1\}$ be a finite function. We say, that a randomized protocol $R=(Prob,S)$ is a* **(public) Las Vegas protocol for f** *if*

(i) for every $(\alpha,\beta)\in U\times V$ with $f(\alpha,\beta)=1$,
$Prob(R(\alpha,\beta)=1)\geq\frac{1}{2}$ and $Prob(R(\alpha,\beta)=0)=0$, and
(ii) for every $(\alpha,\beta)\in U\times V$ with $f(\alpha,\beta)=0$,
$Prob(R(\alpha,\beta)=0)\geq\frac{1}{2}$ and $Prob(R(\alpha,\beta)=1)=0$.

The **Las Vegas communication complexity of f** *is*

$$\mathbf{lvcc(f)}=\min\{\mathrm{cc}(R)\mid R \text{ is a Las Vegas protocol for } f\}.$$

The **one-way Las Vegas communication complexity of f** *is*

$$\mathbf{lvcc_1(f)}=\min\{\mathrm{cc}(R)\mid R \text{ is a one-way Las Vegas protocol for } f\}.$$

We present a simple example of a one-way Las Vegas protocol. More involved ideas for the design of Las Vegas protocols can be found in Section 4. Consider the function $\text{Index}_n : \{0,1\}^n \times \{1,2,\ldots,n\} \to \{0,1\}$ defined as follows[3]

$$\text{Index}_n((x_1, x_2, \ldots, x_n), j) = x_j.$$

A Las Vegas one-way protocol D for Index_n can be described as the pair ($Prob$, $\{D_1, D_2\}$), where

(i) $Prob(D_1) = Prob(D_2) = \frac{1}{2}$,
(ii) $D_1 = (D_{I,1}, D_{II,1})$, where $D_{I,1}$ sends the first half $\alpha_1 \ldots \alpha_{\lceil \frac{n}{2} \rceil}$ of its input $\alpha_1 \ldots \alpha_n$ to $D_{II,1}$. $D_{II,1}$ outputs α_j if the input j of $D_{II,1}$ belongs to $\{1,2,\ldots,\lceil \frac{n}{2} \rceil\}$. If $j > \lceil \frac{n}{2} \rceil$, then $D_{II,1}$ outputs "?",
(iii) $D_2 = (D_{I,2}, D_{II,2})$ where $D_{I,2}$ sends the second half $\alpha_{\lceil \frac{n}{2} \rceil + 1} \ldots \alpha_n$ of its input $\alpha_1 \ldots \alpha_n$ to $D_{II,2}$. $D_{II,2}$ outputs α_j if the input j of $D_{II,2}$ belongs to $\{\lceil \frac{n}{2} \rceil + 1, \ldots, n\}$. If $j \leq \lceil \frac{n}{2} \rceil$, then $D_{II,2}$ outputs "?".

Another possibility to describe D is as follows.

Las Vegas one-way protocol $D = (D_I, D_{II})$ for Index_n.

Input: (α, j), $\alpha = \alpha_1 \ldots \alpha_n \in \{0,1\}^n$, $j \in \{1,\ldots,n\}$.
{ D_I gets the input α, and D_{II} gets the input j.}
Step 1: D_I chooses a random bit $r \in \{0,1\}$.
{ Note, that D_{II} knows r, too.}
If $r = 0$, then D_I sends the message $\alpha_1 \alpha_2 \ldots \alpha_{\lceil \frac{n}{2} \rceil}$.
If $r = 1$, then D_I sends the message $\alpha_{\lceil \frac{n}{2} \rceil + 1} \alpha_{\lceil \frac{n}{2} \rceil + 2} \ldots \alpha_n$.
Step 2: If $r = 0$ and $j \in \{1,2,\ldots,\lceil \frac{n}{2} \rceil\}$, then D_{II} outputs α_j.
If $r = 1$ and $j > \lceil \frac{n}{2} \rceil$, then D_{II} outputs α_j.
Else, D_{II} outputs "?".

In what follows we shall prefer the second form of the description of randomized protocols. Clearly, D never errs, and the probability of giving the output "?" is $\frac{1}{2}$ for every input $(\alpha, j) \in \{0,1\}^n \times \{1,\ldots,n\}$. The communication complexity of D is $\lceil \frac{n}{2} \rceil$.

Note, that the constant $\frac{1}{2}$ bounding the probability of the output "?" in the definition of (two-way) Las Vegas protocols is not essential from the asymptotic point of view. Instead of giving the output "?" a Las Vegas protocol may start a new communication from the beginning with new random bits. If it outputs "?" only if it reaches "?" in k independent computation attempts, then the probability to obtain the output "?" decreases from $\frac{1}{2}$ to $\frac{1}{2^k}$, but the communication complexity increases only by a factor of k in comparison with the original protocol.

Definition 4. *Let $f : U \times V \to \{0,1\}$ be a finite function. We say that a randomized protocol $R = (Prob, S)$ is a* **(public) one-sided-error Monte Carlo protocol for f** *if*

[3] Observe, that Index_n can be also viewed as a Boolean function if one represents the numbers $1, 2, \ldots, n$ by binary strings.

(i) *for every* $(\alpha, \beta) \in U \times V$ *with* $f(\alpha, \beta) = 1$,
$Prob(R(\alpha, \beta) = 1) \geq \frac{1}{2}$, *and*
(ii) *for every* $(\alpha, \beta) \in U \times V$ *with* $f(\alpha, \beta) = 0$,
$Prob(R(\alpha, \beta) = 0) = 1$.

We say that a randomized protocol R *is a* **(public) two-sided-error Monte Carlo protocol for** f *if, for every* $(\alpha, \beta) \in U \times V$,

$$Prob(R(\alpha, \beta) = f(\alpha, \beta)) > \frac{2}{3}.$$

The **one-sided-error Monte Carlo communication complexity of** f *is*

$$\mathbf{1mccc}(f) = \min\{\mathrm{cc}(R) \mid R \text{ is a one-sided-error Monte Carlo protocol for } f\}.$$

The **two-sided-error Monte Carlo communication complexity of** f *is*

$$\mathbf{2mccc}(f) = \min\{\mathrm{cc}(R) \mid R \text{ is a two-sided-error Monte Carlo protocol for } f\}.$$

Because of the condition (ii) of one-sided-error Monte Carlo protocols it is clear that private one-sided-error Monte Carlo protocols are a restricted version of nondeterministic ones. For the public randomized protocols defined here we obtain

$$\mathrm{ncc}(f) \leq 1\mathrm{mccc}(f) + \text{ the number of random bits}$$

for every finite function f.

Similarly as in the case of Las Vegas, the constant $\frac{1}{2}$ in the inequality $Prob(R(\alpha, \beta) = 1) \geq \frac{1}{2}$ is not essential for one-sided-error Monte Carlo from the asymptotic point of view and so one-sided-error Monte Carlo protocols can be viewed as a restricted version of two-sided-error Monte Carlo protocols, too.

Example 1. (based on [19]) The idea of a randomized protocol is based on the **abundance of witnesses** method for the design of randomized algorithms. Let f be a Boolean function. A **witness for** $f(\gamma) = a$ is any binary string δ, such that using δ there is an efficient way to prove (verify) that $f(\gamma) = a$. For instance, any factor (nontrivial divisor) y of a number x is a witness of the claim "x *is composite*". Obviously, to check whether $x \bmod y = 0$ is much easier than to prove that x is a composite without any additional information. In general, one considers witnesses only if they essentially decrease the complexity of computing the result. For many functions the difficulty with finding a witness deterministically is that the witness lies in a search space that is too large to be searched exhaustively. However, by establishing that the space contains a large number of witnesses, it often suffices to choose an element at random from the space. The randomly chosen item is likely to be a witness. If this probability is not high enough, an independent random choice of several items reduces the probability that no witness is found.

The framework of this approach is very simple. One has for every input γ a set CandW(γ) that contains all items candidating to be a witness for the input γ. Often CandW(γ) is the same set for all inputs of the same size as γ. Let Witness(γ) contain all witnesses for γ that are in CandW(γ). The aim is to reach a situation where the cardinality of Witness(γ) is proportional to the cardinality of CandW(γ).

To design a randomized protocol for Ineq_n, we say that a **prime p is a witness for $\alpha \neq \beta$**, $\alpha, \beta \in \{0,1\}^n$, if

$$\text{Number}(\alpha) \bmod p \neq \text{Number}(\beta) \bmod p.$$

For every input $(\alpha, \beta) \in \{0,1\}^n \times \{0,1\}^n$, CandW$(\alpha, \beta)$ is the set of all primes from $\{2, 3, \ldots, n^2\}$. Due to the Prime Number Theorem we know that $|\text{CandW}(\alpha, \beta)|$ is approximately $\frac{n^2}{\ln n^2}$. Now, we estimate the lower bound on $|\text{Witness}(\alpha, \beta)|$. Let $\alpha \neq \beta$. If, for a prime p,

$$\text{Number}(\alpha) \bmod p = \text{Number}(\beta) \bmod p, \tag{1}$$

then p divides $h = \text{Number}(\alpha) - \text{Number}(\beta)$. Since $h < 2^n$, h has fewer than n different prime divisors.[4] This means that at most $n-1$ primes from CandW(α, β) have the property (1). So,

$$|\text{Witness}(\alpha, \gamma)| \geq |\text{CandW}(\alpha, \beta)| - n + 1. \tag{2}$$

Now, we use (2) to design our randomized protocol.

One-sided-error Monte Carlo protocol $R = (R_I, R_{II})$ for Ineq_n.

Input: $(\alpha, \beta) \in \{0,1\}^n \times \{0,1\}^n$
Step 1: R_I chooses uniformly a prime $p \in \{2, 3, \ldots, n^2\}$ at random.
{Note, that R_{II} knows this choice of R_I.}
Step 2: R_I computes $s = \text{Number}(\alpha) \bmod p$ and sends the binary representation of s to R_{II}.
{Note, that the length of the message is $\lceil \log_2 n^2 \rceil \leq 2 \cdot \lceil \log_2 n \rceil$.}
Step 3: R_{II} computes $q = \text{Number}(\beta) \bmod p$.
If $q \neq s$, then R_{II} outputs 1 ("accept").
If $q = s$, then R_{II} outputs 0 ("reject").

We show that R is a one-sided-error Monte Carlo protocol for Ineq_n. If $\alpha = \beta$, for an input $(\alpha, \beta) \in \{0,1\}^n \times \{0,1\}^n$, then $\text{Number}(\alpha) \bmod p = \text{Number}(\beta) \bmod p$ for every prime p. So,

$$Prob(R(\alpha, \beta) = \text{"reject"}) = 1.$$

Let $\alpha \neq \beta$, i.e. $\text{Ineq}_n(\alpha, \beta) = 1$. Due to the inequality (2), the probability that R chooses a prime with the property (1) is at most

$$\frac{|\text{CandW}(\alpha, \beta)| - |\text{Witness}(\alpha, \gamma)|}{|\text{CandW}(\alpha, \beta)|} \leq \frac{n-1}{|\text{CandW}(\alpha, \beta)|}.$$

[4] Observe, that $n! > 2^n$.

Since $|\text{CandW}(\alpha,\beta)| \geq \frac{n}{2\ln n^2}$ already for small n's, the probability that R rejects (α,β) is at most

$$\frac{n-1}{|\text{CandW}(\alpha,\beta)|} \leq \frac{n-1}{n^2/2\ln n^2} \leq \frac{2\ln n^2}{n}.$$

Thus,

$$Prob(R(\alpha,\beta) = \text{“accept”}) \geq 1 - \frac{2\ln n^2}{n},$$

that even tends to 1 for sufficiently large n's. So, we have proved

$$1\text{mccc}(\text{Ineq}_n) = O(\log_2 n).$$

3 Lower Bounds on Communication Complexity

In this section we do not deal with randomized protocols. The aim here is to present methods for proving lower bounds on the communication complexity of deterministic and nondeterministic protocols. These methods will be used in the subsequent sections to show the power of randomization in the comparison with deterministic and nondeterministic communications.

The most transparent way to explain the methods for proving lower bounds on communication complexity is to consider the representation of functions to be computed in the form of so-called communication matrices.

Definition 5. *Let $U = \{\alpha_1,\ldots,\alpha_k\}$, $V = \{\beta_1,\ldots,\beta_m\}$ be two sets, and let $f : U \times V \to \{0,1\}$. The* **communication matrix** *of f is the $|U| \times |V|$ Boolean matrix $\boldsymbol{M_f} = [a_{ij}]_{i=1,\ldots,k,j=1,\ldots,m}$ defined by*

$$a_{ij} = a_{\alpha_i\beta_j} = f(\alpha_i,\beta_j).$$

For every $\gamma \in U$, the row corresponding to the input part γ is called the **row of $\boldsymbol{\gamma}$** *and it is denoted by $\mathbf{row_\gamma} = (a_{\gamma\beta_1}, a_{\gamma\beta_2},\ldots,a_{\gamma\beta_m})$. Similarly, the column corresponding to an input part $\delta \in V$ is called the* **column of $\boldsymbol{\delta}$**, *and it is denoted by $\mathbf{column_\delta} = (a_{\alpha_1\delta}, a_{\alpha_2\delta},\ldots,a_{\alpha_k\delta})$.*
For all non-empty index sets $\{i_1,\ldots,i_l\} \subseteq \{1,\ldots,k\}$, $\{j_1,\ldots,j_r\} \subseteq \{1,\ldots,m\}$,

$$\boldsymbol{M_f(\{\alpha_{i_1},\alpha_{i_2},\ldots,\alpha_{i_l}\},\{\beta_{j_1},\beta_{j_2},\ldots,\beta_{j_r}\})}$$

denotes the $l \times r$ submatrix of M_f obtained by the intersection of the rows of M_f that correspond to inputs $\alpha_{i_1},\alpha_{i_2},\ldots,\alpha_{i_l}$ and the columns corresponding to the inputs $\beta_{j_1},\beta_{j_2},\ldots,\beta_{j_r}$.

Observe that one needs an ordering on the elements of U and V in order to unambiguously determine M_f for a function $f : U \times V \to \{0,1\}$. Since we usually work with Boolean functions one may always consider the lexicographic order on the Boolean vectors (binary words). So, for the Boolean function $f_{\text{pr}} : \{0,1\}^3 \times \{0,1\}^2 \to \{0,1\}$ defined by

$$f_{\text{pr}}(u,v) = 1 \text{ iff } v \text{ is a prefix of } u$$

(i.e., $v_1 \equiv u_1, v_2 \equiv u_2$ if $u = u_1u_2u_3, v = v_1v_2$), $M_{f_{\text{pr}}}$ is the following matrix

	00	01	10	11
000	1	0	0	0
001	1	0	0	0
010	0	1	0	0
011	0	1	0	0
100	0	0	1	0
101	0	0	1	0
110	0	0	0	1
111	0	0	0	1

All lower bound techniques we consider are based on some algebraic or combinatorial properties of communication matrices. Let P be a protocol computing a function $f : U \times V \to \{0,1\}$. As already mentioned above, any row of M_f (an element of U) corresponds to an input of C_I and any input of C_{II} (an element of V) corresponds to a column of M_f. So, the work of P is the game where C_I knows M_f and the row of M_f corresponding to its input and C_{II} knows M_f and the column corresponding to its input. The goal is to determine the value on the intersection between this row and this column. Note that C_I and C_{II} do not necessarily need to determine the exact position of the intersection of their row and column, because if the actual row (seen by C_I) contains only 0's, then C_I knows already that the output value is 0. The communication between C_I and C_{II} may be viewed as taking smaller and smaller submatrices of M_f until one obtains a monochromatic submatrix of M_f. To support this intuition consider that C_I sends the first message $c_1 \in \{0,1\}^+$ to C_{II}. Because C_{II} knows the communication protocol, it knows the set $S(c_1)$ of rows for which C_I sends the message c_1. So, C_{II} knows that the output value lies in the intersection of its column and the rows of $S(c_1)$. So, the first communication of this game reduces M_f to a submatrix $M_f(c_1)$ of M_f consisting of the rows of $S(c_1)$. Next, C_{II} sends a message c_2 to C_I. Since C_I knows the protocol and the messages c_1 and c_2, it can determine for which columns C_{II} sends c_2 if receiving c_1. So, the matrix $M_f(c_1\$c_2)$ remaining in the game becomes smaller again. Following this game one may suggest that every computation of P determines a set of inputs from $U \times V$, that corresponds to a monochromatic submatrix of M_f. So, all computations of any protocol P for f partition M_f into monochromatic submatrices. We confirm this intuition by using the following fundamental observation.

Observation 1 *Let $f : U \times V \to \{0,1\}$ be a function, and let P be a protocol. For all $\alpha_1, \alpha_2 \in U$, $\beta_1, \beta_2 \in V$, if P has the same computation C on the inputs (α_1, β_1) and (α_2, β_2), then P has this computation C on inputs (α_1, β_2) and (α_2, β_1), too.*

Particularly it means, that if P accept [rejects] (α_1, β_1) and (α_2, β_2), then P accepts [rejects] (α_1, β_2) and (α_2, β_1), too.

Observation 1 provides two powerful techniques for proving lower bounds on communication complexity. Figure 1 gives a transparent explanation of the following claim.

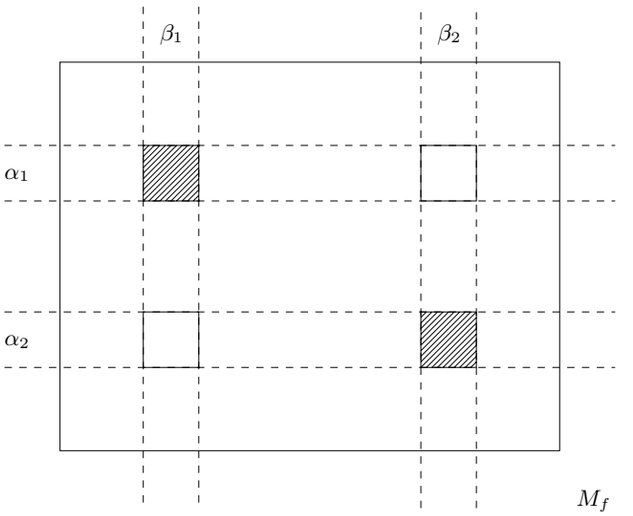

Fig. 1.

Corollary 1. *Let $f : U \times V \to \{0,1\}$ be a function that is computed by a protocol P. Let, for every computation C of P, $\mathbf{Set(C)}$ be the set of all inputs from $U \times V$ that are proceeded by C. Then, Set(C) corresponds to a monochromatic submatrix $\boldsymbol{M_f(C)}$ of M_f and $M_f(C)$ is a 1-monochromatic [0-monochromatic] submatrix of M_f iff C is an accepting [rejecting] computation.*

	000	001	010	011	100	101	110	111
000	0	1	1	0	1	0	0	1
001	1	0	0	1	0	1	1	0
010	1	0	0	1	0	1	1	0
011	0	1	1	0	1	0	0	1
100	1	0	0	1	0	1	1	0
101	0	1	1	0	1	0	0	1
110	0	1	1	0	1	0	0	1
111	1	0	0	1	0	1	1	0

Fig. 2.

Figure 2 depicts a 1-monochromatic submatrix that is the intersection of rows 001, 010, 100, 111 and the columns 000, 011, 101, and 110 of the communication matrix M_f for the function $f : \{0,1\}^3 \times \{0,1\}^3 \to \{0,1\}$ defined by

$$f(x_1, x_2, x_3, y_1, y_2, y_3) = x_1 \oplus x_2 \oplus x_3 \oplus y_1 \oplus y_2 \oplus y_3.$$

The computation corresponding to this 1-monochromatic submatrix is involved in the protocol P with $P(x_1x_2x_3) = x_1 \oplus x_2 \oplus x_3$ and $P(y_1y_2y_3, a\$) =$ accept [reject] iff $y_1 \oplus y_2 \oplus y_3 \oplus a = 1[0]$.

Corollary 1 immediately implies that every protocol P computing a function f with d different computations unambiguously determines a partition of M_f into d pair-wise disjoint monochromatic submatrices.[5] So, the number of computations of any protocol computing a function f is at least the cardinality of the minimal cover of M_f by pair-wise disjoint monochromatic submatrices. Let us formalize this idea in what follows.

Definition 6. *Let M be a Boolean matrix and let $S = \{M_1, \ldots, M_k\}$ be a set of monochromatic submatrices of M. We say that* $\boldsymbol{S}$ **is a cover of** $\boldsymbol{M}$ *if, for every element a_{ij} of M, there exists an $m \in \{1, \ldots, k\}$ such that a_{ij} is an element of M_m. We say that* $\boldsymbol{S}$ **is an exact cover of** $\boldsymbol{M}$ *if S is a cover of M and $M_r \cap M_s = \emptyset$ for every $r \neq s$, $r, s \in \{1, \ldots, k\}$.*

The **tiling complexity of** $\boldsymbol{M}$ *is*

$$\textbf{Tiling}(\boldsymbol{M}) = \min\{|S| \mid S \text{ is an exact cover of } M\}.$$

Theorem 2. *For every finite function $f : U \times V \to \{0, 1\}$,*

$$\mathrm{cc}(f) \geq \lceil \log_2 \lceil \mathrm{Tiling}(M_f) \rceil \rceil - 1.$$

Proof. Above we have proved that every protocol P for f determines an exact cover S of M_f, where $|S|$ is the number of different computations of P. So, $\mathrm{Tiling}(M_f)$ is a lower bound on the number of computations of any protocol computing f. Because of the prefix-freeness property of the protocols, every protocol for f must have a computation C of length at least $\log_2 \lceil \mathrm{Tiling}(M_f) \rceil$. So, $|\mathrm{Com}(C)| \geq \log_2 \lceil \mathrm{Tiling}(M_f) \rceil - 1$ and $\mathrm{cc}(f) \geq \log_2 \lceil \mathrm{Tiling}(M_f) \rceil - 1$. □

One can observe that the matrix M_f depicted at Figure 2 can be exactly covered by two 1-monochromatic submatrices

$$M_f(\{001, 010, 100, 111\}, \{000, 011, 101, 110\}),$$

$$M_f(\{000, 011, 101, 110\}, \{001, 010, 100, 111\}),$$

and by two 0-monochromatic submatrices

$$M_f(\{000, 011, 101, 110\}, \{000, 011, 101, 110\}),$$

and

$$M_f(\{001, 010, 100, 111\}, \{001, 010, 100, 111\}).$$

So, the communication protocol sending the bit $x_1 \oplus x_2 \oplus x_3$ for an input $x_1x_2x_3$ is an optimal one and $\mathrm{cc}(f) = 1$ for the function f whose matrix M_f is depicted in Figure 2.

[5] Note, that a protocol computing f must have exactly one computation for every entry of the matrix M_f.

A simple transparent example of the application of the tiling method is proving the optimal lower bound on $\mathrm{cc}(\mathrm{Eq}_n)$. Since M_{Eq_n} is the diagonal $2^n \times 2^n$ matrix, one needs 2^n 1-monochromatic submatrices of M_{Eq_n} to cover the diagonal elements (1's) and 2^n 0-monochromatic submatrices to cover all 0's of M_{Eq_n}. So, $\mathrm{Tiling}(M_{\mathrm{Eq}_n}) \geq 2^{n+1}$, and $\mathrm{cc}(\mathrm{Eq}_n) \geq n$.

Note, that in general, to find an optimal exact cover for a given Boolean matrix is a nontrivial combinatorial task and so in some cases the search for $\mathrm{Tiling}(M_f)$ is no simplification of the search for the communication complexity of an optimal protocol f.

The next method may be very practical in some cases because it is a constructive method. To get a lower bound on $\mathrm{cc}(f)$ it is sufficient to find a set of input instances with some special property. This method is also based on Observation 1. We search for such a set of inputs that any two inputs from this set cannot be covered by the same monochromatic submatrix. So, the cardinality of every exact cover must be at least the cardinality of this set. In this relation this method can be viewed as a constructive method for proving lower bounds on $\mathrm{Tiling}(M_f)$, too.

Definition 7. *Let $f : U \times V \to \{0,1\}$ be a function. For every $\delta \in \{0,1\}$, we say that a set $\mathcal{A} \subseteq U \times V$ is a* ***δ-fooling set for f*** *if, for all $(\alpha_1, \beta_1), (\alpha_2, \beta_2) \in \mathcal{A}$,*

(i) $f(\alpha_1, \beta_1) = f(\alpha_2, \beta_2) = \delta$, and
(ii) $f(\alpha_1, \beta_2) \neq \delta$ or $f(\alpha_2, \beta_1) \neq \delta$.

We define

$$\mathbf{Fool}(f) = \max\{|\mathcal{A}| \mid \mathcal{A} \text{ is } \delta\text{-fooling set for } f \text{ for some } \delta \in \{0,1\}\}.$$

Looking at Figure 2 we see that $\mathcal{A} = \{(001, 011), (011, 100)\}$ is a 1-fooling set because $f(001, 011) = f(011, 100) = 1$ and $f(001, 100) = f(011, 011) = 0$. So, there does not exist any 1-monochromatic submatrix of M_f that would cover both elements of $\mathcal{A}$.

Theorem 3. *For every finite function $f : U \times V \to \{0,1\}$,*

$$\mathrm{cc}(f) \geq \lceil \log_2(\mathrm{Fool}(f)) \rceil.$$

Proof. Let $\mathcal{A}$ be a δ-fooling set for f. Following Figure 1 or our consideration above, no pair of inputs $(\alpha_1, \beta_1), (\alpha_2, \beta_2)$ of $\mathcal{A}$ can be covered by one δ-monochromatic submatrix. So, only to cover the elements of $\mathcal{A}$ in M_f, one needs at least $|\mathcal{A}|$ δ-monochromatic submatrices. This is the same as to say that, for $\delta = 1[0]$, every protocol for f must have at least $|\mathcal{A}|$ different accepting [rejecting] computations. But $|\mathcal{A}|$ different accepting [rejecting] computations mean $|\mathcal{A}|$ different communications. As already observed, every protocol with $|\mathcal{A}|$ different communications must have a communication of length at least $\lceil \log_2 |\mathcal{A}| \rceil$. Thus

$$\mathrm{cc}(f) \geq \lceil \log_2 |\mathcal{A}| \rceil$$

for every δ-fooling set $\mathcal{A}$ for f. □

The third method for proving lower bounds on communication complexity is based on the rank of communication matrices. Let, for any field F with identity elements 0 and 1 and any Boolean matrix M, $\mathbf{rank_F(M)}$ denote the rank of the matrix M over the field F. $\mathbf{rank(M)}$= $\mathrm{rank}_{\mathrm{GF}(2)}(M)$ denotes the rank of M over the Galois field GF(2), that contains only the elements 0 and 1. We define, for every Boolean matrix M,

$$\mathbf{Rank(M)} = \max\{\mathrm{rank}_F(M) \mid F \text{ is a field with identity elements 0 and 1}\}.$$

Observation 2 *For every Boolean matrix M, $\mathrm{Rank}(M) = \mathrm{rank}_Q(M)$, where Q is the set of all rational numbers.*

Theorem 4. *For every finite function $f : U \times V \to \{0,1\}$,*

$$\mathrm{cc}(f) \geq \lceil \log_2(\mathrm{Rank}(M_f)) \rceil.$$

Proof. Let P be a protocol computing f, and let F be any field with identity elements 0 and 1. It is sufficient to show that the number of accepting computations of P is at least $\mathrm{rank}_F(M_f)$.

Let $C_1, C_2, \ldots, C_k$ be all accepting computations of P. We already know, that, for every $i \in \{1, 2, \ldots, k\}$, C_i determines the 1-monochromatic submatrix $M_f(C_i)$ of M_f. Let $\bar{M}_f(C_i) = [a_{rs}]$ be the $|U| \times |V|$ Boolean matrix defined by

$$a_{rs} = 1 \text{ iff } a_{rs} \text{ is an element of } M_f(C_i)$$

for $i = 1, \ldots, n$. Obviously

$$M_f = \sum_{i=1}^{k} \bar{M}_f(C_i).$$

By the properties of the rank over any field we have

$$\mathrm{rank}_F(M_f) \leq \sum_{i=1}^{k} \mathrm{rank}_F(\bar{M}_f(C_i)).$$

Thus, $k \geq \mathrm{rank}_F(M_f)$ and to get k accepting computations P must contain a communication of at least $\log_2 k$ bits. □

Now, we look for lower bounds on one-way communication complexity. Here, the situation is quite simple.

Definition 8. *Let $f : U \times V \to \{0,1\}$ be a finite function. A set $\mathcal{A} \subseteq U$ is called a* **one-way fooling set for f**, *if, for every $\alpha, \beta \in \mathcal{A}$, $\alpha \neq \beta$, there exists a $\gamma \in V$ such that*

$$f(\alpha, \gamma) \neq f(\beta, \gamma).$$

We define

$$\textbf{One-Way-Fool}(f) = \max\{|\mathcal{A}| \mid \mathcal{A} \textit{ is a one-way fooling set for } f\}.$$

For every Boolean matrix M, we define $\textbf{Row}(M)$ *to be the number of different rows of M.*

Theorem 5. *For every finite function $f : U \times V \to \{0,1\}$,*

$$2^{\mathrm{cc}_1(f)} \geq \text{One-Way-Fool}(f) = \text{Row}(M_f).$$

Proof. One can easily observe that two rows α and β of M_f are different if and only if there is a column γ such that $f(\alpha,\gamma) \neq f(\beta,\gamma)$. So, One-Way-Fool($f$) = Row($M_f$) because every one-way fooling set $\mathcal{A}$ for f determines $|\mathcal{A}|$ different rows of M_f and, vice versa, any k different rows of M_f determine a one-way fooling set of the cardinality k.

Let P be an one-way protocol for f. Let $P(\alpha) = P(\beta)$ for some $\alpha, \beta \in U$, $\alpha \neq \beta$ (i.e. C_I sends the same message to C_{II} for both α and β). Then, for every $\gamma \in V$, P must compute the same output for both inputs (α,γ) and (β,γ) because $P(\gamma, P(\alpha)\$) = P(\gamma, P(\beta)\$)$. So, $P(\alpha) = P(\beta)$ can be true only if the rows α and β are equal. Thus, the number of different messages (communications) of P must be at least Row(M_f). □

We can immediately observe that there may exist an exponential gap between $\mathrm{cc}_1(f)$ and $\mathrm{cc}(f)$ because there are Boolean matrices with an exponential gap between their Row(M) and Rank(M). For instance, for any positive integer n, one can construct a $2^n \times n$ Boolean matrix M consisting of 2^n different rows, i.e. containing all n-dimensional Boolean vectors as the rows. Obviously, Row(M) $= 2^n$ and Rank(M) $= n$.

To see the difference for a function from $\{0,1\}^n \times \{0,1\}^n$ to $\{0,1\}$ consider the function $f_{\mathrm{ind}(n)}$ introduced in Section 2.

Lemma 1. *For every positive integer $n = 2^k$, $k \in \mathbb{N} - \{0\}$,*

$$\mathrm{cc}_1(f_{\mathrm{ind}(n)}) = n \textit{ and } \mathrm{cc}(f_{\mathrm{ind}(n)}) \leq 2 \cdot \log_2 n.$$

Proof. A protocol computing $f_{\mathrm{ind}(n)}$ within the communication complexity $2 \cdot \log_2 n$ was presented in Section 2. The fact $\mathrm{cc}_1(f_{\mathrm{ind}(n)}) \leq n$ is obvious.

It remains to show $\mathrm{cc}_1(f_{\mathrm{ind}(n)}) \geq n$. To do this observe that the $2^n \times 2^n$ matrix $M_{f_{\mathrm{ind}(n)}}$ has 2^n different rows. Let $\alpha = \alpha_1 \ldots \alpha_n \in \{0,1\}^n$, $\beta = \beta_1 \ldots \beta_n \in \{0,1\}^n$, $\alpha \neq \beta$ be two input parts determining two rows of $M_{f_{\mathrm{ind}(n)}}$. Since $\alpha \neq \beta$, there exists an integer $j \in \{1,\ldots,n\}$ such that $\alpha_j \neq \beta_j$. Without loss of generality assume $\alpha_j = 1$ and $\beta_j = 0$. Let j be the smallest integer with the above property. We distinguish two possibilities according to whether $j \geq \log_2 n$ or $j \leq \log_2 n$.

If $j > \log_2 n$, $\alpha_1 \ldots \alpha_{\log_2 n} = \beta_1 \ldots \beta_{\log_2 n}$, and so

$$a = \text{Number}(\alpha_1 \ldots \alpha_{\log_2 n}) + 1 = \text{Number}(\beta_1 \ldots \beta_{\log_2 n}) + 1.$$

We choose an $\gamma = \gamma_1 \dots \gamma_n$ such that

$$\text{Number}(\gamma_a \gamma_{(a+1) \bmod n} \cdots \gamma_{(a+\lceil \log_2 n \rceil) \bmod n}) + 1 = j.$$

Obviously $f(\alpha, \gamma) = 1$ and $f(\beta, \gamma) = 0$ and so the rows corresponding to α and β are different.

The proof for the case $j \leq \log_2 n$ is left to the reader. □

Finally, we deal with proving lower bounds on nondeterministic communication complexity. First of all we observe that Observation 1 holds for nondeterministic protocols, too. This means, that if C is an accepting computation on inputs (α_1, β_1) and (α_2, β_2), then C is a possible accepting computation on the inputs (α_1, β_2) and (α_2, β_1). Thus we have again the situation that every accepting computation corresponds to a 1-monochromatic submatrix. The situation differs from the deterministic case in that the nondeterministic protocol has possibly several computations on one input and so the monochromatic matrices determined by computations may overlap. Thus, any nondeterministic protocol determines a cover of 1's of the communication matrix by possibly non-disjoint 1-monochromatic submatrices. Observe that a nondeterministic protocol does not necessarily determine any cover of 0's of M_f.

Definition 9. *Let M be a Boolean matrix, and let $S = \{M_1, \dots, M_k\}$ be a set of 1-monochromatic submatrices of M. We say that S is a 1-cover of M if every 1 of M is contained in at least one of the 1-submatrices of S. We define*

$$\textbf{Cover}(\boldsymbol{M}) = \min\{|S| \mid S \text{ is a 1-cover of } M\}.$$

Theorem 6. *For every finite function $f : U \times V \to \{0, 1\}$,*

$$\text{ncc}(f) = \lceil \log_2(\text{Cover}(M_f)) \rceil.$$

Proof. We have already argued that $\text{ncc}(f) \geq \lceil \log_2(\text{Cover}(M_f)) \rceil$ because every nondeterministic protocol with k accepting computations determines a 1-cover of M_f of size k.

Thus, it remains to prove $\text{ncc}(f) \leq \lceil \log_2(\text{Cover}(M_f)) \rceil$. Let $S = \{M_1, M_2, \dots, M_k\}$ be a 1-cover of M_f, and let $k = \text{Cover}(M_f)$. We construct a nondeterministic one-way protocol $D = (C_I, C_{II})$ for f as follows. For every input $\alpha \in U$, C_I looks for all submatrices of S with non-empty intersections with row_α. C_I chooses one of these 1-submatrices M_i and sends the message $\text{BIN}_{\lceil \log_2 k \rceil}(i)$. If C_{II} receives $\text{BIN}_{\lceil \log_2 k \rceil}(i)$ and column_β of the input $\beta \in V$ of C_{II} intersect M_i, then C_{II} accepts. Otherwise, C_{II} rejects. □

Observe, that the proof of Theorem 6 provides also an alternative proof of the fact $\text{ncc}(f) = \text{ncc}_1(f)$ for every f. Since the cardinality of any 1-fooling set for f is a lower bound[6] on $\text{Cover}(M_f)$, we obtain the following result.

[6] Remember that no pair of elements of a 1-fooling set can be covered by one 1-monochromatic submatrix.

Theorem 7. *Let $f : U \times V \to \{0,1\}$ be a finite function. For every 1-fooling set $\mathcal{A}$ for f,*

$$\mathrm{ncc}(f) \geq \lceil \log_2 |\mathcal{A}| \rceil.$$

We see that the function Eq_n is hard for nondeterministic protocols because one needs 2^n 1-monochromatic submatrices to cover the 2^n diagonal elements of the $2^n \times 2^n$ diagonal matrix M_{Eq_n}.

For Ineq_n the situation essentially differs. The ones of an $2^n \times 2^n$ 0-diagonal matrix can be covered by $2n$ 1-monochromatic submatrices

$$M_1, M_2, \ldots, M_n, M'_1, M'_2, \ldots, M'_n,$$

where M_i (M'_i) is the intersection of all rows whose i-th bit is 1 (0) with rows whose i-th bit is 0 (1). On the other hand to cover 0's of M_{Ineq_n} one needs 2^n 0-monochromatic matrices. So, this implies the following exponential gap between deterministic communication complexity and nondeterministic communication complexity.

Theorem 8. *For every positive integer n*

(i) $\mathrm{ncc}(\mathrm{Ineq}_n) \leq \lceil 2 \log_2 n \rceil$, *and*
(ii) $\mathrm{cc}(\mathrm{Ineq}_n) = n$.

4 Randomized Protocols

First, we relate Las Vegas and determinism for communication complexity. The results show that Las Vegas can be more efficient than determinism, but the difference between the power of these two models of computation is not very large. One conjectures that this kind of relation should hold for several fundamental computing models and their complexity measures, including the time complexity of Turing machines. Unfortunately, one fails in the effort to establish such relations for most complexity measures investigated. In this section we show that there is an at most quadratic gap between $\mathrm{cc}(f)$ and $\mathrm{lvcc}(f)$ and a linear relation between one-way communication complexity and Las Vegas one-way communication complexity. Examples of concrete functions show that Las Vegas can do something better than determinism in the framework of relations established above.

First of all we show that there is a polynomial relation between Las Vegas communication complexity and deterministic communication complexity. To get this result one proves a more powerful result claiming that if both a function f and its complement $\overline{f}$ are easy for a nondeterministic protocol, then f is easy for a deterministic protocol, too. In what follows, for every function $f : U \times V \to \{0,1\}$, the **complement of f** is the function $\overline{f}$ defined by

$$\overline{f}(\alpha, \beta) = \Gamma(f(\alpha, \beta))$$

for all $(\alpha, \beta) \in U \times V$. Γ denotes the unary Boolean operation called **negation**.

Theorem 9 ([20]). *For every finite function $f : U \times V \to \{0,1\}$,*

$$\mathrm{cc}(f) \leq \mathrm{ncc}(f) \cdot (\mathrm{ncc}(\overline{f}) + 2).$$

Proof. Let $r_1 = \mathrm{ncc}(f)$ and $r_0 = \mathrm{ncc}(\overline{f})$. This directly implies that the 1's of M_f can be covered by at most 2^{r_1} 1-monochromatic submatrices, and that the 0's of M_f can be covered by at most 2^{r_0} 0-monochromatic submatrices. We shall prove that $\mathrm{cc}(f) \leq r_1 \cdot (r_0 + 2)$.

The proof relies on the following property of monochromatic submatrices. Let $M(R,S)$ be a 0-monochromatic submatrix, and let $M(S',R')$ be a 1-monochromatic submatrix of M_f. Then either $R \cap R' = \emptyset$ or $S \cap S' = \emptyset$. The proof for this claim is straightforward because if $R \cap R' \neq \emptyset$ and $S \cap S' \neq \emptyset$, then there exists a pair $(\alpha,\beta) \in (R \cap R') \times (S \cap S')$, i.e. a pair that belongs to both $M(R,S)$ and $M(R',S')$. However, this is impossible because (α,β) in $M(R,S)$ implies $f(\alpha,\beta) = 0$, whereas (α,β) in $M(R',S')$ implies $f(\alpha,\beta) = 1$.

Before starting the proper proof we fix some useful notation. Let $\overline{C} = \{C_1,\ldots,C_m\}$, $m \leq 2^{r_0}$, be a set of 0-monochromatic submatrices of M_f such that $\overline{C}$ covers all 0's of M_f. Let $\overline{H} = \{H_1,\ldots,H_l\}$, $l \leq 2^{r_1}$, be a set of 1-monochromatic submatrices of M_f covering all 1's in M_f. Let A_i denote the submatrix of M_f formed by those rows of M_f that are contained in C_i (i.e., if $C_i = M(R,S)$, then $A_i = M(R,V)$). Let B_i denote the submatrix of M_f formed by the columns of M_f that meet C_i (i.e., if $C_i = M(R,S)$, then $B_i = M(U,S)$). Let $\mathbf{int}(\boldsymbol{A_i})$ and $\mathbf{int}(\boldsymbol{B_i})$ respectively denote the number of 1-monochromatic submatrices from $\overline{H}$ that have a non-empty intersection with A_i and B_i respectively. Since the intersection of A_i and B_i is exactly the 0-monochromatic submatrix C_i, the claim stated above implies that no matrix $H_j \in \overline{H}$ has non-empty intersections with both A_i and B_i. So,

$$\mathrm{int}(A_i) + \mathrm{int}(B_i) \leq l = |\overline{H}|$$

for every $i \in \{1,2,\ldots,m\}$. Set

$\overline{C}_1 = \{C_k \in \overline{C} \mid \mathrm{int}(A_k) \leq \lceil l/2 \rceil\}$, and
$\overline{C}_2 = \{C_s \in \overline{C} \mid \mathrm{int}(B_s) \leq l/2\} = \overline{C} - \overline{C}_1$.

Now, we describe the first two rounds of a deterministic protocol $D = \langle D_I, D_{II} \rangle$ for f. For every input $(\alpha,\beta) \in U \times V$, D works as follows.

Round 1. D_I looks on the row of M_f that corresponds to α in order to see whether it intersects any of the 0-monochromatic submatrices in $\overline{C}_1$. If so, it sends the message "1BIN$_{r_0}(j)$", where j is the smallest index such that $C_j \in \overline{C}_1$ and C_j intersects the row of α. If row$_\alpha$ does not intersect any submatrix of $\overline{C}_1$, then D_I sends the message "0".

Round 2. If D_{II} receives "0", it looks whether the column corresponding to its input β intersects any of the 0-monochromatic submatrices in $\overline{C}_2$. If so, it sends the message "1BIN$_{r_0}(k)$", where k is the smallest index such that $C_k \in \overline{C}_2$ and C_k intersects the column of β.
Otherwise, D_{II} sends the message "0".
If D_{II} receives "1BIN(j)", then it sends "1" to D_I.

Now, let us distinguish and discuss three possible situations after the first two rounds of D.

Case 1. The current communication history is 0\$0\$, i.e. both computers failed to find an appropriate 0-monochromatic submatrix. Since $\overline{C}_1 \cup \overline{C}_2 = \overline{C}$ and the set $\overline{C}$ covers all zeros in M_f, we get $f(\alpha, \beta) = 1$. So, both D_I and D_{II} know that the output has to be "accept".

Case 2. The current communication history is $1\text{BIN}_{r_0}(j)\$1\$$. In this case both D_I and D_{II} know, that the input (α, β) belongs to A_j. Since $C_j \in \overline{C}_1$, $\text{int}(A_j) \leq \lceil l/2 \rceil \leq 2^{r_1-1}$, i.e., all ones in A_j can be covered by at most $2^{r_1-1} = 2^{r_1}/2$ 1-monochromatic submatrices of A_k. (Note that these 1-monochromatic submatrices are all intersections of A_k with 1-monochromatic submatrices $H_1, H_2, \ldots, H_l$ of M_f.)

Case 3. The current communication history is $0\$\text{BIN}_{r_0}(k)\$$. In this case both D_I and D_{II} know that the input (α, β) lies in B_k. Since $C_k \in \overline{C}_2$, they know that $\text{int}(B_k) \leq l/2 \leq 2^{r_1-1}$, i.e., all ones in B_k can be covered by at most 2^{r_1-1} 1-monochromatic submatrices.

Thus, after this first two rounds either both computers know the output value $f(\alpha, \beta)$ or both D_I and D_{II} know that (α, β) lies in a matrix $M_1 \in \{A_k, B_m\}$ whose ones can be covered by at most 2^{r_1-1} 1-monochromatic submatrices of M_1, and whose zeros can be covered by at most 2^{r_0} 0-monochromatic submatrices. Following the same communication strategy as described above for M_f in the next two rounds for the matrix M_1, D_I and D_{II} either learn the result $f(\alpha, \beta)$ or they agree that (α, β) is in a submatrix M_2 such that

(i) all ones of M_2 can be covered by at most 2^{r_1-2} 1-monochromatic submatrices of M_2, and
(ii) all zeros of M_2 can be covered by at most 2^{r_0} 0-monochromatic submatrices of M_2.

Continuing in this way, D_I and D_{II} learn $f(\alpha, \beta)$ after at most r_1 rounds. Since every information exchange in rounds $2i$ and $2i+1$ has the length $2 + r_0$, $\text{cc}(D) \leq r_1 \cdot (2 + r_0)$. □

Obviously, $\text{lvcc}(f) = \text{lvcc}(\overline{f})$ for every function f. Since $\text{ncc}(f)$ is a lower bound on the private Las Vegas communication complexity, there is at most an quadratic gap between determinism and private Las Vegas communication protocols. Since the relation between the private randomized communication complexity and the public one is linear if the randomized communication complexity is at least $\Omega(\log_2 n)$ for Boolean functions of n variables[7], we can express the consequences of Theorem 9 in terms of public Las Vegas communication complexity as follows.

Theorem 10. *For every Boolean function* $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$,

$$\text{cc}(f) = O\left((\text{lvcc}(f))^2 + \log_2 n\right).$$

[7] See Section 6 for the exact formulation of the relation and its proof.

Our next aim is to show, that the at most quadratic gap between determinism and Las Vegas presented in Theorem 10 can be achieved for a concrete Boolean function. Let, for every positive integer n, $n = m^2$,

$$\begin{aligned}\mathrm{ExAll}_{2n}(&x_{1,1}, x_{1,2}, \ldots, x_{1,m}, \ldots, x_{m,1}, x_{m,2}, \ldots, x_{m,m},\\ &y_{1,1}, y_{1,2}, \ldots, y_{1,m}, \ldots, y_{m,1}, y_{m,2}, \ldots, y_{m,m})\end{aligned}$$

be a Boolean function from $\{0,1\}^n \times \{0,1\}^n$ to $\{0,1\}$ defined by

$$\mathrm{ExAll}_{2n}(\alpha_{1,1}, \ldots, \alpha_{m,m}, \beta_{1,1}, \ldots, \beta_{m,m}) = 1$$

$$\text{iff } \exists j \in \{1, \ldots, m\} \text{ such that } \alpha_{j,1}, \ldots, \alpha_{j,m} = \beta_{j,1}, \ldots, \beta_{j,m}.$$

Theorem 11 ([21]). *For every positive integers n, m, $n = m^2$,*

(i) $\mathrm{cc}(\mathrm{ExAll}_{2n}) = n = m^2$, *and*
(ii) $\mathrm{lvcc}(\mathrm{ExAll}_{2n}) \leq 2m(\lceil \log_2 m \rceil^2 + 1)$ *for sufficiently large m.*

Proof.

(i) The fact $\mathrm{cc}(\mathrm{ExAll}_{2n}) \leq n$ is obvious. To prove the lower bound it is sufficient to prove $\mathrm{cc}(\overline{\mathrm{ExAll}_{2n}}) \geq n$. Note that

$$\overline{\mathrm{ExAll}_{2n}}(\alpha_{1,1}, \ldots, \alpha_{m,m}, \beta_{1,1}, \ldots, \beta_{m,m}) = 1$$

iff, for every $i \in \{1, \ldots, m\}$, $\alpha_{i,1}, \ldots, \alpha_{i,m} \neq \beta_{i,1}, \ldots, \beta_{i,m}$. To prove the lower bound we use the rank method. More precisely, we show that the rows of the $2^n \times 2^n$ diagonal matrix are linear combinations of the row of $M_{\overline{\mathrm{ExAll}_{2n}}}$ and so $\mathrm{rank}(M_{\overline{\mathrm{ExAll}_{2n}}}) = 2^n$. Consider the function $\mathrm{q}_{2n} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ defined for all $\delta_1, \ldots, \delta_m, \gamma_1, \ldots, \gamma_m \in \{0,1\}^m$ as follows:

$$\mathrm{q}_{2n}(\delta_1\delta_2 \ldots \delta_m, \gamma_1\gamma_2 \ldots \gamma_m) =$$

$$\sum_{\alpha_1 \neq \delta_1} \sum_{\alpha_2 \neq \delta_2} \cdots \sum_{\alpha_m \neq \delta_m} \overline{\mathrm{ExAll}_{2n}}(\alpha_1\alpha_2 \ldots \alpha_m, \gamma_1\gamma_2 \ldots \gamma_m) \bmod 2,$$

where $\alpha_i \in \{0,1\}^m$ for $i = 1, 2, \ldots, m$. Our aim is to show that

$$\mathrm{q}_{2n}(\alpha, \beta) = \mathrm{Eq}_n(\alpha, \beta)$$

for every positive integer n and all $(\alpha, \beta) \in \{0,1\}^n \times \{0,1\}^n$. Let us prove this. First, consider inputs $(\delta, \delta) = (\delta_1\delta_2 \ldots \delta_m, \delta_1\delta_2 \ldots \delta_m)$ for any $\delta_i \in \{0,1\}^m$ for $i = 1, \ldots, m$.

$$\mathrm{q}_{2n}(\delta, \delta) = \mathrm{q}_{2n}(\delta_1\delta_2 \ldots \delta_m, \delta_1\delta_2 \ldots \delta_m) = (2^m - 1)^m \bmod 2 = 1$$

because there are exactly $2^m - 1$ words from $\{0,1\}^m$ different from δ_i and $\overline{\mathrm{ExAll}_{2n}}(w, \delta_1, \ldots, \delta_m) = 1$ for all

$$w \in \{w_1w_2 \ldots w_m \mid w_i \in \{0,1\}^m - \{\delta_i\} \text{ for } i = 1, \ldots, m\}.$$

Now, consider inputs $(\delta, \gamma) = (\delta_1\delta_2 \dots \delta_m, \gamma_1\gamma_2 \dots \gamma_m)$, $\delta_i, \gamma_i \in \{0,1\}^m$ for $i = 1, \dots, m$, with $\delta \neq \gamma$. Let $S(\delta, \gamma) \subseteq \{1, \dots, m\}$ such that, for all $j \in S(\delta, \gamma)$, $\delta_j \neq \gamma_j$, and for every $k \in \{1, \dots, n\} - S(\delta, \gamma)$, $\delta_k = \gamma_k$. Let $|S(\delta, \gamma)| = r$. We know that $r \geq 1$. Then

$$\mathrm{q}_{2n}(\delta, \gamma) = (2^m - 2)^r \cdot (2^m - 1)^{m-r} \bmod 2 = 0.$$

We proved that the communication matrix $M_{\mathrm{q}_{2n}}$ is the diagonal matrix of size $2^n \times 2^n$, and so $\mathrm{rank}(M_{\mathrm{q}_{2n}}) = 2^n$. Following the definition of q_{2n} we see that every row of $M_{\mathrm{q}_{2n}}$ is a linear combination of rows of $M_{\overline{\mathrm{ExAll}_{2n}}}$. Thus,

$$\mathrm{rank}(M_{\overline{\mathrm{ExAll}_{2n}}}) \geq \mathrm{rank}(M_{\mathrm{q}_{2n}}) = 2^n.$$

(ii) First, we describe a randomized protocol $D = (D_I, D_{II})$ for ExAll_{2n} and then we analyze its communication complexity. Let $\mathrm{Prim}_m = \{p \in I\!N \mid p \leq m \text{ and } p \text{ is a prime}\}$.

Protocol D

Input: $(\alpha, \beta) = (\alpha_1\alpha_2 \dots \alpha_m, \beta_1\beta_2 \dots \beta_m)$, $\alpha_i, \beta_i \in \{0,1\}^m$ for $i = 1, 2, \dots, m$.

Step 1. $d = 2 \cdot \lceil \log_2 m \rceil$ prime numbers $s_1, s_2, \dots, s_d$ are uniformly chosen from Prim_m at random.
{ Thus, both D_I and D_{II} know $s_1, s_2, \dots, s_d$.}

Step 2. The first computer D_I computes $\mathrm{Number}(\alpha_i) \bmod s_j$ for all $i \in \{1, \dots, m\}$ and all $j \in \{1, \dots, d\}$ and sends the binary representation of all $m \cdot d$ results to D_{II}.

Step 3. If, for every $i \in \{1, \dots, m\}$, there exists j_i such that

$$\mathrm{Number}(\alpha_i) \bmod s_{j_i} \neq \mathrm{Number}(\beta_i) \bmod s_{j_i},$$

then D_{II} outputs "reject".
Else, let $k \in \{1, \dots, m\}$ be the smallest integer such that

$$\mathrm{Number}(\alpha_k) \bmod s_j = \mathrm{Number}(\beta_k) \bmod s_j$$

for all $j \in \{1, 2, \dots, d\}$. Then D_{II} sends the binary representation of k to D_I.

Step 4. When D_I receives the binary representation of an positive integer k, then D_I send α_k to D_{II}.

Step 5. Receiving α_k, D_{II} compares α_k with β_k.
If $\alpha_k = \beta_k$, then D_{II} outputs "accept".
If $\alpha_k \neq \beta_k$, then D_{II} outputs "?".

We again use the fact presented in Section 2 that, for all $\gamma, \delta \in \{0,1\}^m$ and sufficiently large m's, $\gamma \neq \delta$ implies

$$\left| \{p \in \mathrm{Prim}_m \mid \mathrm{Number}(\gamma) \bmod p \neq \mathrm{Number}(\delta) \bmod p\} \right| \geq \frac{|\mathrm{Prim}_m|}{2}.$$

Consider an input (α, β) such that $\text{ExAll}_{2n}(\alpha, \beta) = 0$. This means $\alpha = \alpha_1\alpha_2 \dots \alpha_m$, $\beta = \beta_1\beta_2 \dots \beta_m$ and $\alpha_i \neq \beta_i$ for $i = 1, \dots, m$. For every $i \in \{1, \dots, m\}$, the probability that

$$\text{Number}(\alpha_i) \bmod s_j = \text{Number}(\beta_i) \bmod s_j$$

for all $j \in \{1, \dots, d\}$ is 2^{-d}. So, the probability that D_{II} recognizes $\alpha_i \neq \beta_i$ in Step 3 is $1 - 2^{-d}$. This implies that the probability to recognize $\alpha_i \neq \beta_i$, for all $i \in \{1, \dots, m\}$ and so to reject (α, β) is at least $(1 - \frac{1}{2^d})^m = (1 - \frac{1}{m^2})^m > \frac{1}{2}$ for sufficiently large m. Obviously, if D_{II} does not reject (α, β) in Step 4, then D_{II} must output "?" in Step 5.

Now, consider $\text{ExAll}_{2n}(\alpha, \beta) = 1$, i.e. there exists a $k \in \{1, \dots, n\}$ such that $\alpha_k = \beta_k$. If there are several such numbers, let k be the smallest one with this property. Obviously,

$$\text{Number}(\alpha_k) \bmod s_j = \text{Number}(\beta_k) \bmod s_j$$

for all $j \in \{1, 2, \dots, m\}$ and so D_{II} cannot reject (α, β) in Step 3. If D_{II} sends the binary representation of k to D_I, then D_I sends α_k to D_{II} and D_{II} accepts the input. This scenario fails to work only if D_{II} sends $\text{BIN}_{\lceil \log_2 m \rceil}(j)$ to D_I for some $j < k$ despite of the fact that $\alpha_j \neq \beta_j$. In that case D_{II} outputs "?". But this can happen only with probability at most

$$\frac{1}{2^d} \cdot \left(1 - \frac{1}{2^d}\right)^{j-1} < \frac{1}{2^d} \cdot \left(1 - \frac{1}{m^2}\right)^m < \frac{1}{2^d} = \frac{1}{m^2}$$

for every $j \in \{1, \dots, m\}$. So,

$$\mathit{Prob}\,(D(\alpha, \beta) = \text{"?"}) \leq \sum_{l=1}^{k-1} \left(\frac{1}{2^d}\left(1 - \frac{1}{2^d}\right)^{l-1}\right) < \sum_{l=1}^{m} \frac{1}{m^2} = \frac{1}{m}.$$

Thus,

$$\mathit{Prob}\,(D(\alpha, \beta) = \text{"accept"}) > 1 - \frac{1}{m}.$$

We conclude that D is a Las Vegas protocol for ExAll_{2n}.

It remains to calculate the communication complexity of D. In Step 2, D_I sends $m \cdot d$ binary representations of length $\lceil \log_2 m \rceil$ and so the length of the first message is always exactly $m \cdot 2 \cdot \lceil \log_2 m \rceil^2$. If D_{II} sends a message to D_I, then this message has the length $\lceil \log_2 m \rceil$. Then in Step 4 D_I answers with a message of length m. So, the longest communication takes

$$2 \cdot m \cdot \lceil \log_2 m \rceil^2 + \lceil \log_2 m \rceil + m \leq 2m(\lceil \log_2 m \rceil^2 + 1)$$

bits. □

Observe that Theorem 8 and Theorem 9 together imply that there is an exponential gap between nondeterminism and Las Vegas for communication complexity.

The next question we consider in this section is whether the established relation between $\mathrm{cc}(f)$ and $\mathrm{lvcc}(f)$ is valid also for one-way communication protocols. Surprisingly, there is even a linear relation between $\mathrm{cc}_1(f)$ and $\mathrm{lvcc}_1(f)$, i.e. determinism is almost as powerful as Las Vegas for one-way protocols. We shall show nice applications of this tight relation between $\mathrm{cc}_1(f)$ and $\mathrm{lvcc}_1(f)$ for proving polynomial relations between determinism and Las Vegas for other computing models in Section 7.

Theorem 12 ([22]). *For every function $f : U \times V \to \{0,1\}$ with finite sets U and V,*

$$\mathrm{lvcc}_1(f) \geq \mathrm{cc}_1(f)/2.$$

We have shown that Las Vegas communication complexity is closely related to deterministic communication complexity. This is not true for Monte Carlo randomization. In this section we show that

(i) already one-sided-error Monte Carlo protocols may be much more powerful than deterministic ones, but the gap between one-sided-error Monte Carlo communication complexity and (deterministic) communication complexity can be at most exponential,
(ii) two-sided-error Monte Carlo protocols can be much more efficient than nondeterministic ones, and
(iii) nondeterminism can be much more powerful than Monte Carlo randomization.

We present the results in the sequence as written above. Observe that the power of one-sided-error Monte Carlo randomization lies between determinism and nondeterminism. Our first result shows that an exponential gap between determinism and one-sided-error Monte Carlo is possible for communication complexity.

Theorem 13 ([23]).

(i) For every positive integer n, $\mathrm{cc}(\mathrm{Ineq}_n) = n$, and
(ii) $1\mathrm{mccc}(\mathrm{Ineq}_n) \leq 2\lceil \log_2 n \rceil$ for sufficiently large integer n.

Proof.

(i) We showed already that all three lower bound techniques (Tiling, Rank, Fooling sets) provide $\mathrm{cc}(\mathrm{Ineq}_n) \geq n$.
(ii) In Example 1 we proved $1\mathrm{mccc}(\mathrm{Ineq}_n) \leq 2 \cdot \lceil \log_2 n \rceil$ for sufficiently large n.

□

A natural question is whether there is a possibility to bound the difference between $\mathrm{cc}(f)$ and $1\mathrm{mccc}(f)$ for all functions f (i.e., whether a larger gap than the exponential gap presented in Theorem 13 is possible). The following theorem shows that the gap cannot be larger than exponential because nondeterminism can be simulated by determinism with an exponential blow-up of communication complexity. Since one-sided-error Monte Carlo protocols can be viewed as restricted nondeterministic protocols, we are done.

Theorem 14 ([24]). *For every finite function $f : U \times V \to \{0,1\}$,*

$$\mathrm{cc}_1(f) \leq 2^{\mathrm{ncc}(f)}.$$

Proof. We know that $\mathrm{ncc}(f)$ is exactly the cardinality of an optimal cover of 1's of M_f by 1-monochromatic submatrices of M_f. Let $\{M_1, M_2, \ldots, M_{\mathrm{ncc}(f)}\}$ be such an optimal cover. One can construct a (deterministic) one-way protocol $P = (C_I, C_{II})$ that computes f as follows.

Input: A pair $(\alpha, \beta) \in U \times V$.
{ α is the input part of C_I, and β is the input part of C_{II}.}
Step 1. For every input $\alpha \in U$, C_I sends the message

$$d_1 d_2 \ldots d_{\mathrm{ncc}(f)} \in \{0,1\}^{\mathrm{ncc}(f)}$$

to C_{II}, where, for all $i \in \{1, \ldots, \mathrm{ncc}(f)\}$, $d_i = 1$ if row_α has a nonempty intersection with the 1-monochromatic submatrix M_i.
{ In this way C_I tells C_{II} the names of all 1-monochromatic submatrices that candidate to contain the element (α, β) from the C_I point of view.}
Step 2. After receiving $d_1 d_2 \ldots d_{\mathrm{ncc}(f)}$, C_{II} accepts, if there exists an $i \in \{1, \ldots, \mathrm{ncc}(f)\}$ such that $d_i = 1$ and column_β has a nonempty intersection with the 1-monochromatic submatrix M_i. Otherwise, C_{II} rejects.

Obviously, P computes f and its communication complexity is exactly $\mathrm{ncc}(f)$. □

The next result continues to demonstrate the power of Monte Carlo randomization by showing that two-sided-error Monte Carlo protocols may be much more efficient than their nondeterministic counterparts.

Theorem 15 ([23]). *For every positive integer n,*

(i) $\mathrm{ncc}(\mathrm{Eq}_n) = n$, *and*
(ii) $2\mathrm{mccc}(\mathrm{Eq}_n) \leq 2\lceil \log_2 n \rceil$ *for sufficiently large n.*

Proof.
(i) The fact $\mathrm{ncc}(\mathrm{Eq}_n) \leq n$ is obvious and the lower bound $\mathrm{ncc}(\mathrm{Eq}_n) \geq n$ was already observed as an application of the 1-fooling set technique and the cover technique in Section 3.
(ii) Consider the one-sided-error Monte Carlo protocol R of Example 1 for Ineq_n. If one modifies R to $\overline{R}$ in such a way that $\overline{R}$ accepts iff R rejects, then $\overline{R}$ computes $\mathrm{Eq}_n = \overline{\mathrm{Ineq}_n}$. This is because, for every input $(\alpha, \beta) \in \{0,1\}^n \times \{0,1\}^n$,
(1) if $\alpha = \beta$ (i.e. $\mathrm{Eq}_n(\alpha, \beta) = 1$), then

$$Prob(\overline{R}(\alpha, \beta) = \text{"accept"}) = 1,$$

(2) if $\alpha \neq \beta$ (i.e. $\mathrm{Eq}_n(\alpha, \beta) = 0$), then

$$Prob(\overline{R}(\alpha, \beta) = \text{"reject"}) \geq 1 - \frac{2 \ln n^2}{n}.$$

Thus, for sufficient large n, $\overline{R}$ is a two-sided-error Monte Carlo protocol for Eq_n. Since R works within $2 \cdot \lceil \log_2 n \rceil$ communication complexity, $\overline{R}$ works with $2 \cdot \lceil \log_2 n \rceil$ communication complexity, too.

□

Observe that Theorem 15 implies that there is an exponential gap between two-sided-error Monte Carlo randomization and one-sided-error Monte Carlo randomization for communication complexity, too.

The following result together with Theorem 15 shows that bounded-error Monte Carlo randomization and nondeterminism are incomparable. For some problems one computing mode can be essentially better than the other one, and for another problem the situation may be vice versa. To see it, consider the following Boolean function

$$\mathrm{Disj}_n(x_1, \ldots, x_n, y_1, \ldots, y_n) : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$$

defined by

$$\mathrm{Disj}_n(\alpha_1, \alpha_2, \ldots, \alpha_n, \beta_1, \beta_2, \ldots, \beta_n) = \Gamma\left(\min\left\{\sum_{i=1}^{n} \alpha_i \beta_i, 1\right\}\right).$$

So, $\mathrm{Disj}_n(\alpha_1, \alpha_2, \ldots, \alpha_n, \beta_1, \beta_2, \ldots, \beta_n) = 1$ iff $\sum_{i=1}^{n} \alpha_i \beta_i = 0$. In other words, $\overline{\mathrm{Disj}}_n(\alpha_1, \alpha_2, \ldots, \alpha_n, \beta_1, \beta_2, \ldots, \beta_n) = 1$ iff there exists a $j \in \{1, 2, \ldots, n\}$ such that $\alpha_j = \beta_j = 1$.

Theorem 16 ([25,26,27]). *For every positive integer n*

(i) $\mathrm{ncc}(\overline{Disj}_n) \leq \lceil \log_2 n \rceil$, *and*
(ii) $2\mathrm{mccc}(\overline{Disj}_n) = \Omega(\sqrt{n})$.

There are several further papers devoted to randomized protocols (see, for instance, [3,4,28,29,30,31,32,33,34]). For all we mention here only that Ablayev [28] showed a nice combinatorial characterization of Boolean functions for which Monte Carlo one-way protocols cannot be much better than the one-way deterministic protocols.

5 A Bound on the Number of Random Bits

The number of random bits used is an important characteristic of every randomized algorithm. To consider the number of random bits as a complexity measure of randomized computation, especially in some tradeoffs with other computational complexity measures, is one of the central research tasks on randomization. There are at least two important general reasons for this.

(i) The random bits are not for free and the cost of producing a sequence of random bits grows essentially with the length of the sequence.
(ii) If the number of random bits used in a randomized computation is not too large, then there is a possibility of an efficient derandomization (i.e., one can create an efficient deterministic algorithm doing the same job as the original randomized one).

Besides this two general reasons above we have one more special reason to investigate the number of random bits for communication protocols. As already discussed in Section 2, we have the two fundamental possibilities to randomize protocols, namely either by using one common random source (public randomized protocols) or by using two independent random sources (private randomized protocols). The difference between these two randomization approaches is that to simulate public randomized protocols by private ones, it may happen that the private random bits need to be communicated. Thus, any upper bound on the number of random bits is also an upper bound on the possible difference between the communication complexity of public randomized protocols and private randomized protocols.

In this section we present the result of Newman [18] who showed that $O(\log_2 n)$ bits are enough to exploit the full power of bounded error (Las Vegas, one-sided-error Monte Carlo, two-sided-error Monte Carlo) protocols.

In what follows, for any finite function f, we denote by **priv-lvcc(f)**, **priv-1mccc(f)**, and **priv-2mccc(f)** resp. the private counterparts of the public randomized communication complexities lvcc(f), 1mccc(f), and 2mccc(f) respectively.

Theorem 17 ([18]). *Let D be a public x-randomized protocol for a Boolean function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, $n \in \mathbb{N}$, where $x \in \{$Las Vegas, one-sided-error Monte Carlo, two-sided-error Monte Carlo$\}$. Then, there exists an equivalent public x-randomized protocol D' for f that uses at most $O(\log_2 n)$ random bits and*

$$\mathrm{cc}(D') = O(\mathrm{cc}(D)).$$

Proof. Because the error probability of randomized protocols can be decreased to an arbitrary constant by a few repetitions of the work of randomized protocols, we may assume that there is a protocol $\widetilde{D}$ that computes f with error probability at most $\frac{1}{6}$ and $\mathrm{cc}(\widetilde{D}) = O(\mathrm{cc}(D))$. The idea of the proof is to use some kind of derandomization of $\widetilde{D}$ in order to construct D' that computes f with error probability at most $\frac{1}{3}$ and $O(\log_2 n)$ random bits.

Let Π be the set of all random sequences used by $\widetilde{D}$, and let *Prob* be the probability distribution over Π. So, we can write $\widetilde{D} = (Prob, \{D_r | r \in \Pi\})$, where D_r is the protocol corresponding to r. Let $Z(\alpha, \beta, r)$ be a random variable that gets the value 1 if $\widetilde{D}$ gives the wrong answer ("?" in the case of Las Vegas protocols) on the input $(\alpha, \beta) \in U \times V$, if choosing the random sequence r (i.e. the (deterministic) protocol D_r corresponding to r gives a wrong answer). Otherwise (if $D_r(\alpha, \beta) = f(\alpha, \beta)$), $Z(\alpha, \beta, r) = 0$. Because $\widetilde{D}$ computes f with error at most $\frac{1}{6}$, we have

$$E_{r \in \Pi}[Z(\alpha, \beta, r)] = \sum_{r \in \Pi} Prob(r) \cdot Z(\alpha, \beta, r) \leq \frac{1}{6} \tag{3}$$

for all $(\alpha, \beta) \in U \times V$.

Our aim is to show that there exist $t = 36n$ deterministic protocols $D'_1, D'_2, \ldots, D'_t$ in $\{D_r \,|\, r \in \Pi\}$ such that $D' = (Pr, \{D'_1, D'_2, \ldots, D'_t\})$ with $Pr(D_i) =$

$Pr(D_j) = \frac{1}{t}$, for all $i, j \in \{1, 2, \ldots, t\}$, computes f with error at most $\frac{1}{3}$. To prove the existence of $D'_1, D'_2, \ldots, D'_t$ we use the probabilistic method.

Let $D_{r_1}, D_{r_2}, \ldots, D_{r_t}$ be arbitrary t (deterministic) protocols from $\{D_r \mid r \in \Pi\}$. Consider the following randomized protocol $D_{r_1 r_2 \ldots r_t} = (Pr, \{D_{r_1}, D_{r_2}, \ldots, D_{r_t}\})$ with the uniform probability distribution Pr. We use

$$E_{Pr}[Z(x, y, r_i)] = \sum_{i=1}^{t} Pr(D_{r_i}) \cdot Z(x, y, r_i) = \sum_{i=1}^{t} \frac{1}{t} \cdot Z(x, y, r_i)$$

to denote the expected error of $D_{r_1 r_2 \ldots r_t}$. By the Chernoff inequality and the bound (3), we obtain

$$\underset{r_1, \ldots, r_t}{Prob} \left[\left(\frac{1}{t} \cdot \sum_{i=1}^{t} Z(\alpha, \beta, r_i) - \frac{1}{6} \right) > \frac{1}{6} \right] \leq 2e^{-2\left(\frac{1}{6}\right)^2 t} < 2^{-2n} \tag{4}$$

for every $(\alpha, \beta) \in U \times V$. The inequality (4) implies that, for a random choice of $r_1, \ldots, r_t$, the probability that $E_{Pr}[Z(x, y, r_i)] > \frac{1}{6} + \frac{1}{6} = \frac{1}{3}$ for some input (α, β) is smaller than $2^{2n} \cdot 2^{-2n} = 1$. This implies that there exists a choice of $r_1, \ldots, r_t$ where, for every $(\alpha, \beta) \in U \times V$, the error $E_{Pr}[Z(x, y, r_i)]$ of the protocol $D_{r_1 r_2 \ldots r_t}$ is at most $\frac{1}{3}$.

Since the protocol D' is a uniform probability distribution over t deterministic protocols, the number of random bits of D' is

$$\lceil \log_2 t \rceil = \lceil \log_2(36n) \rceil \leq 5 \cdot \lceil \log_2 n \rceil.$$

The communication complexity of D' is at most the communication complexity of $\widetilde{D}$, and so[8] $\mathrm{cc}(D') = O(\mathrm{cc}(D))$. □

A direct consequence of Theorem 17 is a linear relation between private randomized communication complexities and the corresponding public randomized communication complexities, provided that the communication complexities are at least of order $\log_2 n$. This result is formulated in the following theorem.

Theorem 18 ([18]). *For every $x \in \{lv, 1mc, 2mc\}$, and for every Boolean function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, $n \in \mathbb{N} - \{0\}$,*

$$priv\text{-}xcc(f) = O(x\mathrm{cc}(f) + \log_2 n).$$

References

1. Yao, A.C.: Some complexity questions related to distributive computing. *Proc. 11th ACM STOC*, ACM 1979, pp. 209–213.
2. Abelson, H.: Lower bounds on information transfer in distributed computations. *Proc. 19th IEEE FOCS*, IEEE 1978, pp. 151–158.

[8] Note that for Las Vegas and one-sided-error Monte Carlo one can write $\mathrm{cc}(D') \leq 3 \cdot \mathrm{cc}(D)$.

3. Hromkovič, J.: *Communication Complexity and Parallel Computing*. Springer-Verlag 1997.
4. Kushilevitz, E., Nisan, N.: *Communication Complexity*, Cambridge University Press 1997.
5. Thompson, C.D.: Area-time complexity for VLSI. *Proc. 11th ACM STOC*, ACM 1979, pp. 81–88.
6. Thompson, C.D.: A complexity theory for VLSI. Doctoral dissertation. CMU-CS-80-140, Computer Science Department, Carnagie-Mellon University, Pittsburgh, August 1980, 131 p.
7. Savage, J.E.: Area-time tradeoffs for matrix multiplication and related problems in VLSI models. *J. of Computer and System Sciences* 20 (1981), pp. 230–242.
8. Leiserson, C.E.: Area efficient graph algorithms (for VLSI). *Proc. 21st IEEE FOCS*, IEEE 1980, pp. 270–281.
9. Lovász, L.: Communication Complexity: A Survey. *Techn. Report CS-TR-204-89*, Princeton University, 1989 (also in: *Paths, Flows and VLSI Layout* (Korte, Lovász, Promel, and Schrijver, eds.), Springer-Verlag 1990, pp. 235–266).
10. Hromkovič, J.: Some complexity aspects of VLSI computations. Part 1. A framework for the study of information transfer in VLSI circuits. *Computers and Artificial Intelligence* 7 (1988), pp. 229–252.
11. Ďuriš, P., Galil, Z.: On the power of multiple reads in chip. *Information and Computation* 104 (1993), pp. 277–287.
12. Karchmer, M., Wigderson, A.: Monotone circuits for connectivity require superlogarithmic depth. *Proc. 20th ACM STOC*, ACM 1988, pp. 539–550. (also *SIAM J. Discrete Mathematics* 3 (1990), 718–727).
13. Alon, N., Maas, W.: Meanders, Ramsey theory and lower bounds for branching programs. *Proc. 27th IEEE FOCS*, IEEE 1986, pp. 410–417.
14. Ajtai, M.: A non-linear time lower bound for Boolean branching programs. *Proc. 40th IEEE FOCS*, IEEE 1999, pp. 60–70.
15. Beame, P.W., Saks, M., Thathachar, J.S.: Time-space tradeoffs for branching programs. *Proc. 39th IEEE FOCS*, IEEE 1998, pp. 254–263.
16. Hromkovič, J.: Nonlinear lower bounds on the number of processors of circuits with sublinear separators. *Information and Computation* 95 (1991), pp. 117–128.
17. Turán, G.: Lower bounds for synchronous circuits and planar circuits. *Information Processing Letters* 130 (1989), pp. 37–40.
18. Newman, I.: Private versus common random bits in communication complexity. *Information Processing Letters* 39 (1991), pp. 67–71.
19. Freivalds, R.: Probabilistic machines can use less running time. *Information Processing 1977, IFIP*, North Holland 1977, pp. 839–842.
20. Aho, A.V., Ullman, J.D., Yannakakis, M.: On notions of informations transfer in VLSI circuits. *Proc. 15th ACM STOC*, ACM 1983, pp. 133–139.
21. Mehlhorn, K., Schmidt, E.: Las Vegas is better than determinism in VLSI and distributed computing. *Proc. 14th Annual ACM Symposium on Theory of Computing*, San Francisco, ACM 1982, pp. 330–337.
22. Hromkovič, J., Schnitger, G.: On the power of Las Vegas for one-way communication complexity, OBDDs, and finite automata. *Information and Computation* 169 (2001), pp. 284–296.
23. Ja'Ja, J., Prassanna Kamar, V.K., Simon, J.: Information transfer under different sets of protocols. *SIAM J. Computing* 13 (1984), pp. 840–849.
24. Papadimitriou, Ch., Sipser, M.: Communication complexity. *Proc. 14th ACM STOC*, San Francisco, ACM 1982, pp. 196–200.

25. Babai, L., Frankl, P., Simon, J.: Complexity classes in communication complexity theory. *Proc. 27th IEEE FOCS*, IEEE 1986, pp. 337–347.
26. Kalyanasundaram, B., Schnitger, G.: The probabilistic communication complexity of set intersection. *Proc. 2nd Annual Conference on Structure in Complexity Theory*, 1987, pp. 41–47.
27. Razborov, A.A.: On the distributed complexity of disjointness. *Proc. 17th ICALP,* Springer-Verlag 1990, pp. 249–253. (also *Theoretical Computer Science* 106 (1992), 385–390.)
28. Ablayev, F.M.: Lower bounds for one-way probabilistic communication complexity and their application to space complexity. *Theoretical Computer Science* 157 (1996), pp. 139–159.
29. Chor, B., Goldreich, O.: Unbiased bits from sources of weak randomness and probabilistic communication complexity. *Proc. 26th IEEE FOCS*, IEEE 1985, pp. 429–442.
30. Halstenberg, B., Reischuk, R.: On different modes of communication. *Proc. 20th ACM STOC*, ACM 1988, pp. 162–172.
31. King, F.P., Abhasel, G.: Communication complexity of computing the Hamming distance. *SIAM J. Computing* 15 (1986), pp. 932–946.
32. Meinel, Ch., Waack, S.: Lower bounds for the majority communication complexity of various graph accessibility problems. *Proc. 20th MFCS'95, Lecture Notes in Computer Science* 969, Springer-Verlag. 1995, pp.299–308.
33. Nisan, N., Wigderson, A.: Bounds in communication complexity revised. *32nd ACM STOC*, ACM 1991, pp. 419–429 (also in *SIAM J. Computing* 22 (1993), pp. 211–219).
34. Yao, A.C.: Lower bounds by probabilistic arguments. *Proc. 25th ACM STOC*, ACM 1983, pp. 420–428.

Optimal Mutation Rate Using Bayesian Priors for Estimation of Distribution Algorithms

Thilo Mahnig and Heinz Mühlenbein

RWCP* Theoretical Foundation GMD Laboratory
D-53754 Sankt Augustin, Germany
{mahnig|muehlen}@gmd.de

Abstract. *U*MDA (the univariate marginal distribution algorithm) was derived by analyzing the mathematical principles behind recombination. Mutation, however, was not considered. The same is true for the *FDA* (factorized distribution algorithm), an extension of the *UMDA* which can cover dependencies between variables. In this paper mutation is introduced into these algorithms by a technique called Bayesian prior. We derive theoretically an estimate how to choose the Bayesian prior. The recommended Bayesian prior turns out to be a good choice in a number of experiments. These experiments also indicate that mutation increases in many cases the performance of the algorithms and decreases the dependence on a good choice of the population size.

Keywords: Bayesian prior, univariate marginal distribution algorithm, mutation, estimation of distribution algorithm

1 Introduction

The Estimation of Distribution Algorithms *EDA* have been proposed by Mühlenbein and Paass [7] as an extension of genetic algorithms. Instead of performing recombination of strings, *EDA*s generate new points according to the probability distribution defined by the selected points. The crucial problems are: How to estimate the probability distribution of the selected points and how to generate new points?

In [6] Mühlenbein showed that genetic algorithms can be approximated by an algorithm using univariate marginal distributions only (*UMDA*). In [9] Mühlenbein and Mahnig have extended this algorithm to general distributions. The corresponding algorithm was called the Factorized Distribution Algorithm (*FDA*). For this algorithm convergence was proven to the set of global optima if Boltzmann selection is used and the size of the population is large enough [12]. Convergence means that the dynamical system defined by the algorithm converges to stable attractors.

UMDA and *FDA* do not use any kind of stochastic mutation. Variation is achieved by a population of individuals. For each problem there exists a critical

* Real World Computing Partnership

K. Steinhöfel (Ed.): SAGA 2001, LNCS 2264, pp. 33–48, 2001.

population size which is needed for convergence to the optima. The determination of a sufficient size of the population turned out to be difficult. In this paper mutation is introduced. Mutation will increase the number of generations needed to converge, but the algorithms can use a smaller population size.

The outline of the paper is as follows. In Sect. 2, the univariate marginal distribution algorithm *UMDA* is defined. In Sect. 3, the connection between mutation and using a prior distribution for the estimates of the probability distribution of the *UMDA* is established. By computing an upper limit on the Bayesian prior, we derive a recommendation on how to choose the prior parameter (or equivalently, the mutation rate). Numerical results for the *UMDA* are presented in Sect. 4.

In Sect. 5, the theory is extended to *FDA*. The recommended prior parameters are tested in Sect. 6 and finally a comparison is made between the *FDA* without mutation and the *FDA* with mutation.

2 The Univariate Marginal Distribution Algorithm

Let $\mathbf{x} = (x_1, \ldots, x_n)$ denote a vector, $x_i \in \{0, 1\}$. We use the following conventions. Capital letters X_i denote variables, small letters x_i assignments. For simplicity we only consider binary variables here. Let a function $f : \{0,1\}^n \to \mathbb{R}_{>0}$ be given. We consider the optimization problem $\mathbf{x}_{opt} = \operatorname{argmax} f(\mathbf{x})$.

The key idea is to consider a population as a sample of a probability distribution $p(\mathbf{x}, t)$. Selection in the analysis is then an operator that calculates the selected distribution $p^s(\mathbf{x}, t)$ from the previously generated distribution $p(\mathbf{x}, t)$. In the algorithms it can be implemented by choosing a set of points from the previously generated ones, possibly taking some points several times. This selected set often has cardinality M smaller than N, the population size, but this need not be the case.

Definition 1. *Let $p(\mathbf{x}, t)$ denote the probability of $\mathbf{x}$ in the population at generation t. Then $p_i(x_i, t) = \sum_{\mathbf{x}: X_i = x_i} p(\mathbf{x}, t)$ defines the univariate marginal distributions of variable X_i.*

In order to estimate the probabilities from occurences in a population, we need to make an assumption on the distribution, otherwise the population size needed to make an accurant estimate would be exponentially high. The simplest probability distribution is $p(\mathbf{x}, t) = \prod_{i=1}^{n} p_i(x_i, t)$. It is called Robbins' proportions or linkage equilibrium in population genetics and the mean field approach in physics.

The *Univariate Marginal Distribution Algorithm* (*UMDA*) [6] approximates genetic recombination by computing

$$p(\mathbf{x}, t+1) = \prod_{i=1}^{n} p_i^s(x_i, t) \tag{1}$$

directly instead of selecting parents for every variable. Thus the population is kept in linkage equilibrium.

UMDA

STEP 0: Set $t \Leftarrow 1$. Generate $N \gg 0$ individuals $\mathfrak{X}^1, \ldots, \mathfrak{X}^N$ randomly.

STEP 1: Select $M \leq N$ individuals $\hat{\mathfrak{X}}^j$ from $\mathfrak{X}^j$ according to a selection method. Compute the sample marginal frequencies $p_i^s(x_i, t)$ of the selected set.

STEP 2: Generate N new points according to the distribution $p(\mathbf{x}, t+1) = \prod_{i=1}^{n} p_i^s(x_i, t)$. Set $t \Leftarrow t+1$.

STEP 3: If termination criteria are not met, go to STEP 1.

There are many possibilities to select points according to fitness. In this paper, we consider proportionate selection and truncation selection. Let $\bar{f}(t) = \sum_x p(x,t) f(x)$ be the average fitness.

Definition 2. ***Proportionate selection*** *changes the probabilities according to*

$$p^s(\mathbf{x}, t) = p(\mathbf{x}, t) \frac{f(\mathbf{x})}{\bar{f}(t)} \tag{2}$$

Note that in UMDA with proportionate selection, we can also directly adjust the marginal frequencies according to the fitness values.

Truncation selection *with parameter* $0 < \tau < 1$ *selects the* $\tau \cdot N$ *individuals with highest fitness.*

UMDA formally depends on $2n$ parameters, the marginal distributions $p_i(x_i, t)$. We now interprete the average $\bar{f}(t) = \sum_x p(x,t) f(x)$ as a function which depends on $p_i(x_i, t)$ because of the linkage equilibrium assumption. As $p_i(0,t) = 1 - p_i(1,t)$, we eliminate half of the parameters and by setting $p_i(t) := p_i(1,t)$, we write

$$W(t) = W\big(p_1(t), \ldots, p_n(t)\big) := \bar{f}(t) \tag{3}$$

We can now formulate difference equations, describing the dynamic behaviour of $p_i(t)$.

Theorem 3. *For an infinite population and proportionate selection UMDA changes the gene frequencies as follows:*

$$p_i(t+1) = p_i^s(t) = p_i(t) + p_i(t) \frac{\big(1 - p_i(t)\big)}{W(t)} \frac{\partial W}{\partial p_i} \tag{4}$$

The stable attractors of Wright's equation are at the corners, i.e. $p_i \in \{0, 1\}$ *for* $i = 1, \ldots, n$. *In the interior there are only saddle points or local minima where* $\operatorname{grad} W(p)) = 0$. *The attractors are local maxima of* $f(x)$ *according to one bit changes. Wright's equation solves the continuous optimization problem* $\operatorname{argmax}\{W(\mathbf{p})\}$ *by gradient ascent.*

The theorem has been proven in [6]. Equation (4) was already conjectured by Wright [14]. In populations genetics it is now called Wright's equation.

3 Mutation and the Bayesian Prior for *UMDA*

We will introduce mutation into *UMDA* by a concept called Bayesian prior. There are two possibilities to estimate the probability of "head" of a biased coin. The maximum-likelihood estimate counts the number of occurrences of each case. With m times "head" in N throws, the probability is estimated as $p = m/N$.

The Bayesian approach assumes that the coin is determined by an unknown parameter θ. Starting from an *a priori* distribution of this parameter, the probability distribution of this parameter is modified by observing coin flips (data) using Bayes' rule. The usual distribution chosen for the binomial experiments is the family of Dirichlet distributions. In practice, this means that the estimated probability becomes $p = (m + r)/(N + 2r)$ and the so called *hyperparameter* r has to be chosen in advance [3].

Mutation in genetic algorithms works in the following way: When generating new individuals, with a probability of μ the generated bit is inverted.

The following theorem relates mutation with the Bayesian prior.

Theorem 4. *For binary variables, the expectation value for the probability using a Bayesian prior with parameter r is the same as mutation with mutation rate $\mu = r/(N + 2r)$ and using the maximum likelihood estimate.*

This can be proven simply by calculating the probability of generating a particular bit for both cases.

3.1 A Bound for the Bayesian Prior

A Bayesian prior moves the attractors from the corners into the interior. For $r \to \infty$ there is a unique attractor at $p = 0.5$. Using this prior *UMDA* is a purely random algorithm. Thus the question arises how to choose the prior parameter r or equivalently, the mutation rate μ, so that an optimum will be generated with a certain probability. The following observation allows us to find an upper limit.

Let the optimum be $(1, \ldots, 1)$. When the population consists only of this optimum, new individuals would be generated with probability

$$P_s := p(X_1 = 1, \ldots, X_n = 1) = \left(\frac{N + r}{N + 2r}\right)^n . \tag{5}$$

In order that the optimum remains an attractor, the probability has to be greater than a constant, say 0.3. For $r = N/n$ we have

$$\left(\frac{N + r}{N + 2r}\right)^n = \left(\frac{1 + 1/n}{1 + 2/n}\right)^n = \left(1 - \frac{1}{n + 2}\right)^n \geq 0.3 \tag{6}$$

as the expression on the right side is monotonically converging towards e^{-1}. So for this choice of r, the population can get sufficiently close to the optimum. If r increases, the probability of generating the optimum decreases exponentially.

This bound has been derived with an unrealistic model. In reality the attractors are defined by a dynamic equilibrium between selection (driving the population to the corner) and mutation (driving the population to the interior). This problem is investigated next.

3.2 Wright's Equation with Bayesian Prior

With mutation, the dynamical equations for proportionate selection have to be extended. When using the Bayesian prior, we get for binary variables

$$p_i(t+1) = \frac{p_i^s(t)N + r}{N + 2r} = p_i^s(t) + \frac{r}{N+2r} - \frac{2r}{N+2r} \cdot p_i^s(t), \tag{7}$$

where $p_i^s(t)$ is given by (4). Combining these equations, we get

Theorem 5. *Wright's equation with Bayesian prior instead of maximum likelihood estimates is given by*

$$\begin{aligned} p_i(t+1) =& p_i(t) + p_i(t)\frac{\big(1-p_i(t)\big)}{W(t)}\frac{\partial W}{\partial p_i} + \frac{r}{N+2r} \\ & - \frac{2r}{N+2r}\left(p_i(t) + p_i(t)\frac{\big(1-p_i(t)\big)}{W(t)}\frac{\partial W}{\partial p_i}\right) \end{aligned} \tag{8}$$

The attractors are given by $p_i^* := p_i(t+1) = p_i(t)$. *They are moving towards* $p_i = 1/2$ *with increasing* r.

Consider the function $f(\mathbf{x}) = \mathit{OneMax}(\mathbf{x}) = \sum_{i=1}^{n} x_i$. We will apply (8) to the maximization problem $\max_{\mathbf{x}} \mathit{OneMax}(\mathbf{x})$. Because the equation is identical for all p_i, we have $p_i = p_j =: p$ for all i, j. Then we obtain $W = n \cdot p$ and $\partial W/\partial p = 1$ (see [11] for details how to calculate the average fitness and derivatives).

Corollary 6. *The attractors of UMDA with proportionate selection for the function OneMax with bit length* n *and population size* N *are given by*

$$p^* = \frac{1 + rn/N}{1 + 2rn/N} \tag{9}$$

Choosing $r = \frac{N}{n}$ yields a stable attractor with $p^* = 2/3$ and an expectation value $\bar{f} = 2/3n$, while the optimum has value n. A simulation run of *UMDA* with $n=10$, $N=1000$ and $r = 100$ had as average marginal frequency in the last generation $\bar{p}_i = 0.669$. So a large-population simulation behaves similarly to the infinite population models from Corollary 6.

Thus this Bayesian prior is too large. In fact, a much smaller prior has to be chosen. A prior of $r = N/n^2$ (corresponding to a mutation rate of $1/n^2$) yields a value of $p^* \approx 1 - 1/n$ (for large n). In this case, we have $\lim_{n\to\infty} P_s(\mathbf{x}_{\max}) = e^{-1} > 0.3$ and $\bar{f} = n(n+1)/(n+2) \approx n$.

3.3 Truncation Selection and the Bayesian Prior

Wright's equation has been derived under the assumption of proportionate selection. But proportionate selection is far too weak for optimization. Truncation selection is much more efficient. Unfortunately a general dynamic equation for truncation selection has not yet been found. Again, the difference equations are equal for all p_i, so $p := p_i$. Mühlenbein [6] has derived the dynamical equations for the function *OneMax* as

$$p(t+1) = p^s(t) \approx p(t) + \frac{I_\tau}{n}\sqrt{n \cdot p(t)\big(1-p(t)\big)} \tag{10}$$

where I_τ, the selection intensity, depends on the truncation threshold τ [6]. It is defined as $I_\tau = S(t)/\sigma(t)$, the quotient of the selection differential and the standard deviation of the population, where $S := \bar{f}_s(t) - \bar{f}(t)$ and $\bar{f}_s(t)$ is the average of the selected points. Using a normality assumption, I_τ depends only on τ [2].

As the Bayesian estimator is used only for the selected points, we have to use $M = \tau N$ instead of N in the formulas. Thus when using the Bayesian estimate after selection, we get

$$p(t+1) = \frac{p^s(t)M + r}{M + 2r}, \tag{11}$$

Using (10) we get:

Theorem 7. *For UMDA with truncation selection we obtain for the function OneMax the dynamical equation*

$$\begin{aligned} p(t+1) = p(t) + \frac{I_\tau}{n}\sqrt{n \cdot p(t)\big(1-p(t)\big)} + \frac{r}{M+2r} \\ - \frac{2r}{M+2r}\left(p(t) + \frac{I_\tau}{n}\sqrt{n \cdot p(t)\big(1-p(t)\big)}\right) \end{aligned} \tag{12}$$

Setting $p(t+1) = p(t)$, the stable attractor is given by

$$p^* = \frac{1}{2} + \frac{I_\tau M}{2\sqrt{I_\tau^2 M^2 + 4nr^2}} = \frac{1}{2} + \frac{1}{2\sqrt{1 + 4\rho^2/n}} \tag{13}$$

with $\rho := (r \cdot n)/(I_\tau M)$.

The attractor depends only on ρ and n. Comparison with actual simulation runs gave excellent results. We just show two runs in Table 1.

An attractor in the interior is good because it reduces the probability of fixation. But we have the constraint that the optimum should be generated with a high probability. In Fig. 1 we show the probability to generate the optimum

Table 1. Comparing the attractor from (13) with a simulation run for *OneMax* with population size 2000.

	n	Theory	Run	n	Theory	Run
p^*	32	0.9714	0.967	64	0.9851	0.983

$P_s := (p^*)^n$ versus ρ. A constant value of ρ keeps the probability high enough for arbitrary n. Thus we obtain the following rule of thumb:

For truncation selection use a value of $\rho = 1$*, corresponding to* $r = I_\tau M/n$ *or* $\mu = 1/\big[n/(I_\tau\tau) + 2\big]$*. This leads to the UMDA-M algorithm, or UMDA with mutation.*

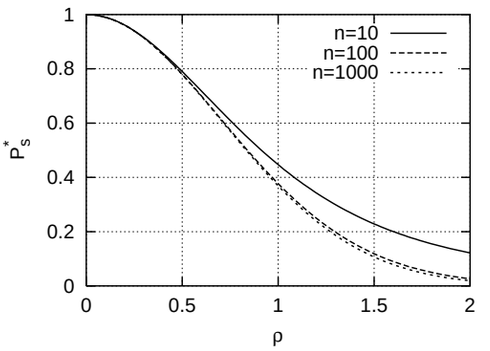

Fig. 1. Value of P_s, the probability to generate the optimum, when varying ρ for different bit lengths using formula (13).

Remark: The algorithm with Bayesian prior can run with a very small population size. Setting $N = \lambda$ and $M = 1$, we obtain the $(1, \lambda)$ evolution strategy. By using the approximation $I_\tau \approx \ln\lambda/\ln 2$, we get $r \approx \ln\lambda/(n \ln 2)$.

4 Numerical Results for *UMDA*

The preceeding analysis was done using many simplifications. In this section we compare actual simulation results with the theoretical results. In addition we investigate a difficult multi-modal function by simulations.

4.1 Simulation Results for *OneMax*

Our goal is to show that our rule of thumb is correct for *OneMax*. ρ depends on M, I_τ, r and n.

In Fig. 2, we examine the behaviour for a fixed number of bits (100) and for population sizes between 10 and 70 with a truncation threshold of $\tau = 0.3$. Plotted are the number of function evaluation for successful runs until 10% of the population consisted of the optimum against the prior parameter r scaled by $n/(I_\tau M)$, where $M = \tau N$ is the number of selected points. Only points where the success rate was higher than 90% are shown.

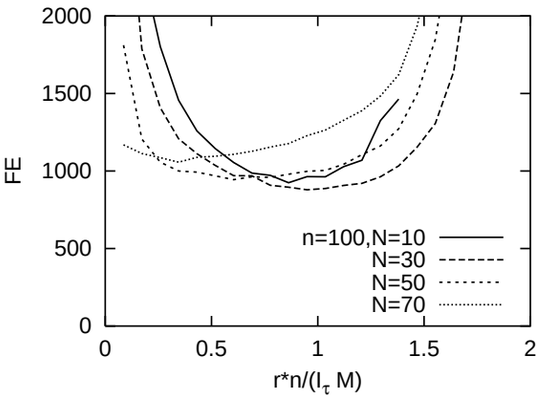

Fig. 2. Function evaluations with varying prior for different population sizes with *OneMax* and 100 bits. Shown are the number of function evaluation for successful runs (when the success rate was above 90%), against the scaled prior parameter.

When the mutation rate is too high, the number of function evaluations increases rapidly, as the probabilities are shifted towards $1/2$ too much.

When the mutation rate decreases, the number of evaluations increases also, because it becomes too improbable to flip the remaining bits that are wrong. On the far left the success rate got below 90%.

It is remarkable that for all four population sizes, with $r = I_\tau M/n$, roughly the same number of function evaluations are needed. As this number is the product of the number of generations and the population size, this implies that the number of generations needed decreases at roughly the same rate as the number of individuals increases.

Similar results have been obtained for varying τ and varying n. The simulation results confirm the theoretical result.

To summarize: *For OneMax with truncation selection, taking $\rho \approx 1$ is a good choice for a range of population sizes, selection intensities and bit lengths. Despite big changes in the number of generations, the number of function evaluations remains remarkably constant for a given number of bits when the population size and the truncation threshold are varied within a reasonable range.*

4.2 Simulation Results for *Saw*

The definition of *Saw* can be seen from figure 3. A local search has to cross many valleys of depth 2 in order to get to the optimum. Thus *Saw* is very difficult to optimize for any kind of local optimization [8]. In contrast, *Saw* is surprisingly easy to optimize for population based search methods like genetic algorithms and *UMDA*. The reason is that the fitness landscape defined by the average fitness W is smooth and has only a single valley shortly before the optimum $W = 8$ (see [11] for a discussion).

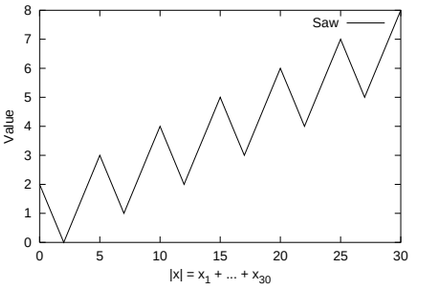

Fig. 3. Definition of the *Saw* function

In Fig. 4, we can see results of using different prior parameters for this function. We varied the size of the problem and set $M = \sqrt{n}$. This approximation was proposed in [10]. The truncation threshold was set to $\tau = 0.25$.

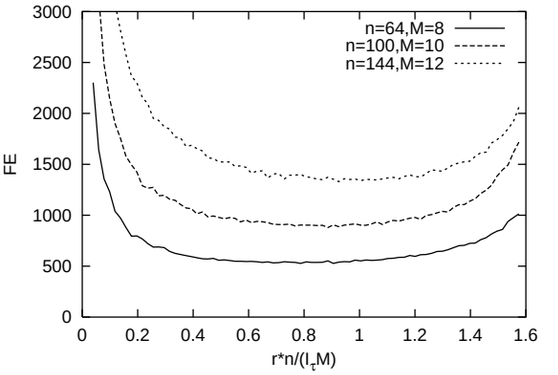

Fig. 4. Function evaluations for several bit lengths and population sizes for the *Saw* function with success rate $> 90\%$; $\tau = 0.25$.

We see that $\rho \approx 1$ is a good choice for this function also.

UMDA has been tested with many other functions also. In all cases $\rho \approx 1$ turns out to be a good recommendation.

5 Bayesian Prior and *FDA*

UMDA is an efficient algorithm for many multi-modal functions. But when interactions between variables have to be considered, it might fail. Thus we have extended it to the *FDA*, the factorized distribution algorithm [11,12].

The idea of the *FDA* is to chose a different factorization for the underlying probability distribution. Instead of $p(x) = \prod_i p_i^s(x_i)$, we have the more general expression

$$p(x) = \prod_{i=1}^{n} p^s(x_i|\pi_i) = \prod_{i=1}^{n} p^s(x_i|x_{i_1}, \ldots, x_{i_{k_i-1}}) \tag{14}$$

where each $\pi_i = (x_{i_1}, \ldots, x_{i_{k_i-1}})$ is are subset of $(x_1, \ldots, x_n)$. They are chosen in such a way that interactions in the function definition are represented and that a valid probability distribution is generated. This can be done using an additive decomposition of the function [10]. An additive decomposition is a representation $f(\mathbf{x}) = \sum_i f_i(x_i, \pi_i)$, where each f_i depends only on the subset (x_i, π_i) of the variables.

In the context of Bayesian networks, these factorizations can be represented by a directed acyclic graph where the π_i are called the *parents* of each node x_i. For an introduction in modelling probability distributions using Bayesian networks see [13,4].

If the function is additively decomposed and separable (the S_i have empty intersection), then *FDA* is equivalent to *UMDA* with multiple alleles. For proportionate selection an extension of Wright's equation can be obtained [10]. This equation can be extended to include mutation as before. The dynamical equations for other selection methods are very difficult, therefore we use heuristics to derive some bounds for the mutation rate.

5.1 Calculating a Bound for the Prior Parameter for *FDA*

For *UMDA*, we have seen the equivalence between using mutation and using a Bayesian prior. There seems to be no obvious reason to prefer one to the other. This is different for *FDA*. The concept of linked variables leads to the need of a mutation operator that is in accordance with the structure of the distribution.

The theory of Bayesian estimates can be extended to the cases of conditional probabilities [1]. The chain rule of conditional probabilities says that

$$p(x_1, \ldots, x_k) = p(x_1) \cdot p(x_2|x_1) \cdot p(x_3|x_1, x_2) \cdots p(x_k|x_1, \ldots, x_{k-1}) \tag{15}$$

Using the Bayesian priors, we get the following equation for binary variables x_i:

$$\begin{aligned}\frac{N(x_1,\ldots,x_k)+r'}{N+2^k r'} =& \frac{N(x_1)+r_1}{N+2r_1}\cdot\frac{N(x_1,x_2)+r_2}{N(x_1)+2r_2}\cdot\frac{N(x_1,x_2,x_3)+r_3}{N(x_1,x_2)+2r_3}\cdots \\ &\cdot\frac{N(x_1,\ldots,x_k)+r_k}{N(x_1,\ldots,x_{k-1})+2r_k}\end{aligned} \tag{16}$$

where $N(\cdot) = N \cdot p(\cdot)$ is the number of occurrences in the population and the denominators where chosen in such a way that $\sum_{x_1,\ldots,x_k} p(x_1,\ldots,x_k) = 1$ and $\sum_{x_i} p(x_i|x_1,\ldots,x_{i-1}) = 1$.

In order for (16) to hold, the fractions on the right hand side have to cancel each other out. We get the following identities for the parameters:

$$2r_i = r_{i-1} \implies r_i = 2^{-(i-1)} r_1 \quad \text{and} \quad r' = r_k = 2^{-(k-1)} r_1 \tag{17}$$

Thus we get

Lemma 1. *When r is the prior for a single binary variable, the prior r' for a factor $p(x_1,\ldots,x_k)$ and the prior r^* for a factor $p(x_k|x_1,\cdots,x_{k-1})$ should be*

$$r' = r^* = 2^{-(k-1)} \cdot r \tag{18}$$

We now derive an estimate for r' as before. Let the probability distribution be the product of marginal distributions of k_i variables each.

Rule of thumb: *Using $r'_i = 2^{-(k_i-1)} r$ with $r = I_\tau M/n$ is a reasonable choice for the Bayesian prior for truncation selection.*

This result can be obtained from the following argument: When all individuals are equal to the optimum, using (14), the probability of generating the optimum is approximately

$$P_s \approx \prod_{i=1}^{l} \left(\frac{M + r'_i}{M + 2^{k_i} \cdot r'_i} \right) \tag{19}$$

where $r'_i = 2^{-(k_i-1)} r$. If we set $r = I_\tau M/n$ we get

$$P_s = \prod_{i=1}^{l} \left(\frac{n + I_\tau 2^{-(k_i-1)}}{n + 2I_\tau} \right) = \prod_{i=1}^{l} \left(1 - \frac{2(1 - I_\tau 2^{-k_i})}{n + 2I_\tau} \right) \geq 0.1 \tag{20}$$

for $\tau > 0.4$.

6 Numerical Results

As a first simple example for the *FDA* we use the separable deceptive function of order m. It is defined as follows:

$$Dec_m(x) := \begin{cases} m - 1 - |x| & 0 \leq |x| < m - 1 \\ m & |x| = m \end{cases} \tag{21}$$

where $|x| = \sum_i x_i$. The global maximum is isolated at $x = (1, \ldots, 1)$. A deceptive function of order n is a needle in a haystack problem. This is far too difficult to optimize for any optimization method. We simplify the optimization problem by adding l distinct Dec_4-functions to give a fitness function of size $n = l \cdot 4$. This function is also deceptive for population based search algorithms. The local optimum $x = (0, \ldots, 0)$ is surrounded by good fitness values, whereas the global optimum is isolated.

$$Dec_{(l,4)}(x) = \sum_{i=1}^{l} Dec_4(x_{4i-3}, x_{4i-2}, \ldots, x_{4i}) \tag{22}$$

$Dec_{(l,4)}$ is separable. The corresponding *FDA* factorization is given by

$$p(x,t) = \prod_{i=1}^{l} p_i(x_{4i-3}, x_{4i-2}, \ldots, x_{4i}, t)$$

Thus *FDA* is identical to *UMDA* with 16 states (alleles) at each position. Thus these two algorithms behave similarly [9]. In Table 2 the attractors of *UMDA* and *OneMax* are compared to the attractor of *FDA* and $Dec_{(l,4)}$. It compares the average marginal frequencies $p_i(1)$ for *OneMax* (taken from Table 1) with the same frequencies for the Dec_4-function.

Table 2. Attractor for *OneMax* and $Dec_{(l,4)}$ for $r = I_\tau M = 1/n$ together with the limiting value when the population consists only of the optimum.

	n	*OneMax*	$Dec_{(8,4)}$	n	*OneMax*	$Dec_{(16,4)}$
p^*	32	0.967	0.965	64	0.983	0.982

As the blocks consist of four binary parameters, the recommendation for the prior parameter is $r' = 2^{-(k-1)}r$. Setting $r = I_\tau M/n$ we obtain $r' = I_\tau M/8n$. All simulations confirm that $r' = I_\tau M/8n$ is not the optimal point, but it is a reasonably good choice. Overall our results clearly show that separable functions pose no difficulties.

6.1 Using a Prior versus a Larger Population Size

A Bayesian prior reduces the fixation problem and therefor the probability of finding the optimum. This comes at a price, however. The search using mutation needs more generations than without mutation. So, the number of function evaluations increases. In order to make a fair comparison, we have to compare the case with prior to a run without prior, but with a bigger population size.

In Table 3, these two cases were compared for several test functions. The population sizes were chosen to give comparable success rates.

The *Kaufman* function is derived from Kaufman's $n-k$-model [5]. These functions consist of n binary variables and n sub-functions f_{s_i} depending on

x_i and k further variables. The function values of the sub functions are chosen uniformly random from the interval $[0, 1]$:

$$Kaufman_{n,k}(x) = \frac{1}{n}\sum_{i=1}^{n} f_{s_i}(x) \quad |s_i| = k+1,\ i \in s_i \tag{23}$$

In our case, the functions f_{s_i} were chosen to be dependent of x_i and the adjacent variables x_{i-1} and x_{i+1}. For this special case, the global optimum can be calculated in linear time by dynamic programming. The chain structure leads to the probability distribution

$$p(x) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2)p(x_4|x_2, x_3)\cdots \tag{24}$$

Experiments have shown that the recommended prior is again a good choice, see figure 5.

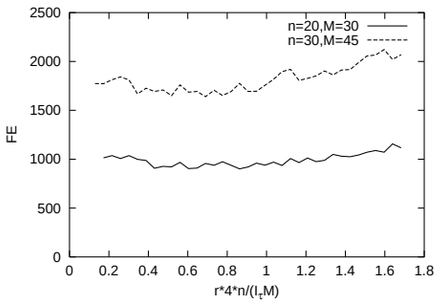

Fig. 5. Kaufman's $n-k$-function with $k = 2$ and adjacent neighbourhood. Only points with success rate above 90% are plotted.

A more difficult example is the function *IsoPeak*. It is defined as follows:

$$IsoPeak(x) = \sum_{i=1}^{l} Iso_1(x_{2i-1}, x_{2i}) + Iso_2(x_1, x_2) \tag{25}$$

with

x	00	01	10	11
Iso_1	l	0	0	$l-1$
Iso_2	0	0	0	l

(26)

and $n = l + 1$. This function is very hard to optimize. The global optimum is $(1, 1, \ldots, 1)$ with value $l \cdot (l-1)+1$. This optimum is triggered by Iso_2. It is very isolated, the second best value is $(0, 0, \ldots, 0)$ with value $l(l-1)$. The associated factorization is

$$p(x) = p(x_1, x_2)p(x_3|x_2)p(x_4|x_3)\cdots p(x_n|x_{n-1})$$

Table 3. Performance comparison with and without prior for suitably chosen population sizes and truncation selection with $\tau = 0.3$.

		With Prior				Without Prior			
Name	n	N	Succ	FE	σ	N	Succ	FE	σ
OneMax	100	30	500/500	764	216	100	491/500	1146	71
Saw	40	40	500/500	474	137	60	475/500	484	63
Dec-4	32	60	500/500	663	404	150	475/500	814	107
Kaufman	30	150	486/500	1279	883	250	484/500	1488	223
IsoPeak	30	900	483/500	4764	707	400	484/500	1996	246

It can be seen in table 3 that for all functions except the last one, *IsoPeak*, performance with a prior was better than performance without a prior.

An even more convincing reason for using the prior is given by Table 4. For the runs from Table 3 without prior, the population size had to be carefully chosen in order to have a good result. When using the prior, this choice is much more easy. In Table 4, several runs were made for the *OneMax* function with 100 bits and different population sizes. When using the prior, both success rate and number of function evaluations were not much affected by choosing a population size between 20 and 50.

Table 4. The effect of having the "wrong" population for *OneMax* with 100 bits.

With prior				Without prior			
N	Succ	FE	σ	N	Succ	FE	σ
20	499/500	801	214	70	405/500	838	54
30	500/500	764	216	100	491/500	1146	71
50	500/500	786	135	130	500/500	1452	77

Without the prior, reducing the population size from 100 to 70 reduced the number of successful runs from 491 to 405 out of 500. The number of function evaluations decreased in this process, but because the success rate was too low, this is not a good choice.

When the population sized was increased from 100 to 130, the success rate reached the limit of 100%. Because the number of generations for the *UMDA* without a prior depends only on the bit length, the number of function evaluations (product of population size and number of generations) has to rise. Note that the ratio of function evaluations ($1452/1146 \approx 1.27$) is almost the same as the ratio of population sizes, 1.3.

To summarize: Without prior, choosing a too small population size leads to a small success rate; while choosing a too big population size is a waste of function evaluations. Both effects are overcome by using a prior.

7 Conclusions

By using a simple argument, we were able to derive an upper bound on a good choice of the mutation rate both for the simple product distribution of *UMDA* and for the conditional distributions of *FDA*. Experimentally, it has turned out that this upper bound is actually a good choice for a range of fitness functions, bit lengths, populations sizes and truncation thresholds. By using mutation, the number of function evaluations can be reduced for most of the test functions considered. In addition, by using this method, the dependence on the right choice of the population size could be drastically reduced.

UMDA with mutation is a robust search method. We recommend to use a population size of 50 and $\tau = 0.3$. It can also be used with very severe selection (e.g. $\tau = 0.02$) making *UMDA*-M mainly driven by mutation and similar to evolution strategies.

Acknowledgments

We thank the reviewers for many helpful comments and suggestions.

References

1. G.F. Cooper and E.A. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992.
2. D. S. Falconer. *Introduction to Quantitative Genetics*. Longman, London, 1981.
3. D. Heckerman. A tutorial on learning with Bayesian networks. In Jordan [4], pages 301–354.
4. M.I. Jordan, editor. *Learning in Graphical Models*. MIT Press, Cambrigde, 1999.
5. St.A. Kauffman and S. Levin. Towards a general theory of adaptive walks on rugged landscapes. *Journal of Theoretical Biology*, 128:11–45, 1987.
6. H. Mühlenbein. The equation for response to selection and its use for prediction. *Evolutionary Computation*, 5:303–346, 1998.
7. H. Mühlenbein and G. Paaß. From recombination of genes to the estimation of distributions i. binary parameters. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Lecture Notes in Computer Science 1141: Parallel Problem Solving from Nature - PPSN IV*, pages 178–187, Berlin, 1996. Springer-Verlag.
8. H. Mühlenbein and J. Zimmermann. Size of neighborhood more important than temperature for stochastic local search. In *Proceedings of the 2000 Congress on Evolutionary Computation*, pages 1017–1024, New Jersey, 2000. IEEE Press.
9. Heinz Mühlenbein and Thilo Mahnig. Convergence theory and applications of the factorized distribution algorithm. *Journal of Computing and Information Technology*, 7:19–32, 1999.
10. Heinz Mühlenbein and Thilo Mahnig. FDA – a scalable evolutionary algorithm for the optimization of additively decomposed functions. *Evolutionary Computation*, 7(4):353–376, 1999.
11. Heinz Mühlenbein and Thilo Mahnig. Evolutionary algorithms: From recombination to search distributions. In L. Kallel, B. Naudts, and A. Rogers, editors, *Theoretical Aspects of Evolutionary Computing*, Natural Computing, pages 137–176, Berlin, 2000. Springer Verlag.

12. Heinz Mühlenbein, Thilo Mahnig, and A. Rodriguez Ochoa. Schemata, distributions and graphical models in evolutionary optimization. *Journal of Heuristics*, 5(2):215–247, 1999.
13. J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufman, San Mateo, 1988.
14. S. Wright. Evolution in Mendelian populations. *Genetics*, 16:97–159, 1931.

An Experimental Assessment of a Stochastic, Anytime, Decentralized, Soft Colourer for Sparse Graphs*

Stephen Fitzpatrick and Lambert Meertens

Kestrel Institute, 3260 Hillview Avenue, Palo Alto, California, U.S.A.
fitzpatrick@kestrel.edu, meertens@kestrel.edu

Abstract. This paper reports on a simple, decentralized, anytime, stochastic, soft graph-colouring algorithm. The algorithm is designed to quickly reduce the number of colour conflicts in large, sparse graphs in a scalable, robust, low-cost manner. The algorithm is experimentally evaluated in a framework motivated by its application to resource coordination in large, distributed networks.

Keywords: Constraint optimization, conflict minimization, decentralized algorithms, anytime algorithms, graph colouring.

1 Introduction

Soft graph colouring is an extension of traditional graph colouring in which the hard constraint that no two adjacent nodes have the same colour is relaxed into a soft constraint: the number of adjacent nodes with the same colour is to be minimized (in other words, the number of *colour conflicts* is to be minimized).

The soft version is useful when large, distributed graphs must be coloured in real-time: the size of the graphs, time constraints, combinatorial complexity and communication latency practically ensure that hard colouring will be impossible to achieve. Instead, an application is designed to work with graphs that are *mostly* properly coloured, but which may contain some colour conflicts. Such applications arise in resource coordination in large, distributed networks; for example, the nodes may represent resources that are to be scheduled, the colours may represent time slots in a cyclic schedule, and the edges may represent mutual exclusion constraints between the resources (see the appendix for an example).

It is assumed that colouring is performed simultaneously with some client process that requires the graph to be coloured, in an anytime, pipeline fashion: the colourer continually and incrementally improves the colouring and continually feeds the current colouring to the client system. Since it assumed that

* This work is sponsored in part by DARPA through the 'Autonomous Negotiating Teams' program under contract #F30602-00-C-0014, monitored by the Air Force Research Laboratory. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

K. Steinhöfel (Ed.): SAGA 2001, LNCS 2264, pp. 49–63, 2001.

higher quality colourings result in better performance by the client system, the emphasis is on quickly reducing the colour conflicts to an acceptable level rather than on deliberating at length to construct a proper colouring.

Moreover, the colouring algorithm must be decentralized to allow it to respond quickly to changing circumstances: e.g., the graph topology changing as a result of hardware fluctuations. Indeed, the algorithm must be tolerant of hardware/communication failures.

Finally, the colouring algorithm must be efficient in its own use of network resources: the colourer is bound to require, for example, communication and computational resources and the colourer's use of these resources must not reach a level at which it interferes with the real tasks that the client process is supposed to be achieving. (Of course, the client process cannot achieve those tasks until the colourer has made some headway — a balance must be struck by the colourer to quickly coordinate the client process while not obstructing it.)

In summary, the problem to be addressed is that of reducing the number of colour conflicts in large, distributed graphs using a quick, real-time, low-cost, robust, decentralized algorithm.

The remainder of this paper is as follows: soft graph colouring is defined; a simple, iterative repair algorithm for soft graph colouring is introduced and some flaws identified by experimental evaluation; an improvement is proposed, based on an algorithm previously published by another author; potential problems with this too are identified and a final variant of the algorithm defined. Then the final variant is assessed against the criteria given in the preceding paragraph.

Soft graph colouring may be viewed as an interesting and challenging problem in and of itself. Nevertheless, an appendix is included that outlines the use of soft colouring in a resource management problem, which may further illuminate some of the evaluation criteria.

2 Soft Graph Colouring

Let G be an undirected, irreflexive graph with node set N and non-empty edge set E. An edge between nodes $u \in N$ and $v \in N$ is denoted by $\{u, v\}$. Two nodes are said to be neighbours (or to be adjacent) iff there is an edge between them.

In this paper, only k-colourings are considered: a k-colouring ($k \geq 1$) C of a graph is an assignment of an integer from Z_k to each of the graph's nodes; the colour of node u is denoted by C_u.

An edge $\{u, v\} \in E$ is said to be a conflict iff $C_u = C_v$. The *unnormalized degree of conflict* γ of a colouring C is the fraction of edges that are conflicts: $\gamma \equiv |\{\{u, v\} \in E | C_u = C_v\}|/|E|$. The unnormalized degree of conflict has range $[0, 1]$, where 0 corresponds to a proper colouring and 1 corresponds to a colouring in which every node has the same colour.

The chromatic number χ of a graph is the smallest k for which a proper k-colouring is possible. In traditional graph colouring, it is usual to consider k-colourings only for $k \geq \chi$. However, in soft graph colouring, a k-colouring for any $k \geq 1$ makes sense, although if $k < \chi$ then it will not be possible to reduce

- Each node randomly chooses a colour from Z_k, collectively constructing the initial colouring C^1.
- The following synchronized loop is then performed indefinitely, for step $s = 1, 2, ...$, by each node i:
  1. Determine an activation probability $\alpha_i \in [0, 1]$.
  2. Generate a random number $r_i \in [0, 1)$.
  3. Choose a colour C_i^{s+1} for the next step:
     a. If $r_i < \alpha_i$, choose C_i^{s+1} such that it minimizes the conflicts with i's neighbours.
     b. If $r_i \geq \alpha_i$, choose $C_i^{s+1} = C_i^s$ (i.e., no change).
  4. If $C_i^{s+1} \neq C_i^s$, inform neighbours.

Fig. 1. Synchronous, stochastic min-conflicts colouring algorithm

γ to 0. In general, the analytical determination of the smallest possible value of γ for a given $k < \chi$ is not straightforward.

If the nodes are randomly k-coloured, the expected value of the unnormalized degree of conflict is k^{-1}. This suggests a normalization that is particularly useful when colourings using different number of colours are to be compared: the *normalized* degree of conflict G of a k-colouring is defined by $\Gamma \equiv k\gamma$. Using this metric, a random k-colouring has an expected value of 1. The unqualified phrase *degree of conflict* should be taken to refer to the normalized metric.

Note that a random colouring can be produced in a distributed environment without communication. A colourer that incurs communication costs can thus be required to reduce the normalized degree of conflict below 1 if it is to be considered useful.

3 A Decentralized, Synchronous, Stochastic Algorithm

The objective of a soft graph colourer is to quickly reduce the degree of conflict, optimally to 0 when the number of colours is at least the chromatic number. In a decentralized environment, computing the degree of conflict at run-time is not feasible because the communication and coordination costs are prohibitive.

However, it is assumed that each node is autonomous and can directly communicate with its neighbours (i.e., the constraint network is a subgraph of the communication network). So the general framework for colouring is one in which each node determines its own colour and collaborates with its immediate neighbours to reduce conflicts.

Specifically, if each node informs its neighbours when it changes its colour, then each node can compute how many conflicts it currently has with its neighbours, and strive to reduce this number. If every node manages to achieve zero conflicts with its neighbours, then the (global) degree of conflict will also be reduced to zero (since there will be no conflicts in the graph).

This is the basis for the synchronous, decentralized k-colouring algorithm shown in Figure 1. Each node initially chooses a colour at random. Then the nodes repeatedly update their colours in synchronized steps.

In each step, each node decides whether or not to *activate* by comparing a randomly generated number with some *activation probability*. If the node activates, then it chooses a colour that minimizes the number of colour conflicts that it has with its neighbours based on their colours in the previous step. Those nodes that change colour inform their neighbours: all of a node's operations are thus based on information that it has available locally.

One danger in this synchronized algorithm is that neighbours can perform colour changes simultaneously, so while one node is striving to accommodate its neighbours' choices of colours, those neighbours may also be striving to accommodate its colour choice. Thus, it is possible that by trying to reduce conflicts, simultaneously activated neighbours preserve or even introduce conflicts.

For a trivial example, consider a 2-colouring of the two node graph shown in Figure 2. Initially, both nodes have colour 1 (by chance); then both nodes activate, and both adopt colour 0, since that was the one colour unused by their neighbour(s).

In such a case, the nodes may be said to be acting incoherently. Incoherence could be eliminated by imposing a total order on the nodes so that only one changes at any given step. However, such a sequential solution is not scalable, and is overly severe: the algorithm, in general, seems to be robust enough to tolerate *some* level of incoherence.

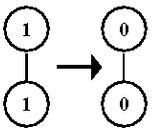

Fig. 2. Incoherent colouring

What is needed is a method for balancing parallel activity against the probability that two neighbours will activate simultaneously. This balance can be achieved by adjusting the activation probabilities (see Step 1 of Figure 1).

Unfortunately, the authors do not know of any analytical way of determining optimal activation probabilities *a priori*. Nevertheless, experiments with a simple algorithm, the Fixed Probability colourer (FP), have yielded some interesting results and suggest that the general framework is useful.

3.1 The Fixed Probability Colourer

In the FP(α) algorithm, the activation probabilities α_i are all set uniformly to the constant α. Figure 3 shows FP's typical behaviours over 10000 steps for various values of α (note that the step axis is logarithmic and that the experiments were abbreviated for $\alpha = 0.9$). The best values for α, in this case, are around 0.3–0.5; 0.3 was chosen for most of the experiments reported below.

The behaviours are averages over 20 graphs of various sizes (in the range 1000 to 5000 nodes) in which the nodes are arranged in a regular 2-dimensional grid

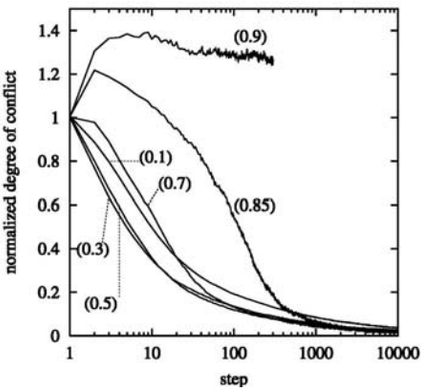

Fig. 3. Effect of activation probability on FP, 2D grids

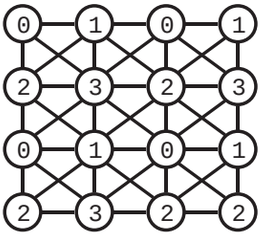

Fig. 4. A 2D grid

and the edges occur between nodes that are adjacent along an axis or diagonal (see Figure 4). The chromatic number of such grids (of large enough size) is 4.

Experiments yielding similar results were also performed with 3-dimensional grids (having edges along axes and diagonals, mean degree 26 and chromatic number 8) and with random 10-colourable graphs of mean degree 50. The random graphs were constructed by randomly colouring the nodes with 10 colours and then randomly generating the requisite number of edges between randomly chosen nodes of different colours (and, of course, finally discarding the colouring). This method of construction ensures that the graph's chromatic number is no more than 10, and is likely exactly 10.

These random graphs are similar to those reported used in other colouring experiments, with the difference that typically the edge probability is fixed, rather than the mean degree. The latter is more convenient for the experiments reported here, as the behaviour of the colourer seems to be quite independent of the number of nodes when the mean degree is fixed, whereas the mean degree varies with the number of nodes when the edge probability is fixed.

Referring again to Figure 3, for $\alpha = 0.9$, the colourer performs worse than random; i.e., the degree of conflict that it produces is higher than the expected value for randomly colouring the nodes. By measuring the number of nodes that change in each step, it can be determined that not only is the colouring extremely poor, but also that it is constantly changing. Such behaviour — a high degree of change but no improvement — may be called *thrashing*, and it is probably as undesirable a behaviour as can be produced, for the colourer not only delivers a low quality colouring to its client, it also consumes a large amount of system resources (for communication and perhaps computation).

For $\alpha = 0.85$, the degree of conflict eventually reduces to a low level (as low as for smaller values of α). However, there is a significant period (steps 1 to 100, say) during which the degree of conflict is high relative to that achieved with

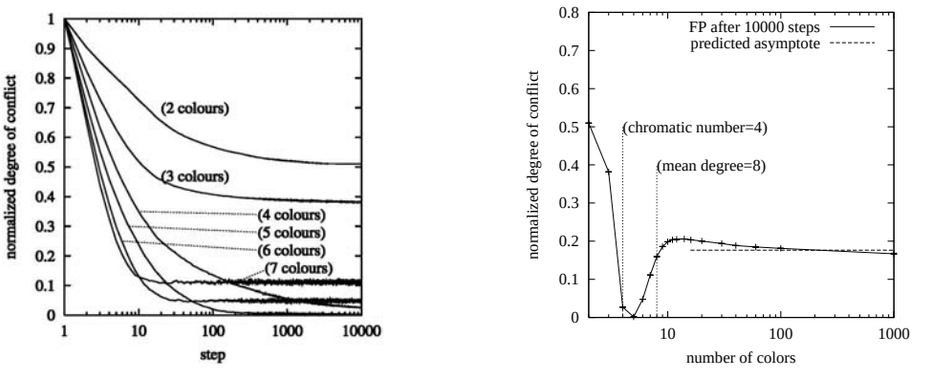

Fig. 5. Effect of number of colours on FP, 2D grids

smaller α. This is considered undesirable behaviour since during that period, the colouring is of low quality and will degrade the performance of whatever client process uses the colouring.

For $\alpha = 0.1$, the degree of conflict reduces smoothly to a low value. This is probably acceptable behaviour, but note that the degree of conflict can be reduced more quickly to an equally low value by choosing a slightly higher α — this would be better behaviour.

In other words, the objective is to choose an α that nimbly reduces the degree of conflict to a low level. This objective is made more interesting due to local minima: it can happen that lower values of α cause the colourer to become trapped in a local minimum that would be quickly escaped if α were significantly higher (due to extra incoherence — this is similar to a downward step in a hill-climbing algorithm). Whether or not the possibility of better long-term values outweighs the possibility of worse short-term values can really only be decided in the context of a specific application that provides concrete quality metrics.

3.2 Effect of Number of Colours

A k-colouring problem may be characterized based on the number of colours k relative to the chromatic number χ of the graph to be coloured: a problem is over-constrained when $k < \chi$, critically constrained when $k = \chi$, and under-constrained when $k > \chi$; a problem may also be characterized as loosely constrained when $k >> \chi$.

The performance of FP on these types of problem is shown in Figure 5: the left plot shows the degree of conflict as the colouring algorithm proceeds, while the right plot shows the mean of the degree of conflict achieved after 10000 steps.

It can immediately be seen that FP's performance is disappointing on loosely constrained problems: FP performs worse than on critically constrained problems, producing asymptotic degrees of conflict that are significantly above zero. Moreover, this asymptotic value initially increases with the number of colours, leveling off at very high numbers of colours.

This effect can be explained as follows. Consider the step of the FP algorithm in which a node chooses a colour that minimizes its conflicts with its neighbours. When the number of colours k is much greater than the chromatic number χ, the number of colours being used by any given node's neighbours is likely to be small compared with number of colours available, so each node will have a large number of zero-conflict colours from which to choose. It chooses one of these at random. Thus, it behaves partly like a random colour and, if all of the nodes were activating on each step, the expected value of the normalized degree of conflict would be 1.

A simple probabilistic analysis can account for the activation probability and gives $\Gamma \approx k\alpha/[(k-\delta)(2-\alpha)]$ where $\delta << k$ is a small correction to account for the average number of colours used in a neighbourhood. Experimentally, it is found that Γ is close to the asymptotic value predicted by this formula, $\alpha/(2-\alpha)$, for large numbers of colours, as shown in Figure 5 (right).

In the next two sections, possible improvements to FP are considered.

3.3 Deterministic Fixed Probability Colourer

One way to improve FP's under-constrained behaviour is to make the choice of colour deterministic. For example, step (3a) of the algorithm (Figure 1) can be modified to choose the smallest colour that minimizes the number of conflicts; denote the resulting algorithm as DFP.

However, as shown in Figure 6, this simple form of deterministic colour choice also has undesirable behaviour when under-constrained: while the convergence value of the degree of conflict does reduce to zero, as intended, the degree of conflict temporarily rises to extreme values during the first few steps.

A proposed explanation for these short-term peaks is as follows: after random initialization with a high number of colours, there will be many sets of neighbours that have the same smallest, optimal colour. When such sets of neighbours

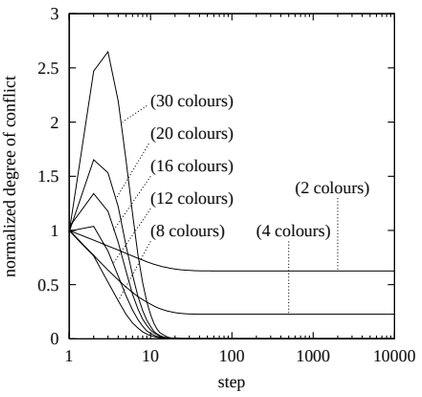

Fig. 6. DFP on 2D grids

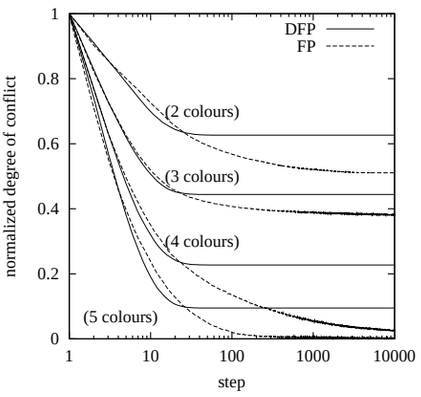

Fig. 7. DFP vs. FP

activate, they will introduce conflicts. (Note that the peak occurs around steps 2 and 3, by which time a majority of the nodes are expected to have activated, since the activation probability is 0.3.)

This short-term, aberrant behaviour can be ameliorated, if not quite eliminated, by biasing the nodes' colour choices in a non-uniform manner. For example, each node can be given a unique integer identity tag (or generate a random integer at startup). When choosing a colour, it chooses the colour that, primarily, minimizes the conflicts and, secondarily, minimizes the value of (identity+colour) modulo the number of colours.

However, there remains another problem with DFP, one that is common to deterministic hill-climbing algorithms: it is much more likely than non-deterministic algorithms to become stuck in a local minimum. Figure 7 compares DFP with FP for small numbers of colours — while their initial rates of reducing the degree of conflict are similar, FP continues the reduction long after DFP stabilizes. (By measuring the number of nodes changing colour, it can be confirmed that DFP's colouring becomes fixed, not just its degree of conflict.)

3.4 Conservative Fixed Probability Colourer

The basic characteristic of FP that causes poor performance on underconstrained problems is that it continues to change colour even when it has no conflicts with its neighbours. In a hill-climbing algorithm, this is not necessarily a bad characteristic: prematurely fixing colours may cause the algorithm to become stuck in a local minimum, as in DFP.

Nevertheless, FP can be simply modified so that a node will not activate if it has no conflicts with its neighbours. Denote the modified algorithm as CFP (*Conservative* Fixed Probability). Figure 8 (left) compares the behaviour of FP(0.3) and CFP(0.3) for numbers of colours ranging from 2 to 5 on graphs with chromatic number 4 — their performances are virtually indistinguishable. Figure 8 (right) compares the performance for higher numbers of colours — CFP performs much better, quickly eliminating all conflicts.

These experiments suggest that CFP does not suffer from local minima (at least no more than FP). In light of its much better performance for underconstrained colourings, the rest of this paper will deal only with CFP.

4 Assessment of CFP

Figure 9 summarizes the performance of CFP for an activation probability of 0.3. In this section, some further details of this performance are noted, and then the CFP colourer is assessed with respect to the desired characteristics listed earlier: cost, scalability and robustness.

When the number of colours is equal to the chromatic number, 4, the degree of conflict is reduced to a low value, 0.03, but not to zero. When the number of colours is 5, the degree of conflict is reduced to 6×10^{-5}. For higher numbers of colours, the degree of conflict is quickly reduced to exactly zero (after a about

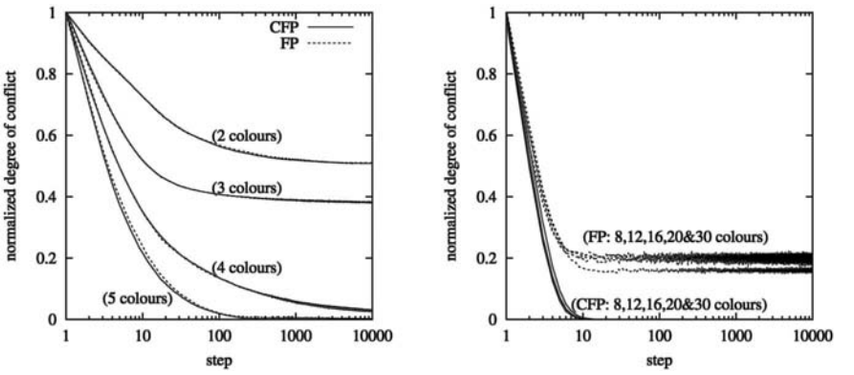

Fig. 8. CFP vs. FP

140 steps for 7 colours, and 40 steps for 8 colours). This is typical of the colourer: it first achieves proper colourings slightly above the chromatic number.

When the number of colours used to colour a graph is less than the graph's chromatic number, the degree of conflict cannot be reduced to zero. Nevertheless, it is still to be hoped that a colourer will manage to get close to the minimum. This presents a problem when assessing the performance of a colourer, since typically the minimum is unknown from analytical considerations, and finding it using, for example, branch and bound search is not likely to be feasible given the graph sizes.

However, some regular graphs, such as the n-dimensional grids, do yield at least lower bounds to analysis. While there is a risk that a colourer's performance on these graphs is not representative of its performance on more general graphs, it is the best information available so far.

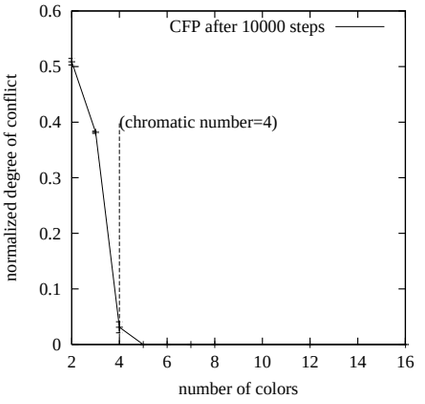

Fig. 9. Performance of CFP

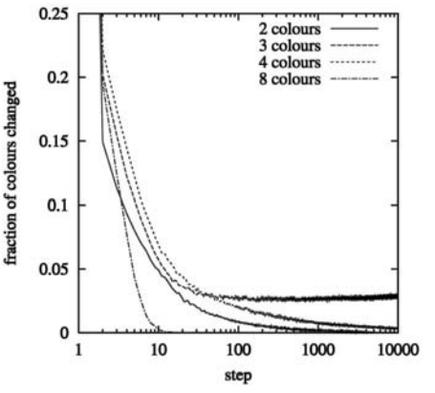

Fig. 10. Communication costs of CFP

For the (infinite) 2D grids, the chromatic number is 4. For a 2-colouring, the minimum degree of conflict possible is exactly 0.5; after 10000 steps, CFP has reduced the degree of conflict to 0.509 on average. For a 3-colouring, the minimum degree of conflict is exactly 0.375; FP achieves a mean of 0.382.

Although preliminary, these results, combined with those for 3D grids and random graphs, suggest that CFP performs well on sparse graphs.

4.1 Communication Costs

In the absence of concrete cost metrics for particular applications, the primary cost metric for a decentralized colourer is the amount of communication it uses. This is measured as the fraction of nodes that change colour per step, since each such change requires communicating the new colour to each neighbour.

Figure 10 shows the communication costs incurred by CFP(0.3) to colour 2D grids with various numbers of colours. The general pattern for communication costs is that the costs are initially large after random initialization while many conflicts are resolved. As the colouring stabilizes, the costs reduce.

When the colouring is over constrained, the degree of conflict cannot be reduced to zero and CFP will, in general, continue to try to improve the colouring even if it achieves an optimal colouring. (This general rule may be violated by 2-colourings which can give rise to extremely stable, large-scale regions of proper colourings, with conflicts occurring only along their borders. In these cases, CFP may reach a stable colouring.)

Although the rate of colour change remains low, the fact that it does not reach zero should be considered a weakness of the CFP colourer — ideally, there would be a way to determine that a colouring, although improper, is nevertheless (near) optimal, and adapt the colourer's behaviour accordingly.

When the number of colours is equal to or just larger than the chromatic number, the communication costs settle down to a very low value. When the number of colours is significantly greater than the chromatic number, CFP rapidly achieves a proper, stable colouring and the communication costs drop to zero.

4.2 Scalability

CFP is a minor modification to FP so it should be clear from Figure 1 that CFP is scalable in the number of nodes: the per-node, per-step computational, storage and communication costs are dependent on the mean degree of the graph and the number of colours rather than on the number of nodes.

Of course, for some types of graph (e.g., complete graphs), the mean degree is proportional to the number of nodes. However, such graphs are not likely to arise in large sensor networks, since the network itself would likely not be scalable.

Experimentally, we find that for large, sparse graphs, the performance of the colourer (not just its costs) shows no dependence on the number of nodes. This justifies the use of averages over different graph sizes in reports of experiments.

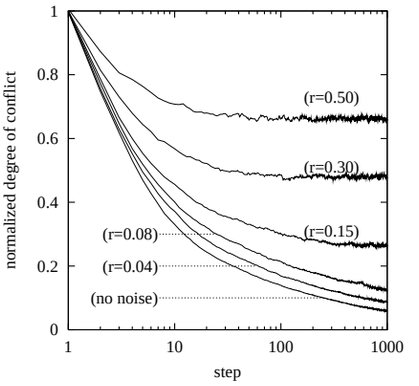

Fig. 11. Communication noise

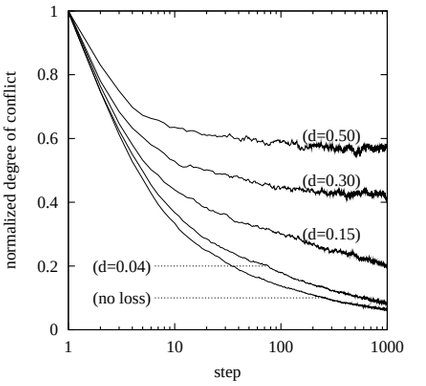

Fig. 12. Communication loss

4.3 Robustness

The robustness of CFP was explored in two ways: (1) observe the effect unreliable communication has on the degree of conflict: (2) observe the effect a dynamic graph topology has on the degree of conflict.

Unreliable Communication. In a distributed environment, each time a node changes colour it sends a message containing the new colour to its neighbours. These messages can be subjected to a random process in which they may be: corrupted by having the colour field changed randomly (with probability r); completely discarded (with probability d); or passed through unchanged.

Figure 11 shows the effect communication noise has on CFP(0.3) when communication loss is kept at zero, and Figure 12 shows the effect of communication loss when communication noise is kept at zero. In both cases, small values of noise/loss proportionately increase the degree of conflict. For large amounts of noise/loss, the increase is significant.

However, given that the colourer is operating in a distributed environment, this is to be expected — in the absence of any application-specific metrics, what is important is that the algorithm continues to function under unreliable communication rather than catastrophically failing.

Dynamic Topology. To partially simulate sensor failure and recovery, nodes and edges can be removed and reinserted into the graph over time. The process is designed so that if the rate/extent of change is small, then it is likely that structural aspects of the graph (e.g., the chromatic number) will not be much changed (such changes could complicate analysis of the experimental data).

Figure 13 shows the effect of continuous change at levels of less than 10% of the nodes per step — there is very little effect. Figure 14 shows a typical response to intermittent change: 20% of the nodes were affected every 30 steps. The degree

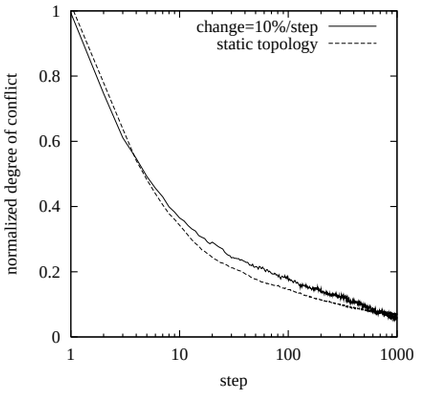

Fig. 13. Continuous topology change

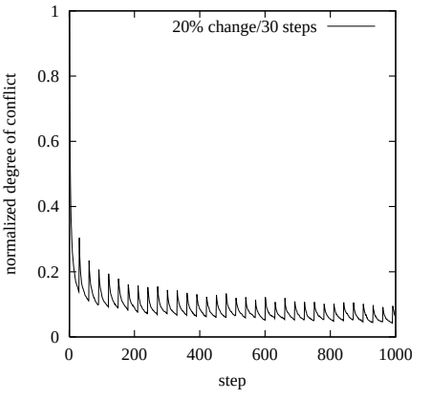

Fig. 14. Intermittent topology change

of conflict spikes immediately after each change, but it quickly and robustly decreases afterwards. Indeed, in this case, the degree of conflict continues to decrease in spite of the topology changes. However, it is to be expected that at sufficiently high levels of change and sufficiently short periods between changes, the colouring would degrade.

5 Related Work

The Deterministic Fixed Probability algorithm was, to the authors' knowledge, first published by Fabiunke [3] in the context of general distributed constraint satisfaction. He uses a randomization technique to break out of local minima and generally achieve proper colourings: however, the cost is that the degree of conflict remains high initially. In the framework of this paper, that is likely to be undesirable.

Yokoo at al. have published several algorithms for distributed constraint satisfaction [6]. They are concerned primarily with complete algorithms (i.e., algorithms guaranteed to find a feasible solution when one exists, and to terminate if one does not exist) and thus their algorithms are considerably more complex and incur considerable overheads to track the search space.

Lemaitre and Verfaillie [4] consider soft constraint optimization in a distributed setting, but their algorithm uses a central coordinating agent and seems to be sequential and unscalable. They do suggest that their algorithm can be parallelized, but do not consider details.

Aspects of iterative algorithms for hard colouring have been widely studied. Two papers that are particularly relevant are by Lewandowski and Condon [5], and Culberson [2]. However, hard colouring typically seems to deal with a time-intensive effort to (properly) colour a single, fixed graph using as few colours as possible. The ongoing reduction of conflicts in a dynamic graph does not seem to be as widely studied.

6 Conclusions and Future Work

This paper presents a simple framework for studying real-time constraint optimization in the form of soft graph colouring. It presents a set of criteria for assessing the performance of a distributed, soft colourer. It modifies an already published algorithm to enhance its performance over a wide range of problems and presents the results of experimental assessments.

The soft colourers were found to be scalable, low cost, robust and capable of responding in real-time. They should prove useful for resource coordination in large networks.

Although graph colouring is a restricted form of constraint satisfaction, it nevertheless may be conjectured that the algorithms discussed in this paper are readily extensible to the more general problem. However, it seems unnecessary at this stage of research to introduce additional details into the framework: there are still many interesting problems to be addressed in the simpler form.

Some of those problems include: determining optimal activation probabilities in a local manner; determining the minimum degree of conflict for over-constrained colourings; recognizing that an improper colouring is nevertheless an optimal colouring; dynamically adjusting the number of colours. Yet more problems, such as phase-transition phenomena, have presented themselves in initial investigations into colouring denser graphs. Finally, the algorithms can be extended to operate asynchronously.

References

1. *The ANTs Challenge Problem*, http://ants.kestrel.edu/challenge-problem/index.html
2. *Iterated Greedy Graph Colouring and the Difficulty Landscape*, Joseph Culberson, Technical Report TR 92-07, Department of Computing Science, The University of Alberta, Edmonton, Alberta, Canada, June 1992
3. *Parallel Distributed Constraint Satisfaction*, Marko Fabiunke, Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99), pp 1585-1591, Las Vegas, June 1999
4. *An Incomplete Method for Solving Distributed Valued Constraint Satisfaction Problems*, Michel Lemaitre & Gerard Verfaillie, AAAI-97, Workshop on Constraints and Agents, Providence, Rhode Island, USA, July 1997
5. *Experiments with Parallel Graph Colouring Heuristics*, Gary Lewandowski & Anne Condon, in *Cliques, Colouring and Satisfiability*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science , Volume 26, American Mathematical Society, 1996, pages 309-334
6. *The Distributed Constraint Satisfaction Problem: Formalization and Algorithms*, Makoto Yokoo, Edmund H. Durfee, Toru Ishida & Kazuhiro Kuwabara, IEEE trans. on Knowledge and Data Engineering, vol. 10, no. 5, September/October 1998

Appendix

A Motivation from Distributed Resource Management

The advent of small, simple, battery-powered sensors equipped with low-power, unreliable radio communication has stimulated interest in the deployment of large, distributed, autonomous networks of perhaps tens of thousands of such sensors [1].

To accomplish specific tasks (e.g., to track a moving target) several of the sensors typically must coordinate their actions (e.g., to simultaneously scan the same target to achieve accurate triangulation). The nature of the tasks may be such that real-time coordination is required (e.g., there is only a finite period over which a given sensor is capable of scanning a given moving target).

The number of tasks that a single sensor can service at any given time is bound to be limited (e.g., a sensor may be able to scan in only one direction at a time) so if several tasks impinge upon a small group of sensors (as may happen when several targets are close together) it becomes necessary to distribute the tasks to avoid overtasking sensors.

The number of tasks may vary dramatically over a short period. At times, the number of tasks is expected to approach or even exceed the theoretical capacity of the network, requiring the tasks to be carefully distributed to ensure that as many as possible are accomplished. At other times, the number of tasks is expected to be low, and the network is required to accomplish its tasks with a high degree of success and at low cost.

The cost may be measured, for example, in terms of the amount of battery energy expended or amount of communication required. (Communication may be considered to be costly because it may help alert an adversary to the presence and location of passive sensors.) The sensors may be required to operate in an unfavourable environment: sensors and associated computation nodes may fail and revive, and communication may be unreliable.

Given these considerations, a centralized coordination mechanism is not considerable feasible: it would be a computational and communication bottleneck. What is required is a decentralized coordination mechanism that is scalable, real-time, low-cost and robust, and that can perform well under dynamically varying task load.

A.1 Abstraction — Graph Colouring.

In its general form, sensor coordination can involve constraints on sensors and tasks, cost metrics on resource consumption, and quality metrics on task accomplishment that are arbitrarily complex. In this section, the problem is simplified so that it can be viewed as graph colouring. This simplification retains most of the features that make distributed coordination an interesting problem (since distributed graph colouring is just as computationally challenging), while providing a well-known context.

To view resource coordination as graph colouring, assume that we can impose a graph structure on the sensor network so that each task can be accomplished

by a single node within a predetermined time limit, and the constraints between the sensors give rise to mutual exclusion constraints between the nodes.

For example, a network of simple sensors can be reformulated into a graph in which the nodes are virtual sensors, each of which is an aggregate of three real sensors capable of triangulating a target. Two virtual sensors are not allowed to be active simultaneously if they share a real sensor (under the assumption that each real sensor can service only one task at a time).

Given a graph in which the nodes represent sensors and the edges represent mutual exclusion constraints, a proper node colouring of the graph represents a feasible schedule for the virtual resources:

- A colour can be viewed as a time period in a cyclic schedule. For example 'colour 3' in a 10-colouring may mean 'activate during periods 3-4 seconds, 13-14 seconds, 23-24 seconds, etc'.
- If two nodes are adjacent (i.e., connected by an edge) and they have the same colour, then the virtual resources represented by the nodes are scheduled for simultaneous activation even though they are mutually exclusive. An edge joining two nodes of the same colour may thus be considered to be a conflict.
- A proper colouring, by definition, has no conflicts. Thus, no two mutually exclusive virtual resources are active simultaneously, and the colouring represents a feasible schedule.

Given its operating environment, it is to be expected that the coordination mechanism may fail to construct a proper colouring and still respond in real-time. A flawed colouring (i.e., one that contains conflicts) is still of use to the sensor network: some of the tasks will fail, because a conflict represents an unfulfillable request for a real sensor to service two tasks at the same time, but some of the tasks will succeed. The fewer the conflicts the better.

It is considered better for the coordination mechanism to deliver a (somewhat) flawed colouring on time than to wait until it has constructed a proper colouring, since this may involve an intolerable delay (given that graph colouring is NP hard, the graphs are large and the coordination mechanism uses high-latency, unreliable communication).

Moreover, while the sensors are executing the schedule represented by the flawed colouring, the coordination mechanism can continue to improve the colouring. In other words, the coordination mechanism can operate as an anytime process.

Finally, the number of colours used to colour the graph directly determines the length of the cyclic schedule, since each colour represents a fixed period of time (determined by how long a sensor needs to scan a target once — the *dwell time*). If the schedule length is too long, then there is a good chance that a given target will be able to move entirely though a given sensor's scanning range without that sensor becoming active — this is not desirable (the *revisit period* is too high). Thus, the number of colours may be fixed by physical considerations.

Randomized Branching Programs

Martin Sauerhoff

DIRO, Université de Montréal, Montréal, Canada and
FB Informatik, LS2, Universität Dortmund, 44221 Dortmund, Germany
sauerhof@ls2.cs.uni-dortmund.de

Abstract. Branching programs are a graphical representation of Boolean functions which are considered as a nonuniform model of computation in complexity theory and are also used as a data structure in practice. The talk discusses randomized variants of branching programs which allow to study the relative power of deterministic, nondeterministic, and randomized algorithms in a scenario where space is the primary resource.

Keywords: Randomized branching program, OBDD, read-k-times, linear-length, nondeterminism, randomness, lower bounds.

1 Introduction

Theoretical computer science has been quite successful in inventing many useful, practically relevant formal models of computation. Nevertheless, theory has embarrassingly failed so far in the task of characterizing the relative power of the most basic models of computation. The classical "P $\neq$ NP"-problem of complexity theory, i. e., the question of whether NP-complete problems admit efficient, polynomial time algorithms, is unsolved now for over 30 years and will perhaps remain so for a much longer time. The reason for this is our profound lack of sufficiently powerful techniques for proving lower bounds on complexity measures such as time or space on Turing machines.

In the meantime, the techniques for designing algorithms, i. e., for proving upper bounds, have become fairly advanced. Randomized algorithms have turned out to be a powerful tool in many different areas of application. For some problems, randomized algorithms are more efficient than the best known deterministic algorithms (this is still true, e. g., for primality testing, for which polynomial time deterministic algorithms only exist under unproven number theoretical assumptions). If the error probability of a randomized polynomial time algorithm for a decision problem can be bounded by a constant smaller than 1/2, then it can be further decreased by a polynomial number of independent repetitions to be of the order $2^{-\mathrm{Poly}(n)}$, n the input size. This is as good as a deterministic algorithm for many applications. Hence, it is justified to identify the notion of an efficient algorithm with a randomized polynomial time algorithm with error probability bounded by a constant smaller than 1/2. The complexity class BPP (bounded error probabilistic polynomial time) contains all decision problems which can be solved by such algorithms.

K. Steinhöfel (Ed.): SAGA 2001, LNCS 2264, pp. 65–71, 2001.

Given the importance of randomized algorithms, we face some more "modern" open questions in complexity theory in addition to the classical "P $\neq$ NP" problem. It is desirable to know whether randomized algorithm can really be *provably* more powerful than deterministic ones. In theoretical terms, this is the open problem of whether P $\subsetneq$ BPP. Somewhat surprisingly, there are some powerful derandomization techniques (see, e. g., Impagliazzo and Wigderson [11], Wigderson [22], and Andreev et al. [5,6]) which may be seen as a support for the belief that in fact P = BPP. Furthermore, it would be important for practical reasons to know whether NP-complete problems can be efficiently solved by randomized algorithms or not. In the language of complexity theory, this is the open problem of whether NP $\subseteq$ BPP. This is unlikely to be the case, since Ko [12] has shown that NP $\subseteq$ BPP would imply that all classes in the so-called polynomial time hierarchy are identical to BPP (and most people believe that this is not true).

To sum up, we do not have definite answers even to the basic questions concerning the relationship between the classes P, NP, and BPP, since we lack appropriate lower bound techniques for obtaining separation results. One approach in this situation is to establish results under yet unproven (but plausible) assumptions (we have mentioned examples above). In particular, relativized results belong into this category. On the other hand, one may also study alternative models of computation which promise to be easier to handle by combinatorial tools than the somewhat "unwieldy" classical Turing machine.

The latter approach has been quite successful for some nonuniform models of computation, such as circuits, communication protocols, and branching programs. A nonuniform model of computation allows to specify a sequence of algorithms, one for each input length n, for computing a sequence of Boolean functions $f_n \colon \{0,1\}^n \to \{0,1\}$. Branching programs describe sequential computations in an especially handy way. Furthermore, complexity classes defined in terms of other well-known nonuniform models of computation may be equivalently characterized in terms of branching programs.

Definition 1. *A* (deterministic) branching program (BP) *on the variable set* $\{x_1, \ldots, x_n\}$ *is a directed acyclic graph with a single source node and sinks labeled by one of the constants* 0 *and* 1*. Each internal (non-sink) node is labeled by a variable* x_i *and has exactly two outgoing edges carrying labels* 0 *and* 1*, resp. This graph represents a Boolean function* $f \colon \{0,1\}^n \to \{0,1\}$ *in the following way. To compute* $f(a)$ *for some input* $a \in \{0,1\}^n$*, start at the source node. For a non-sink node labeled by* x_i*, determine the value of this variable and follow the edge which is labeled by this value (this is called a "test of variable* x_i*"). Iterate this until a sink node is reached. The value of* f *on input* a *is the value of the reached sink. For a fixed input* a*, the sequence of nodes visited in this way is uniquely determined and is called the* computation path for a*. The* size *of a branching program* G *is the number of its nodes. The* branching program size *of a function is the minimal size of a branching program representing it.*

Usually, we consider sequences of branching programs representing sequences $(f_n)_{n\in\mathbb{N}}$ of Boolean functions, where $f_n\colon \{0,1\}^n \to \{0,1\}$. Somewhat sloppily, we often talk of functions where we really mean "sequences of functions."

It is a well-known fact that sequences of functions which can be computed by branching programs of polynomial size can also be computed within logarithmic space on the nonuniform variant of Turing machines and vice versa (Cobham [10], Pudlák and Žák [14]). In terms of complexity classes, this means that the class of sequences of functions with polynomial branching program size is equal to the class L/Poly. Proving a superpolynomial lower bound on the branching program size for, e. g., a sequence of functions $(f_n)_{n\in\mathbb{N}}$ whose corresponding language $L_f := \bigcup_{n\in\mathbb{N}} f_n^{-1}(1)$ is contained in P would in particular establish that $\mathrm{L} \subsetneq \mathrm{P}$, resolving another main open problem of complexity theory. On the practical side, this result would imply that no P-complete problem can be efficiently solved by a parallel algorithm.

So far, no superpolynomial lower bound on the branching program size of concrete functions are known. Nevertheless, superpolynomial and exponential lower bounds could be proven for several restricted variants of branching programs. One major goal in branching program theory is to extend the available techniques to more and more general models. The last years have brought some astonishing progress along this line. Here, we consider the following restricted variants of branching programs.

- An *OBDD* (*ordered binary decision diagram*) is a branching program which is associated with an arbitrarily permuted list of the input variables, called the *variable order* of the OBDD. It is required that the list of variables on each path from the source to one of the sinks in the graph is a sublist of the variable order.
- A *read-k-times branching program* is a branching program where each variable may appear at most k times on each path from the source to a sink. For $k = 1$, we call the respective graphs *read-once branching programs*. OBDDs are obviously special read-once branching programs.
- A *linear-length branching program* is a branching program where the number of variables on each computation path is required to be linear in the number of input variables. Notice that this restriction does not extend to so-called *inconsistent paths*, i. e., paths where the same variable is tested with different results and which thus cannot be computation paths for any input assignment.

OBDDs are one of the most restricted variants of branching programs, but have turned out to be extremely useful in practice as a data structure for Boolean functions. The reason is that they allow to compactly represent many practically relevant functions and at the same time can be efficiently manipulated by algorithms, e. g., for computing the disjunction or conjunction of representations and for checking satisfiability or equivalence.

Read-once branching programs are the restricted type of branching programs for which the first exponential lower bounds on the size could be proven (Žák [23], Wegener [20]) and have been extensively been studied since then.

Traditionally, lower bounds for read-k-times models have been investigated as the first step towards time-space tradeoffs. The length of the paths in a branching program corresponds to the computation time. In a read-k-times branching program, this length is obviously bounded by kn. The first exponential lower bounds for read-k-times branching programs for values of k larger than 1 (in fact, for k being allowed to be a logarithmic function of the input size) have been proven by Borodin, Razborov, and Smolensky [9] and by Okolnishnikova [13].

The latest records in the competition for lower bounds on the size of less and less restricted variants of branching programs have been achieved for linear-length branching programs. Beame, Saks, and Thathachar [8],Ajtai [3,4] and Beame, Saks, Sun, and Vee [7] have proven exponential lower bounds for this model. Their results also give the first non-trivial time-space tradeoffs for the general model of branching programs. Furthermore, some of the respective lower bounds even hold for the randomized variant of linear-length branching programs, as discussed later on.

For more background information and an extensive survey of results for all variants of branching programs, we refer to the monograph of Wegener [21].

2 Randomized Branching Programs

Given the polynomial relationship between the size of branching programs and the logarithm of the space complexity for nonuniform Turing machines, branching programs are also a suitable model for comparing the power of determinism, nondeterminism, and randomness in the nonuniform, space-restricted setting. In the following, we introduce randomized and nondeterministic variants of branching programs.

Definition 2. *A* randomized branching program *is a branching program with the following additional properties: (i) The graph may contain unlabeled nodes with two unlabeled outgoing edges. (ii) A randomized branching program may contain three sinks labeled by* 0, 1, *and "?". Nodes labeled by variables are called* decision nodes*, while the unlabeled nodes are called* randomized nodes*. An input a defines a randomized computation path in the graph from the source to a sink as follows. At decision nodes, the successor is chosen as for deterministic branching programs. At randomized nodes, each of the two successors is chosen with probability* $1/2$*, and this decision is independent from all other random decisions. Let $G(a)$ be the random variable which is equal to $r \in \{0, 1, ?\}$ if an r-sink is reached by the randomized computation path for input a. We consider the following modes of acceptance. We say that G computes a function f defined on the decision variables*

- *with* bounded (two-sided) error ε, $0 \leq \varepsilon < 1/2$, *if* $\Pr\{G(a) = f(a)\} \geq 1 - \varepsilon$ *for all inputs a;*
- *with* one-sided error ε, $0 \leq \varepsilon < 1$, *if* $\Pr\{G(a) = 1\} \geq 1 - \varepsilon$ *for all* $a \in f^{-1}(1)$ *and* $\Pr\{G(a) = 0\} = 1$ *for all* $a \in f^{-1}(0)$*;*

– nondeterministically *(or with* unbounded one-sided error*), if* $\Pr\{G(a) = 1\} > 0$ *for all* $a \in f^{-1}(1)$ *and* $\Pr\{G(a) = 0\} = 1$ *for all* $a \in f^{-1}(0)$*;*
– *with* zero error and failure probability ε*,* $0 \leq \varepsilon < 1$*, if* $\Pr\{G(a) = \neg f(a)\} = 0$ *and* $\Pr\{G(a) = ?\} \leq \varepsilon$ *for all inputs* a*.*

In the nondeterministic case, randomized branching programs are called nondeterministic branching programs *and unlabeled nodes are called* nondeterministic nodes*.*

Complexity classes for branching programs analogous to P, NP, BPP etc. are defined by replacing polynomial time complexity with polynomial branching program size in the respective model. Due to the difficulty of proving separation results, it is not too surprising that already for the more restricted variants of branching programs there are several open problems concerning the relationship between the different modes of acceptance.

3 A Collection of Results

In the talk, some exemplary results on the complexity of randomized OBDDs, read-k-times branching programs, and linear-length branching programs are presented. This is by no means an exhaustive treatment. Instead, the presentation is intended to give an impression of the strengths and weaknesses of what we have been able to do so far for randomized branching programs. For more information, we refer to the literature and the monograph [21].

First of all, OBDDs turn out to be one of the few models in complexity theory for which the relative power of determinism, nondeterminism, and randomness can be characterized almost completely. The investigation of the randomized variant of OBDDs has been initiated by Ablayev and Karpinski [2] who have presented a concrete function for which randomized OBDDs require only polynomial size, while every deterministic OBDD requires exponential size. Further upper bounds for randomized OBDDs have been presented by the author [16]. All these upper bounds are based on the so-called fingerprinting technique. Lower bounds for randomized OBDDs have been shown by Ablayev [1] and the author [17] using results from communication complexity.

It is much harder to prove exponential lower bounds already for randomized and nondeterministic variants of read-once branching programs. The first exponential lower bounds for randomized read-once and read-k-times branching programs (for k bounded by a logarithmic function in the input size) have been achieved by the author in [15,18]. The technique is based on ideas due to Borodin, Razborov, and Smolensky [9] for the nondeterministic case and proceeds in two steps. First, one uses the structure of the given randomized read-k-times branching program to obtain a partition of the input space of the represented function into "well-structured" subsets, which are suitable variants of the combinatorial rectangles from communication complexity theory. In the second step, one then derives a lower bound on the number of rectangles in such a representation by

combinatorial means. Usually, the second step is based on discrepancy lower bounds obtained by algebraic techniques.

The described proof technique has also been applied by Thathachar [19] to show that the classes of sequences of functions with polynomial size read-k-times branching programs form a proper hierarchy with respect to k. Furthermore, by more sophisticated constructions, it is also possible to reduce the structure of a linear-length branching program to a combinatorial problem defined in terms of an appropriate variant of combinatorial rectangles. This is the idea behind the more recent results for linear-length branching programs due to Beame, Saks, and Thathachar [8], Ajtai [3,4] and Beame, Saks, Sun, and Vee [7], where the latter paper even contains exponential lower bounds for the randomized variant of this model.

Studying the randomized mode of computation for restricted branching programs has already provided new insights into the power of randomized algorithms which went beyond what we already knew from communication complexity theory. Nevertheless, the results obtained so far for the more general variants of branching programs all work only for rather artificial toy functions. There is still much to be done to solve the lower bound problem for real-life functions and to improve nearly all parameters in the results we have, like, e. g., the bounds on the error probability. Furthermore, lower bounds for general branching programs remain still elusive.

References

1. F. Ablayev. Randomization and nondeterminism are incomparable for polynomial ordered binary decision diagrams. In *Proc. of th 24th Int. Coll. on Automata, Languages, and Programming (ICALP)*, *LNCS 1256*, 195–202. Springer, 1997.
2. F. Ablayev and M. Karpinski. On the power of randomized branching programs. In *Proc. of the 23rd Int. Coll. on Automata, Languages, and Programming (ICALP)*, *LNCS 1099*, 348–356. Springer, 1996.
3. M. Ajtai. Determinism versus non-determinism for linear time RAMs with memory restrictions. In *Proc. of the 31st Ann. ACM Symp. on Theory of Computing (STOC)*, 632–641, 1999.
4. M. Ajtai. A non-linear time lower bound for Boolean branching programs. In *Proc. of the 40th IEEE Symp. on Foundations of Computer Science (FOCS)*, 60–70, 1999.
5. A. E. Andreev, A. E. F. Clementi, J. D. P. Rolim, and L. Trevisan. Weak random sources, hitting sets, and BPP simulations. In *Proc. of the 38th IEEE Symp. on Foundations of Computer Science (FOCS)*, 264–272, 1997.
6. A. E. Andreev, A. E. F. Clementi, J. D. P. Rolim, and L. Trevisan. Weak random sources, hitting sets, and BPP simulations. *SIAM J. Comp.*, 28(6):2103–2116, 1999.
7. P. Beame, M. Saks, X. Sun, and E. Vee. Super-linear time-space tradeoff lower bounds for randomized computation. In *Proc. of the 41st IEEE Symp. on Foundations of Computer Science (FOCS)*, 2000.
8. P. Beame, M. Saks, and J. S. Thathachar. Time-space tradeoffs for branching programs. In *Proc. of the 39th IEEE Symp. on Foundations of Computer Science (FOCS)*, 254–263, 1998.

9. A. Borodin, A. A. Razborov, and R. Smolensky. On lower bounds for read-k-times branching programs. *Computational Complexity*, 3:1–18, 1993.
10. A. Cobham. The recognition problem for the set of perfect squares. In *Proc. of the 7th Symposium on Switching an Automata Theory (SWAT)*, 78–87, 1966.
11. R. Impagliazzo and A. Wigderson. P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *Proc. of the 29th Ann. ACM Symp. on Theory of Computing (STOC)*, 220–228, 1997.
12. K.-I. Ko. Some observations on the probabilistic algorithms and NP-hard problems. *Information Processing Letters*, 14(1):39–43, Mar. 1982.
13. E. A. Okol'nishnikova. On lower bounds for branching programs. *Siberian Advances in Mathematics*, 3(1):152–166, 1993.
14. P. Pudlák and S. Zák. Space complexity of computations. Technical report, Univ. Prague, 1983.
15. M. Sauerhoff. Lower bounds for randomized read-k-times branching programs. In *Proc. of the 15th Ann. Symp. on Theoretical Aspects of Computer Science (STACS)*, *LNCS 1373*, 105–115. Springer, 1998.
16. M. Sauerhoff. *Complexity Theoretical Results for Randomized Branching Programs*. PhD thesis, Univ. of Dortmund. Shaker, Aachen, 1999.
17. M. Sauerhoff. On the size of randomized OBDDs and read-once branching programs for k-stable functions. In *Proc. of the 16th Ann. Symp. on Theoretical Aspects of Computer Science (STACS)*, *LNCS 1563*, 488–499. Springer, 1999. To appear in *Computational Complexity*.
18. M. Sauerhoff. Approximation of Boolean functions by combinatorial rectangles. Technical Report 58, Electr. Coll. on Comp. Compl., 2000.
19. J. Thathachar. On separating the read-k-times branching program hierarchy. In *Proc. of the 30th Ann. ACM Symp. on Theory of Computing (STOC)*, 653–662, 1998.
20. I. Wegener. On the complexity of branching programs and decision trees for clique functions. *Journal of the ACM*, 35(2):461–471, Apr. 1988.
21. I. Wegener. *Branching Programs and Binary Decision Diagrams—Theory and Applications*. Monographs on Discrete and Applied Mathematics. SIAM, Philadelphia, PA, 2000.
22. A. Wigderson. De-randomizing BPP: The state of the art. In *Proc. of the 14th IEEE Int. Conf. on Computational Complexity*, 1999.
23. S. Žák. An exponential lower bound for one-time-only branching programs. In *Proc. of the 11th Int. Symp. on Mathematical Foundations of Computer Science (MFCS)*, *LNCS 176*, 562–566. Springer, 1984.

Yet Another Local Search Method for Constraint Solving

Philippe Codognet[1] and Daniel Diaz[2]

[1] University of Paris 6, LIP6, 8 rue du Capitaine Scott, 75 015 Paris, FRANCE
Philippe.Codognetlip6.fr
[2] University of Paris I, CRI, Bureau C1407, 75 634 Paris Cedex 13, FRANCE
Daniel.Diaz@inria.fr

Abstract. We propose a generic, domain-independent local search method called adaptive search for solving Constraint Satisfaction Problems (CSP). We design a new heuristics that takes advantage of the structure of the problem in terms of constraints and variables and can guide the search more precisely than a global cost function to optimize (such as for instance the number of violated constraints). We also use an adaptive memory in the spirit of Tabu Search in order to prevent stagnation in local minima and loops. This method is generic, can apply to a large class of constraints (e.g. linear and non-linear arithmetic constraints, symbolic constraints, etc) and naturally copes with over-constrained problems. Preliminary results on some classical CSP problems show very encouraging performances.

Keywords: Local search, constraint solving, combinatorial optimization, search algorithms.

1 Introduction

Heuristic (i.e. non-complete) methods have been used in Combinatorial Optimization for finding optimal or near-optimal solutions since a few decades, originating with the pioneering work of Lin on the Traveling Salesman Problem [10]. In the last few years, the interest for the family of Local Search methods for solving large combinatorial problems has been revived, and they have attracted much attention from both the Operations Research and the Artificial Intelligence communities, see for instance the collected papers in [1] and [20], the textbook [12] for a general introduction, or (for the French speaking reader) [8] for a good survey. Although local search techniques have been associated with basic hill-climbing or greedy algorithms, this term now encompasses a larger class of more complex methods, the most well-known instances being simulated annealing, Tabu search and genetic algorithms, usually referred as "meta-heuristics". They work by iterative improvement over an initial state and are thus anytime algorithms well-suited to reactive environments. Consider an optimization problem with cost function which makes it possible to evaluate the quality of a given configuration (assignment of variables to current values) and a transition function that defines for each configuration a set of "neighbors". The basic algorithm

K. Steinhöfel (Ed.): SAGA 2001, LNCS 2264, pp. 73–89, 2001.

consists in starting from a random configuration, explore the neighborhood, select an adequate neighbor and then move to the best candidate. This process will continue until some satisfactory solution is found. To avoid being trapped in local optima, adequate mechanisms should be introduced, such as the adaptive memory of Tabu search, the cooling schedule of simulated annealing or similar stochastic mechanisms. Very good results have been achieved by dedicated and finely tuned local search methods for many problems such as the Traveling Salesman Problem, scheduling, vehicle routing, cutting stock, etc. Indeed, such techniques are now the most promising approaches for dealing with very large search spaces, when the problem is too big to be solved by complete methods such as constraint solving techniques.

In the last years, the application of local search techniques for constraint solving started to raise some interest in the CSP community. Localizer [13,14] proposed a general language to state different kinds of local search heuristics and applied it to both OR and CSP problems, and [18] integrated a constraint solving component into a local search method for using constraint propagation in order to reduce the size of the neighborhoods. GENET [4] was based on the Min-Conflict [16] heuristics, while [17] proposed a Tabu-based local search method as a general problem solver but this approach required a binary encoding of constraints and was limited to linear inequalities. Very recently, [7] developed another Tabu-based local search method for constraint solving. This method, developed independently of our adaptive search approach, also used so-called "penalties" on constraints that are similar to the notion of "constraint errors" that will be described later. It is worth noticing that the first use of such a concept is to be found in [2].

We propose a new heuristic method called Adaptive Search for solving Constraint Satisfaction Problem. Our method can be seen as belonging to the GSAT [22], Walksat [23] and Wsat(OIP) [26] family of local search methods. But the key idea of our approach is to take into account the structure of the problem given by the CSP description, and to use in particular variable-based information to design general meta-heuristics. This makes it possible to naturally cope with heterogeneous problem descriptions, closer to real-life application that pure regular academic problems.

Preliminary results on classical CSP benchmarks such as the simple "N-queens" problem or the much harder "magic square" or "all-intervals" problems show that the adaptive search method performs very well w.r.t. traditional constraint solving systems.

2 Adaptive Search

The input of the method is a problem in CSP format, that is, a set of variables with their (finite) domains of possible values and a set of constraints over these variables. A constraint is simply a logical relation between several unknowns, these unknowns being variables that should take values in some specific domain of interest. A constraint thus restricts the degrees of freedom (possible values)

the unknowns can take; it represents some partial information relating the objects of interest. Constraint Solving and Programming has proved to be very successful for Problem Solving and Combinatorial Optimization applications, by combining the declarativity of a high-level language with the efficiency of specialized algorithms for constraint solving, borrowing sometimes techniques from Operations Research and Numerical Analysis [21]. Several efficient constraint solving systems for finite domain constraints now exists, such as ILOG Solver [19] on the commercial side or clp(FD)[3] and GNU-Prolog [5] on the academic/freeware side. Although we will completely depart in adaptive search from the classical constraint solving techniques (i.e. Arc-Consistency and its extensions), we will take advantage of the formulation of a problem as a CSP. Such representation indeed makes it possible to structure the problem in terms of variables and constraints and to analyze the current configuration (assignment of variables to values in their domains) more carefully than a global cost function to be optimized, e.g. the number of constraints that are not satisfied. Accurate information can be collected by inspecting constraints (that typically involve only a subset of all the problem variables) and combining this information on variables (that typically appear in only a subset of all the problem constraints).

Our method is not limited to any specific type of constraint, e.g. linear constraints as classical linear programming or [26]. However we need, for each constraint, an *error function* that will give an indication on how much the constraint is violated. Consider a n-ary constraint $C(X_1, \ldots, X_n)$ and domains $D_1, \ldots, D_n$ for variables $\{X_1, \ldots, X_n\}$. An error function f_c associated to the constraint C is simply a real-valued function from $D_1 \times \ldots \times D_n$ such that $f_C(X_1, \ldots, X_n)$ has value zero if $C(X_1, \ldots, X_n)$ is satisfied. We do not impose the value of f_C to be different from zero when the constraint is not satisfied, and this will indeed not be the case in some of the examples we will describe below. The error function will in fact be used as a heuristic value to represent the "degree of satisfaction" of a constraint and thus to check how much a constraint is violated by a given tuple. For instance the error function associated to an arithmetic constraint $|X - Y| \leq C$ will be $max(0, |X - Y| - C)$. Adaptive search relies on iterative repair, based on variables and constraint errors information, seeking to reduce the error on the worse variable so far. The basic idea is to compute the error function of each constraint, then combine for each variable the errors of all constraints in which it appears, therefore projecting constraint errors on involved variables. Finally, the variable with the maximal error will be chosen as a "culprit" and thus its value will be modified. In this second step we use the well-known min-conflict heuristics [16] and select the value in the variable domain that has the best error immediate value, that is, the value for which the total error in the next configuration is minimal. This is similar to the steepest ascent heuristics for traditionnal hillclimbing.

In order to prevent being trapped in local minima, the adaptive search method also includes an adaptive memory as in Tabu Search : each variable leading to a local minimum is marked and cannot be chosen for the few next iterations. A local minimum is a configuration for which none of the neighbor im-

prove the current configuration. This corresponds in adaptive search to a variable whose current value is better than all alternative values in its domain. It is worth noticing that conversely to most Tabu-based methods (e.g. [6] or [7] for a CSP-oriented framework) we mark variables and not couples $< variable, value >$, and that we do not systematically mark variables when chosen in the current iteration but only when they lead to a local minimum. Observe however that, as we use the min-conflict heuristics, the method will never choose the same variable twice in a row.

It is worth noticing that the adaptive search method is thus a generic framework parametrized by three components :

- A family of error functions for constraints (one for each type of constraint)
- An operation to combine for a variable the errors of all constraints in which it appears
- A cost function for a evaluating configurations

In general the last component can be derived from the first two one. Also, we could require the combination operation to be associative and commutative.

3 General Algorithm

Let us first detail the basic loop of the adaptive search algorithm, and then present some extra control parameters to tune the search process.

Input :

Problem given in CSP form :

- a set of variables $V = \{V_1, V_2, \ldots, V_n\}$ with associated domains of values
- a set of constraints $C = \{C_1, C_2, \ldots, C_k\}$ with associated error functions
- a combination function to project constraint errors on variables
- a (positive) cost function to minimize

Output :

a sequence of moves (modification of the value of one of the variables) that will lead to a solution of the CSP (configuration where all constraints are satisfied) if the CSP is satisfied or to a partial solution of minimal cost otherwise.

Algorithm :

Start from a random assignment of variables in V

Repeat

1. Compute errors of all constraints in C and combine errors on each variable by considering for a given variable only the constraints on which it appears.
2. select the variable X (not marked as Tabu) with highest error and evaluate costs of possible moves from X
3. if no improving move exists
   then mark X tabu for a given number of iterations
   else select the best move (min-conflict) and change the value of X accordingly

until a solution is found or a maximal number of iterations is reached

Some extra parameters can be introduced in the above framework in order to control the search, in particular the handling of (partial) restarts. One first has to precise, for a given problem, the *Tabu tenure* of each variable, that is, the number of iteration a variable should not be modified once it is marked due to local minima. Thus, in order to avoid being trapped with a large number of Tabu variables and therefore no possible diversification, we decide to randomly reset a certain amount of variables when a given number of variables are Tabu at the same time. We thereafter introduce two other parameters : the *reset limit*, i.e. the number of simultaneous Tabu variables to reach in order to randomly reset a certain ratio of variables (*reset percentage*). Finally, as in all local search methods, we parametrize the algorithm with a maximal number of iterations (*max iterations*). This could be used to perform early restart, as advocated by [23]. Such a loop will be executed at most *max restart* times before the algorithm stops.

This method, although very simple, is nevertheless very efficient to solve complex combinatorial problems such as classical CSPs, as we will see in the next section. It is also worth noticing that this method has several sources of stochasticity. First, in the core algorithm, both in the selection of the variable and in the selection of the value for breaking ties between equivalent choices (e.g. choosing between two variables that have the same value for the combination of their respective constraint errors). Second, in the extra control parameters that have just been introduced, to be tuned by the user for each application. For instance if the reset limit (number of simultaneous tabu variables) is very low, the algorithm with restart very often, enhancing thus the stochastic aspects of the method; but on the other hand if the reset limit is too high, the method might show some trashing behavior and have difficulties in escaping local minima. Last but not least, when performing a restart, the algorithm will randomly modify the values of a given percentage of randomly chosen variables (the reset percentage). Thus a reset percentage of 100 % will amount to restart each time from scratch.

4 Examples

Let us now detail how the adaptive search method performs on some classical CSP examples. We have tried to choose benchmarks on which other constraint solving methods have been applied in order to obtain comparison data, but is it worth noticing that all these benchmarks have satisfiable instances. For each benchmark we give a brief description of the problem and its modeling in the adaptive search approach. Then, we present performance data averaged on 10 executions, including:

- instance number (i.e. problem size)
- average, best and worst CPU time
- total number of iterations (within a single run, on average)
- number of local minima reached (within a single run, on average)
- number of performed swaps (within a single run, on average)

– number of resets (within a single run, on average)

Then we compare those performance results (essentially the execution time) with other methods among the most well-known constraint solving techniques: constraint programming systems [5,19], general local search system [13,14], Ant-Colony Optimization [24]. We have thus favored academic benchmarks over randomly generated problems in order to compare to literature data.

Obviously, this comparison is preliminary and not complete but it should give the reader a rough idea of the potential of the adaptive search approach. We intend to make a more exhaustive comparison in the near future.

4.1 Magic to the Square

The magic square puzzle consists in placing on a NxN square all the numbers in $\{1, 2, \ldots, N^2\}$ such as the sum of the numbers in all rows, columns and the two diagonal are the same. It can therefore be modeled in CSP by considering N^2 variables with initial domains $\{1, 2, \ldots, N^2\}$ together with linear equation constraints and a global all_different constraint stating that all variables should have a different value. The constant value that should be the sum of all rows, columns and the two diagonals can be easily computed to be $N(N^2 + 1)/2$.

The instance of adaptive search for this problem is defined as follows. The error function of an equation $X_1 + X_2 + \ldots + X_k = b$ is defined as the value of $X_1 + X_2 + \ldots + X_k - b$. The combination operation is the absolute value of the sum of errors (and not the sum of the absolute values, which would be less informative : errors with the same sign should add up as they lead to compatible modifications of the variable, but not errors of opposite signs). The overall cost function is the addition of absolute values of the errors of all constraints The method will start by a random assignment of all N^2 numbers in $\{1, 2, \ldots, N^2\}$ on the cells of the NxN square and consider as possible moves all swaps between two values.

The method can be best described by the following example which depicts information computed on a 4x4 square:

Values and Constraint errors

				-8
11	7	8	*15*	7
16	2	4	12	0
10	**6**	5	3	-10
1	14	9	13	3
4	-5	-8	9	-3

Projections on variables

8	2	1	8
4	8	16	9
6	**23**	21	1
1	2	5	9

Costs of next configurations

39	54	51	*33*
53	67	61	41
45	57	57	66
77	43	48	41

The table on the left shows the configuration of the magic square at some iteration (each variable corresponds to a cell of the magic square). Numbers on the right of rows and diagonals, and below lines, denote the errors of the corresponding constraints. The total cost is then computed as the sum of the absolutes values of those constraints errors and is equal to 57. The table immediately on

the right shows the combination of constraint errors on each variable. The cell (3,2) with value **6** (in bold font on the left table) has maximal error (**23**) and is thus selected for swapping. We should now score all possible swaps with other numbers in the square; this is depicted in the table on the right, containing the cost value of the overall configuration for each swap. The cell (1,4) with value *15* (in italic) gives the best next configuration (with cost *33*) and is thus selected to perform a move. The cost of the next configuration will therefore be 33.

Table 1. Magic square results

problem instance	time (sec) avg of 10	time (sec) best	time (sec) worst	nb iterations	nb local minima	nb swaps	nb resets
10x10	0.13	0.03	0.21	6219	2354	3864	1218
20x20	3.41	0.10	7.35	47357	21160	26196	11328
30x30	18.09	0.67	52.51	116917	54919	61998	29601
40x40	58.07	10.13	166.74	216477	102642	113835	56032
50x50	203.42	44.51	648.25	487749	233040	254708	128625

Table 1 details the performances of this algorithm on bigger instances. For a problem of size NxN the following settings are used: Tabu tenure is equal to N-1 and 10 % of the variables are reset when $N^2/6$ variables are Tabu. The programs were executed on a PentiumIII 733 MHz with 192 Mb of memory running Linux RedHat 7.0.

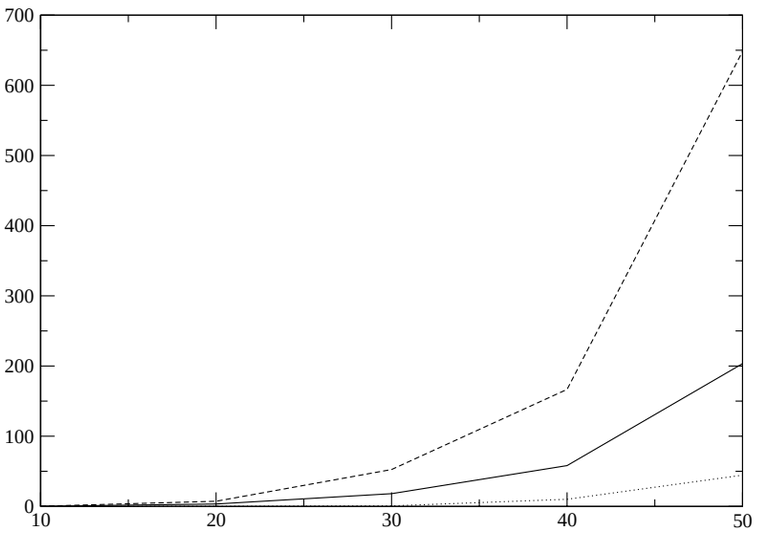

Fig. 1. Magic square graph

Figure 1 depicts how CPU times evolve w.r.t. problem size: the dotted line represents the best execution time, the dashed line the worst one and the solid line the average one.

Table 2. Comparison with Localizer

size	Localizer	Adaptive
16x16	50.82	1.14
20x20	313.2	3.4
30x30	1969	18.09
40x40	8553	58.07
50x50	23158	203.4

Constraint programming systems such as GNU-Prolog or ILOG Solver perform poorly on this benchmark and cannot solve instances greater than 10x10. We can nevertheless compare with another local search method: this benchmark has been attacked by the Localizer system with a Tabu-like meta-heuristics. Localizer [13,14,15] is a general framework and language for expressing local search strategies which are then compiled into C++ code. Table 2 compares the CPU times for both systems (in seconds). Timings for Localizer come from [15] and have been measured on a PentiumIII-800 and thus on a machine similar to ours (PentiumIII-733), but it is worth noticing however that the method used in Localizer consists in exploring at each iteration step the whole single-value exchange neighborhood (of size n^2). Our results compare very favorably with those obtained with the Localizer system, as the adaptive search is two orders of magnitude faster. Moreover its performances could certainly be improved by careful tuning of various parameters (global cost function, Tabu tenure, reset level and percentage of reset variables, ...) in order to make the method truly adaptive indeed...

4.2 God Saves the Queens

This puzzle consists in placing N queens on a NxN chessboard so that no two queens attach each other. It can be modeled by N variables (that is, one for each queen) with domains $\{1, 2, \ldots, N\}$ (that is, considering that each queen should be placed on a different row) and $3 \times N(N-1)/2$ disequation constraints stating that no pair of queens can ever be on the same column, up-diagonal or down-diagonal :

$$\forall (i,j) \in \{1,2,\ldots,N\}^2,\ s.t.\ i \neq j : Q_i \neq Q_j,\ Q_i + i \neq Q_j + j,\ Q_i - i \neq Q_j - j$$

Observe that this problem can also be encoded with three all_different global constraints.

We can define the error function for disequation as follows, in the most simple way : 0 if the constraint is satisfied and 1 if the constraint is violated. The

combination operation on variables is simply the addition, and the overall cost function is the sum of the costs of all constraints.

Table 3. N-Queens results

problem instance	time (sec) avg of 10	time (sec) best	time (sec) worst	nb iterations	nb local minima	nb swaps	nb resets
100	0.00	0.00	0.01	30	0	29	0
200	0.01	0.00	0.01	50	0	50	0
500	0.04	0.04	0.05	114	0	114	0
1000	0.14	0.13	0.15	211	0	211	0
2000	0.53	0.50	0.54	402	0	402	0
3000	1.16	1.13	1.19	592	0	592	0
4000	2.05	2.01	2.08	785	0	785	0
5000	3.24	3.19	3.28	968	0	968	0
7000	6.81	6.74	6.98	1356	0	1356	0
10000	13.96	13.81	14.38	1913	0	1913	0
20000	82.47	81.40	83.81	3796	0	3796	0
30000	220.08	218.18	221.14	5670	0	5670	0
40000	441.93	439.54	444.84	7571	0	7571	0
100000	3369.89	3356.75	3395.45	18846	0	18846	0

Table 3 details the performances of this algorithm on large instances. For a problem of size NxN the following settings are used: Tabu tenure is equal to 2 and 10 % of the variables are reset when $N/5$ variables are Tabu. The programs were executed on a PentiumIII 733 MHz with 192 Mb of memory running Linux RedHat 7.0.

Figure 2 depicts how CPU times evolve w.r.t. problem size: the dotted line represents the best execution time, the dashed line the worst one and the solid line the average one.

Surprisingly the behavior of the adaptive search is almost linear and the variance between different executions is quasi inexistent. Let us now compare with a constraint programming system (ILOG solver) and an ant colony optimization method (Ant-P solver), both timings (in seconds) are taken from [24] and divided by a factor 7 corresponding to the SPECint 95 ratio between the processors. Timings for Localizer come again from [15] and have been measured on a PentiumIII-800 and thus on a machine slighty more performant than ours. Table 4 clearly show that adaptive search is much more performant on this benchmark, which might not be very representative of real-life applications but is a not-to-be-missed CSP favorite...

4.3 All-Intervals Series

Although looking like a pure combinatorial search problem, this benchmark is in fact a well-known exercise in music composition [25]. The idea is to com-

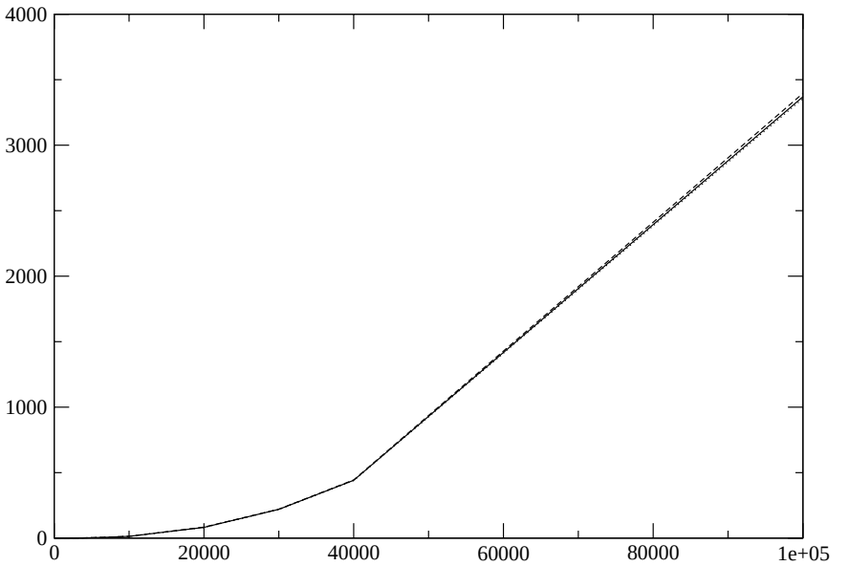

Fig. 2. N-Queens graph

pose a sequence of N notes such that all notes are different and tonal intervals between consecutive notes are also distinct. This problem can be modeled as a permutation of the N first integers such that the absolute difference between two consecutive pairs of numbers are all different.

This problem is modeled by considering N variables $\{V_1, \ldots, V_N\}$ that represent the notes, whose values will represent a permutation of $\{0, \ldots, N-1\}$. There is only one constraint to encode stating that absolute values between each pair of consecutive variables are all different. Possible moves from one configuration consist in all possible swaps between the values of two variables. As all variables appear symmetrically in this constraint there is no need to project errors on each variable (all variable errors would be equal) and we just have to compute the total cost for each configuration. One way to do this is to first compute the distance between 2 consecutive variables:

$$D_i = |V_{i+1} - V_i| \text{ for } i \in [1, n-1]$$

Then one has to define the number of occurrence of each distance value:

$$Occ_j = \Sigma_{i=1}^{N-1}(\textbf{if } D_i = j \textbf{ then } 1 \textbf{ else } 0)$$

Obviously, the all_different constraint on the distance values is satisfied iff for all $j \in [1, n-1]$, $Occ_j = 1$. It is thus interesting to focus on the values j such that $Occ_j = 0$ representing the "missing values" for the distances. We will moreover consider that it is harder to place bigger distances and thus introduce a bias in

Table 4. Comparison with ILOG Solver, Ant-P and Localizer

size	ILOG	Ant-P	Localizer	Adaptive
50	0.09	0.39		0.00
100	0.07	2.58		0.00
150	79.3	20.6		0.00
200	36.6	40.37		0.01
256	*	*	0.16	0.01
512	*	*	0.43	0.04
1024	*	*	1.39	0.15
2048	*	*	5.04	0.54
4096	*	*	18.58	2.14
8192	*	*	68.58	9.01
16384	*	*	260.8	46.29
32768	*	*	1001	270.2
65536	*	*	10096	1320

the total cost as follows:

$$cost = \Sigma_{j=1}^{n-1}(\textbf{if } Occ_j = 0 \textbf{ then } j \textbf{ else } 0)$$

Obviously a solution is found when $cost = 0$.

Table 5 details the performances of this algorithm on several instances. For a problem of size N the following settings are used: Tabu tenure is equal to $N/10$ and 10 % of the variables are reset when 1 variable is Tabu. The programs were executed on a PentiumIII 733 MHz with 192 Mb of memory running Linux RedHat 7.0.

Table 5. All-intervals series result

problem instance	time (sec) avg of 10	time (sec) best	time (sec) worst	nb iterations	nb local minima	nb swaps	nb resets
10	0.00	0.00	0.00	14	6	8	3
12	0.00	0.00	0.01	46	20	25	20
14	0.00	0.00	0.01	85	38	46	38
16	0.01	0.00	0.02	191	88	103	88
18	0.04	0.01	0.08	684	316	367	316
20	0.06	0.00	0.17	721	318	403	320
22	0.16	0.04	0.36	1519	527	992	791
24	0.70	0.10	2.42	5278	1816	3461	2724
26	3.52	0.36	9.26	21530	7335	14194	11003
28	10.61	1.38	25.00	53065	18004	35061	27006
30	63.52	9.49	174.79	268041	89518	178523	134308

Figure 3 depicts how CPU times evolves w.r.t. problem size: the dotted line represents the best execution time, the dashed line the worst one and the solid line the average one.

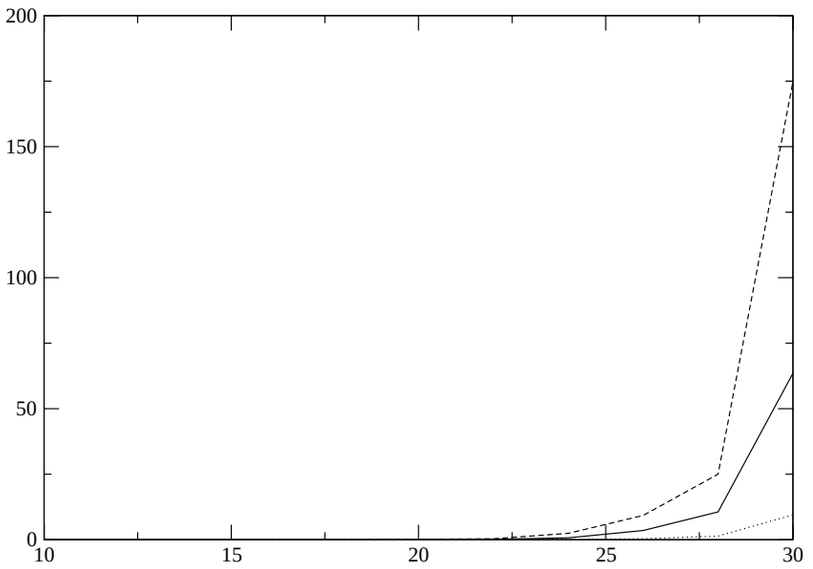

Fig. 3. All-intervals series graph

Let us now compare with a constraint programming system (ILOG solver) and an ant colony optimization method (Ant-P solver), both timings are taken from [24] and divided by a factor 7 corresponding to the SPECint 95 ratio between the processors. ILOG Solver might take advantage of global constraints to model this problem, but nevertheless perform poorly and can only find (without any backtracking) the trivial solution :

$< 0, N-1, 1, N-2, 2, N-3, \ldots >$

For instances greater than 16, no other solution can be found in reasonable time: [24] reported that the execution times where greater than a full hour of CPU time (this is depicted by a * symbol in our table).

Adaptive search is therefore more than an order of magnitude faster than Ant-Colony Optimization on this problem (see table 6, where timings are given in seconds).

4.4 Number Partitioning

This problem consists in finding a partition of numbers $\{1, \ldots, N\}$ into two groups A and B such that:

Table 6. Comparison with ILOG Solver and Ant-P

size	ILOG	Ant-P	Adaptive
14	4.18	0.07	0.00
16	126.05	0.28	0.01
18	*	0.52	0.04
20	*	1.48	0.06
22	*	2.94	0.16
24	*	9.28	0.70

- A and B have the same cardinality
- sum of numbers in A = sum of numbers in B
- sum of squares of numbers in A = sum of squares of numbers in B

This problem admits a solution iff N is a multiple of 8 and is modeled as follows. Each configuration consists in the partition of the values $V_i \in \{1, \ldots, N\}$ in two subsets of equal size. There are two constraints :

$$\Sigma_{i=1}^{n} V_i \;=\; N(N+1)/2$$

$$\Sigma_{i=1}^{n} V_i^2 \;=\; N(N+1)(2N+1)/6$$

The possible moves from one configuration consist in all possible swaps exchanging one value in the first subset with another one in the second subset. The errors on the equality constraints are computed as previously in the magic square problem. In this problem again, as in the previous all-intervals example, all variables play the same role and there is no need to project errors on variables. The total cost of a configuration can be obtained as the sum of the absolute values of all constraint errors. Obviously again, a solution is found when the total cost is equal to zero.

Table 7. Number partitioning results

problem instance	time (sec) avg of 10	time (sec) best	time (sec) worst	nb iterations	nb local minima	nb swaps	nb resets
80	0.01	0.00	0.02	169	108	61	123
120	0.02	0.01	0.04	194	118	76	177
200	0.11	0.06	0.19	383	216	166	433
512	1.13	0.07	3.26	721	348	372	1918
600	1.86	0.02	8.72	870	414	456	2484
720	4.46	0.54	21.12	1464	680	784	5101
800	6.41	0.26	12.55	1717	798	919	6385
1000	8.01	2.44	17.25	1400	630	769	6306

Table 7 details the performances of this algorithm on several instances. For a problem of size N the following settings are used: Tabu tenure is equal to 2

and 2 % of the variables are reset when one variable is Tabu. The programs were executed on a PentiumIII 733 MHz with 192 Mb of memory running Linux RedHat 7.0. Figure 4 depicts how CPU times evolve w.r.t. problem size: the dotted line represents the best execution time, the dashed line the worst one and the solid line the average one. Constraint Programming systems such as GNU Prolog cannot solve this problem for instances larger than 128.

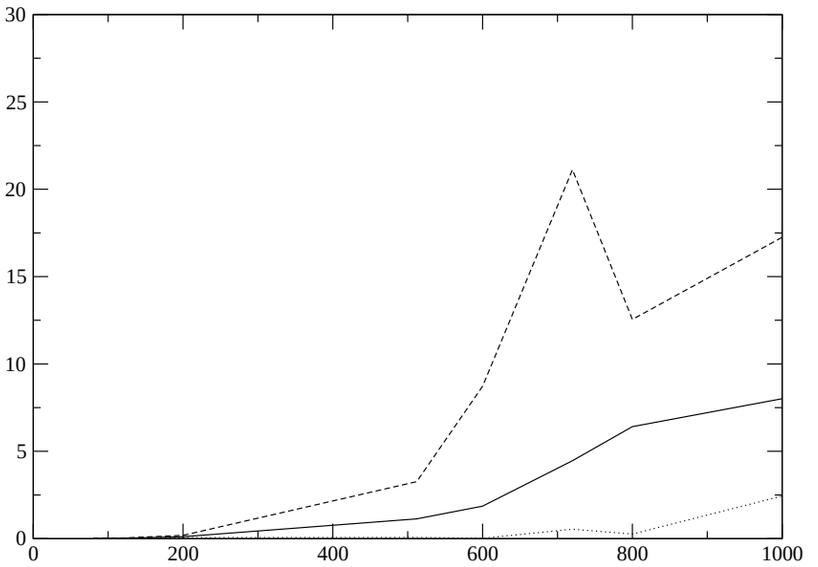

Fig. 4. Number partitioning graph

4.5 The Alpha Cipher

This problem has been posted on the newsgroup rec.puzzles a few years ago, it consists in solving a system of 20 simultaneous equations over the integers as follows. The numbers $\{1, \ldots, 26\}$ have to be assigned to the letters of the alphabet. The numbers beside each word are the total of the values assigned to the letters in the word. e.g for LYRE L,Y,R,E might equal 5,9,20 and 13 respectively or any other combination that add up to 47. The problem consists in finding the value of each letter satisfying the following equations:

BALLET = 45	GLEE = 66	POLKA = 59	SONG = 61
CELLO = 43	JAZZ = 58	QUARTET = 50	SOPRANO = 82
CONCERT = 74	LYRE = 47	SAXOPHONE = 134	THEME = 72
FLUTE = 30	OBOE = 53	SCALE = 51	VIOLIN = 100
FUGUE = 50	OPERA = 65	SOLO = 37	WALTZ = 34

This is obviously modeled by a set of 20 linear equations on 26 variables. The errors on the linear constraints are computed as previously in the magic square example. The projection on variables is the absolute value of the sum of each constraint error multiplied by the coefficient of the variable in that (linear) constraint. The total cost is, as usual, the sum of the absolute values of constraint errors.

Local search is certainly not the best way to solve such a (linear) problem. Nevertheless it could be interesting to see the performances of adaptive search on such a benchmark in order to observe the versatility of this method. Table 8 details the performances of this algorithm. The following settings are used: Tabu tenure is equal to 1 and 5 % of the variables are reset when 6 variables are Tabu. The programs were executed on a PentiumIII 733 MHz with 192 Mb of memory running Linux RedHat 7.0.

Table 8. Alpha cipher result

problem instance	time (sec) avg of 10	time (sec) best	time (sec) worst	nb iterations	nb local minima	nb swaps	nb resets
alpha-26	0.08	0.03	0.18	5419	3648	1770	751

Constraint Programming systems such as GNU Prolog can solve this problem in 0.25 seconds with standard labeling and in 0.01 seconds with the first-fail labeling heuristics. Surprisingly, adaptive search is not so bad on this example, which is clearly out of the scope of its main application domain.

5 Conclusion and Perspectives

We have presented a new heuristic method called adaptive search for solving Constraint Satisfaction Problems by local search. This method is generic, domain-independent, and uses the structure of the problem in terms of constraints and variables to guide the search. It can apply to a large class of constraints (e.g. linear and non-linear arithmetic constraints, symbolic constraints, etc) and naturally copes with over-constrained problems. Preliminary results on some classical CSP problems show very encouraging results, about one or two orders of magnitude faster than competing methods on large benchmarks. Nevertheless, further testing is obviously needed to assess these results.

It is also worth noticing that the current method does not perform any planning, as it only computes the move for the next time step out of all possible current moves. It only performs a move if it immediately improves the overall cost of the configuration, or it performs a random move to escape a local minimum. A simple extension would be to allow some limited planning capability by considering not only the immediate neighbors (i.e. nodes at distance 1) but all configurations on paths up to some predefined distance (e.g. all nodes within at

distance less than or equal to some k), and then choose to move to the neighbor in the direction of the most promising node, in the spirit of variable-depth search [11] or limited discrepancy search [9]. We plan to include such an extension in our model and evaluate its impact. Further work is needed to assess the method, and we plan to develop a more complete performance evaluation, in particular concerning the robustness of the method, and to better investigate the influence of stochastic aspects and parameter tuning of the method. Future work will include the development of dynamic, self-tuning algorithms.

References

1. E. Aarts and J. Lenstra (Eds). *Local Search in Combinatorial Optimization*. Wiley, 1997.
2. A. Borning, B. Freeman-Benson and M. Wilson. Constraint Hierarchies. *Lisp and Symbolic Computation*, vol. 5 no. 3, 1992, pp 223-270.
3. P. Codognet and D. Diaz. Compiling Constraint in `clp(FD)`. *Journal of Logic Programming*, Vol. 27, No. 3, June 1996.
4. A. Davenport, E. Tsang, Z. Kangmin and C. Wang. GENET : a connectionist architecture for solving constraint satisfaction problems by iterative improvement. In *proc. AAAI 94*, AAAI Press, 1994.
5. D. Diaz and P. Codognet. The implementation of GNU Prolog. In *proc. SAC'OO, 15th ACM Symposium on Applied Computing*. Como, Italy, ACM Press 2000.
6. F. Glover and M. Laguna. *Tabu Search*, Kluwer Academic Publishers, 1997.
7. P. Galinier and J-K. Hao. A General Approach for Constraint Solving by Local Search. *draft*, 2001.
8. J-K. Hao, P. Galinier and M. Habib. Metaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes. *Revue d'Intelligence Artificielle*, vol.2 no. 13, 1999, pp 283-324.
9. W. Harvey and M. Ginsberg. Limited Discrepancy Search. In *proc. IJCAI'95, 14th International Joint Conference on Artificial Intelligence*, Montreal, Canada, 1995.
10. S. Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, vol. 44 (1965), pp 2245-2269.
11. S. Lin and B. Kerninghan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, vol. 21 (1973), pp 498-516.
12. Z. Michalewicz and D. Fogel. *How to solve it: Modern Heuristics*, Springer Verlag 2000.
13. L. Michel and P. Van Hentenryck. Localizer : a modeling language for local search. In *proc. CP'97, 3rd International Conference on Principles and Practice of Constraint Programming*, Linz, Austria, Springer Verlag 1997.
14. L. Michel and P. Van Hentenryck. Localizer. *Constraints*, vol. 5 no. 1&2, 2000.
15. L. Michel and P. Van Hentenryck. Localizer++ : an open library for local search. Research Report, Brown University 2001.
16. S. Minton, M. Johnston, A. Philips and P. Laird. Minimizing conflicts : a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, vol. 58, 1992, pp 161-205.
17. K. Nonobe and T. Ibaraki. A tabu search approach to the constraint satisfaction problem as a general problem solver. *European Journal of Operational Research*, vol. 106, 1998, pp 599-623.

18. G. Pesant and M. Gendreau. A Constraint Programming Framework for Local Search Methods. *Journal of Heuristics*, vol. 5 no. 3, 1999, pp 255-279.
19. J-F. Puget. A C++ implementation of CLP. In *proc. SPICIS'94*, Singapore, 1994.
20. V. J. Rayward-Smith, I. H. Osman, C. R. Reeves, G. D. Smith. *Modern Heuristic Search Methods*. Wiley, 1996.
21. V. Saraswat, P. Van Hentenryck et al. Constraint Programming, *ACM Computing Surveys*, vol. 28 no. 4, December 1996.
22. B. Selman, H. Levesque and D. Mitchell. A new method for solving hard satisfiability problems. In *proc. AAAI'92*, AAAI Press 1992.
23. B. Selman, H. Kautz and B. Cohen. Noise strategies for improving local search. In *proc. AAAI'94*, AAAI Press 1994.
24. C. Solnon. Solving permutation problems by ant colony optimization. In *proc. ECAI'2000*, Berlin, Germany, Wiley, 2000.
25. C. Truchet, C. Agon and G. Assayag. Recherche Adaptative et Contraintes Musicales. In *proc. JFPLC2001, Journées Francophones de Programmation Logique et Programmation par Contraintes*, P. Codognet (Ed.), Hermes, 2001.
26. J. P. Walser. *Integer Optimization by Local Search : A Domain-Independent Approach*, LNAI 1637, Springer Verlag 1999.

An Evolutionary Algorithm for the Sequence Coordination in Furniture Production

Carlo Meloni

Dipartimento di Informatica e Automazione
Università di Roma Tre
Via della Vasca Navale, 79 I-00146 Roma, Italy
meloni@dia.uniroma3.it

Abstract. In the material flow of a plant, parts are grouped in batches, each having as attributes the *shape* and the *color*. In both departments, a *changeover* occurs when the attribute of a new part changes. The problem consists in finding a common sequence of batches optimizing an overall utility index. A metaheuristic approach is presented which allows to solve a set of real-life instances and performs satisfactorily on a large sample of experimental data.

Keywords: Evolutionary algorithms, sequencing, manufacturing systems.

1 Introduction

This paper deals with a problem of a serial production system concerning the way in which two consecutive stages should organize their internal work, taking into account the requirements of the other stage.

The models considered in this paper are related to an application in a real industrial context, namely a plant in which a large number of different panels of wood are cut, painted and assembled to build furniture parts. In different departments of the plant, items are grouped according to different attributes. In the cutting department, parts are grouped according to their shape and material, in the subsequent painting department parts are grouped according to their color, and finally the assembly of the furniture is organized on the basis of the different products. This paper focus on the coordination issues between the first two stages, and therefore we are only interested in two attributes, called for simplicity *shape* and *color*.

In both departments, a *changeover* occurs when the attribute of a new part changes. If a part must be cut having a different shape from the previous one, cutting machinery must be reconfigured. Similarly, the painting station, when a new color is used, must be cleaned in order to eliminate the residuals of the previous color. In both cases costs are incurred in terms of time and manpower.

Limited interstage buffering between the two stages forces the sequences in which the items are processed in the two stages to be coordinated. In this paper, we analyze the case in which the two departments follow the same common

K. Steinhöfel (Ed.): SAGA 2001, LNCS 2264, pp. 91–105, 2001.

sequence. Each department would like to minimize its own total changeover cost, but the objectives of the two departments may be conflicting.

Other issues may exist in real life situations. For example, changeovers may have different costs. In the painting department, setup operation takes less time to switch from a lighter to a darker color than viceversa. In some cases, there may even be precedence constraints that limit the set of feasible sequences in the two departments. On the other hand, there are some fixed activities involved in a changeover which are very much the same, regardless of the particular changeover, and the contribution of these fixed activities to the setup cost may be extremely relevant. Hence, the total number of changeovers is usually a meaningful index of performance.

In this paper, the problem in its basic version is addressed, i.e., when all the changeovers in each department and across departments have the same cost and no resequencing is possible between the two stages. Hence, the focus is on the *number* of changeovers, in order to establish a *single* common sequence of items which both stages will follow. Even so, the problem turns out to be hard and calls for a heuristic solution approach [1].

Each item to be produced is characterized by its own shape and color. All the items having the same shape and color form a single *batch*. In fact, there is no convenience, in splitting such batches. In the first (second) department, a changeover is paid when the new batch has a different shape (color) from the previous one. Otherwise, no changeover is incurred. Note that since we want to minimize the number of changeovers, the actual cardinality of each batch is of no interest at all.

Each given sequence of the batches results in a number of changeovers to be paid by each department. In particular two different objectives are considered, namely: i) minimize the total number of changeovers in the two departments (MTC); ii) minimize the maximum number of changeovers paid by either department (MMC).

While the former objective corresponds to the maximization of overall utility, the latter may better capture the need to balance the changeover costs between the two departments. The interest may be also in the combination of different objectives introducing a multicriteria optimization problem.

2 Problem Formulation

The problem we consider can be formulated as follows. Let A be a set of batches to be produced. The batches must be processed by two departments of the plant, called D_S and D_C, in the same order. Each batch is characterized by two attributes, say *shape* and *color*. Let S and C denote the sets of all possible shapes and colors respectively. We denote the shapes as s_i, $i = 1, \ldots, |S|$ and the colors as c_j, $j = 1, \ldots, |C|$. Each batch is therefore defined by a pair (s_i,c_j). If batch (s_i,c_j) is processed immediately after batch (s_h,c_k), a changeover is paid in department D_S if $s_h \neq s_i$, and a changeover is paid in department D_C if $c_k \neq c_j$. We can represent the input as a bipartite graph, $B = (S, C, A)$, in

which nodes in S correspond to shapes, nodes in C to colors and each edge of A corresponds to a batch to be produced. The problem is to sequence the batches in a profitable way from the viewpoint of the number of changeovers. This means that we must find an ordering σ of the edges of B. If two consecutive edges (i, j) and (h, k) in σ have no nodes in common, this means that both departments have to pay one changeover when switching from batch (i, j) to batch (h, k). We refer to this as a *global changeover*. On the other hand, if $i = h$ ($j = k$), only department D_C (D_S) pays a changeover. This is called *local changeover*. For a given sequence σ, we can therefore easily compute the number of change overs incurred by each department, call them $N_S(\sigma)$ and $N_C(\sigma)$ respectively. In fact, let δ_{ih} be equal to 1 if $i \neq h$ and 0 otherwise, and let $s(\sigma(q))$ denote the shape of the q-th batch in the sequence σ, and let $c(\sigma(q))$ denote its color.

Let $N_S(\sigma) = \sum_{q=1}^{|A|-1} \delta_{s(\sigma(q)),s(\sigma(q+1))}$ and $N_C(\sigma) = \sum_{q=1}^{|A|-1} \delta_{c(\sigma(q)),c(\sigma(q+1))}$, then the problems addressed in this paper can be formulated as follows:

- **MTC:** Given a bipartite graph B=(S,C,A), find a sequencing σ of the edges such that $N_S(\sigma) + N_C(\sigma)$ is minimized.
- **MMC:** Given a bipartite graph B=(S,C,A), find a sequencing σ of the edges such that $\max\{N_S(\sigma), N_C(\sigma)\}$ is minimized.

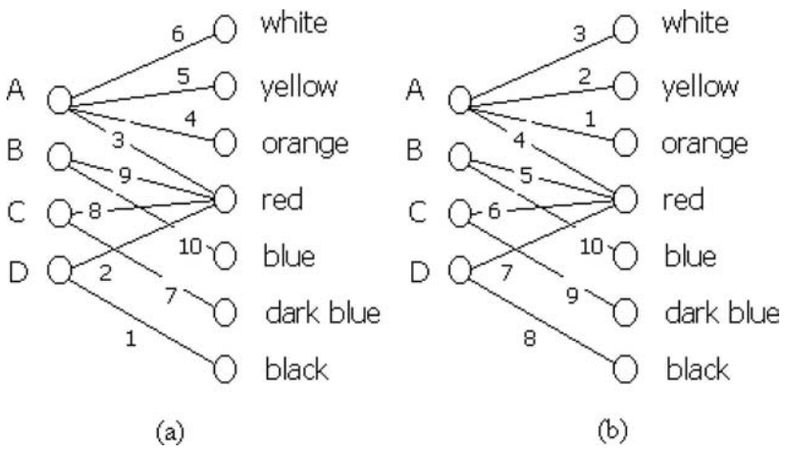

Fig. 1. MTC and MMC: an example.

An example is reported in Figure 1. There are 10 batches to be produced, requiring 4 shapes and 7 colors. Figures 1(a) and (b) show the optimal sequences for MTC and MMC respectively (the edges are numbered according to their position in the sequence). It is easy to see that the two objectives are not equivalent.

The MTC problem consists in finding a sequence σ of the edges of T such that the total number of changeovers is minimum. Clearly, MTC is equivalent to minimizing the number of global changeovers. An *ideal* solution σ_I is one

in which all changeovers are local, i.e., batches are sequenced so that every two consecutive batches have either the same shape or the same color. An ideal solution may not exist, but if it does, it can be characterized in graph theoretical terms.

An ideal sequence on $B = (S, C, A)$ exists if and only if there is a dominating trail on B (i.e. a trail such that every edge of the graph has at least one endpoint on it). Note that a dominating trail completely specifies an ideal schedule, since in fact dominated edges can always be scheduled between two consecutive edges of the trail with no global changeovers. However, if an ideal sequence does not exist, the number of global changeovers must still be minimized. We can therefore express MTC as the problem of dominating the edges of B with the minimum number of (edge-disjoint) trails, i.e., finding a minimum cardinality dominating trail set (MDTS)[3]. The concept of dominating trail has received considerable attention in the graph theoretic literature and MDTS has been shown to be polynomially solvable when B is a tree or a cactus [2,7]; while, MDTS, MTC and MMC are shown to be NP-hard even for bipartite graphs [1,4].

Some formulations for MTC and MMC can be considered. Notice that, given the bipartite graph $B = (S, C, A)$, MTC can always be solved finding the hamiltonian path of minimum weight on a complete graph $K_{|A|}$ (with one node for each edge of B), in which the weight associated with each edge (x, y) of $K_{|A|}$ is 0 if $x \in A$ and $y \in A$ are adjacent in B and 1 otherwise. Clearly, there is a one-to-one correspondence between hamiltonian paths in $K_{|A|}$ and batch sequences, the weight of a hamiltonian path being equal to the number of global changeovers in the corresponding sequence.

However, the size of graph $K_{|A|}$ can easily be too large to find the minimum weight hamiltonian path within the time constraints practitioners are usually faced with. Let the *density* of B be defined as $|A|/|S||C|$. A problem with 50 shapes and 50 colors and density 0.4 results in a graph $K_{|A|}$ with 1000 nodes. Moreover, it is well known that instances of the hamiltonian path problem (as well as the traveling salesman problem) with 0-1 costs are particularly hard.

Finally, in real industrial applications, it may be important to have a method which is time–limited, in particular if the sequencing problem has dynamic aspects. In fact, in the case of large instances, by the time the problem is *solved*, it may have changed (e.g. new orders may have arrived), so a method that takes too long time may be inappropriate whatever the quality of the solutions it produces for a static problem. For these reasons we do not consider ILP formulations based on TSP further in this paper.

An alternative (0-1) integer programming formulation of MTC and MMC may be based on the approach suggested by Crama et al. [6] for the tooling and loading problem of a FMS. Note that the size of these ILP problems are smaller then those obtained in the hamiltonian path or TSP formulations. However, the linear programming relaxation provides lower bounds which are extremely weak. Hence, improvements are needed in the formulations in order to make it effective. These improvements may consist in some vaild constraints to add to the formulations [15] and may be the object for further research.

3 Heuristic Approaches for MDTS and MTC

In this section two heuristic approaches are presented for MDTS and MTC problems. The first heuristic (DOMWALKS) performs better on graphs with high density, while the second heuristic (DOMTREE) obtains good results on sparse and tree-like instances.

3.1 DOMWALKS Heuristic

Agnetis et al. [2] proposed a heuristic (called DOMWALKS) with the aim to build a dominating trail set Σ, trying to keep its cardinality as low as possible. The approach is based on the following considerations. If graph G happens to have a eulerian walk, the solution to MTC is straightforward, since a eulerian walk is a dominating walk, and hence an ideal solution would exist. If a eulerian walk does not exist, the idea is to remove from G some edges so that, in the resulting graph $\bar{G}$, the degree of each node is even. If $\bar{G}$ is connected, it has a eulerian cycle, and hence a eulerian walk. A eulerian walk W_e in $\bar{G}$ is a dominating walk in G, since each removed edge is adjacent to some edge of W_e. If $\bar{G}$ is not connected, each component either has a eulerian cycle or consists of a single node. All the edges of each component can be dominated by an eulerian walk, but a new trail may need when moving from one component to another.

One key feature of this approach is the edge removal phase. Given the graph G, an *odd-vertex pairing* (OVP) Π is a set of edge-disjoint (but possibly not node-disjoint) paths having nodes with odd degree in G as endpoints, such that each node with odd degree in G is the endpoint of exactly one path, and no nodes of even degree are endpoints of any path. If we remove Π from G, we are left with a graph $\bar{G}$ in which all vertices have even degree. Hence, if we find an OVP with few edges, chances are that $\bar{G}$ is still connected. We therefore seek an OVP with a minimum total number of edges. Such a pairing can be found in polynomial time by solving a matching problem [2], as follows. Given the graph G, let $N(G)$ be the set of nodes of G, and let $N_{ODD}(G)$ be the set of nodes having odd degree in G. Consider a complete graph $K_{|N_{ODD}(G)|}$ with $|N_{ODD}(G)|$ nodes, in which each edge (i, j) is weighted by the length (number of edges) of the shortest path from i to j in G. A perfect matching of minimum weight in $K_{|N_{ODD}(G)|}$ corresponds to an OVP with a minimum number of edges in G. Note that the optimality of such OVP implies that the paths are all edge-disjoint.

The algorithm begins by finding an OVP Π. Once the edges of Π are removed, we are left with the graph $\bar{G}$. The algorithm then starts from an empty trail set, and iteratively adds trails one by one until all the edges of G are dominated. We call Σ_C the current set of trails formed by the algorithm. The algorithm builds each trail by finding a dominating trail for one or more connected components of $\bar{G}$. In order to limit the number of used trails as much as possible, the algorithm tries to use the edges of the OVP Π to move from the current component to another one. Of course, no edge of Π can be used more than once throughout the algorithm, so it may be the case that no path of Π is anymore available to move from the current component to another. In this case a trail is

completed and added to the current trail set Σ_C, the algorithm jumps to a new component, and a new trail is started. This algorithmic scheme allows to build good heuristics and yields a simple bound to the number of global changeovers in an optimal solution, therefore: $|MMC| \leq |MTC| \leq \frac{|N_{ODD}(G)|}{2} + |E(G)|$.

When the density of the graph is higher than 25%, DOMWALKS often finds a dominating walk, if it exists, and it requires a low computational time even for large instances. For a given value of $|V(G)|$, the sparse instances are those for which DOMWALKS gives the largest number of global changeovers. This is not surprising, since as the number of edges increases, it is easier to establish a dominating walk.

3.2 DOMTREE Heuristic

An $O(|V(G)|)$ algorithm (known in literature as DOMTREE) has been proposed for $MDTS$ on trees [2]. Therefore, a heuristic for a more general graph G can be obtained applying DOMTREE to a spanning tree of G. As spanning trees can be efficiently individuated [11], the overall algorithm for $MDTS$ based on DOMTREE results efficient too.

In particular, DOMTREE is based on the relationship between $MDTS$ and the Hamiltonian Completion Number problem (HCN). The Hamiltonian Completion Number of a graph $G = (V, E)$ is the minimum number of edges which need to be added to G to make it hamiltonian, and will be denoted as $HCN(G)$. Given a graph G, the hamiltonian completion number of its line graph $L(G)$, i.e. $HCN(L(G))$ is called Edge Hamiltonian Completion number of G. The line graph $L(G)$ of a graph G has a hamiltonian cycle if and only if G has a dominating trail. Moreover, there is a relationship between the cardinality of a minimum dominating trail set of G and $HCN(L(G))$. Given any connected graph $G = (V, E)$, the cardinality of a minimum dominating trail set $|MDTS|$ on G is k if and only if $HCN(L(G)) = k - 1$. DOMTREE algorithm was proposed for $HCN(L(G))$ when G is a tree, and it works also for $MDTS$ on trees [2]. This approach yields good experimental results on sparse and almost *tree-like* instances.

4 An Evolutionary Algorithm for MTC and MMC

Evolutionary algorithms (EA) are optimization techniques inspired from natural processes [8]. They handle a population of individuals that evolve with the help of information exchange procedures. Each individual may also evolve independently. This behavior alternates periods of cooperation with periods of self-adaptation. The population of an EA iteratively evolves according to some specific rules. The proposed algorithmic approach has an evolution process based on a genetic algorithm. The concepts of genetic algorithms (GAs) have been applied successfully to problems involving pattern recognition, classification and other problems in the domain of Artificial Intelligence. Some literature in the recent years reports about applications to large combinatorial [13,5] and industrial scheduling problems [14,10]. This section attempts to apply EAs to two

such problems, namely MTC and MMC. GAs, even in their basic version, have many possible choices of control parameters, but the emphasis here is on showing the feasibility of using GAs in an evolutionary strategy for such problems by producing a working algorithm, rather than on a search for the optimal parameter settings. This approach provides a way to deal with these problems, while the performance may depend on strategies adopted and particular settings of algorithm's parameters.

4.1 The Genetic Evolutionary Mechanism

A part of the research community is involved in the design and development of heuristics for NP–hard problems. One such problem, known to be NP–hard, is the permutation flowshop sequencing problem in which n jobs have to be processed (in the same order) on m machines. The object is to find the permutation of jobs which will minimize the makespan, i.e. the time at which the last job is completed on machine m. For this problem Reeves [14] proposes a GA solution scheme that perform relatively well on large instances. In this paper, starting from the Reeves

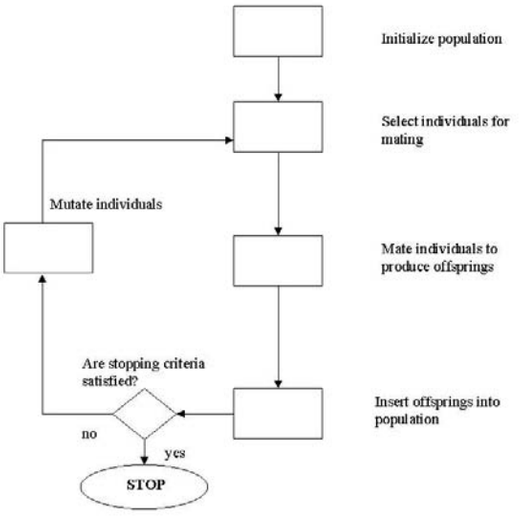

Fig. 2. The scheme of the basic genetic algorithm.

scheme, a GA approach dedicated to the permutation sequencing problems MTC and MMC is developed. In particular, these problems may be considered as a special case of permutation scheduling on a single processor in which the edges of the graph are the jobs to be processed, and the objective is to minimize the number of changeovers. Different genetic *operators* can be identified, the most commonly used ones being *crossover* (an exchange of sections of the parents' chromosomes), and *mutation* (a random modification of the chromosome). In

the context of finding the optimal solution to a large combinatorial problem, a GA works by maintaining a population of M solutions which are potential *parents*, whose *fitness values* have been calculated. In Holland's original GA [9], one parent is selected on a fitness basis (i.e. the better is the fitness value, the higher is the probability of it being chosen), while if crossover is to be applied, the other parent is chosen randomly. They are then *mated* producing offsprings to replace a part of the existing population chosen by means of a probabilistic rule. The *reproduction* is iteratively repeated. Various versions of this procedure are been proposed in literature and different strategies are been adopted.

5 The Implementation

Other than methodological or strategic decisions, there are parametric choices to do: different crossover and mutation probabilities, different population sizes an so on. It is also quite possible that the way the initial population is chosen will have a significant impact on the results. In Figure 2 are depicted the traditional scheme of genetic algorithms. In the this section we focus on GA application in the evolutionary scheme for the specific problems introduced in Sec. 1 and 2.

- Chromosomes and crossover operation - In order to apply any GA to a sequencing problem, there is an obvious pratical difficulty to consider. In most *traditional* GAs, the chromosomal representation is by means of a string of 0s and 1s, and the result of a genetic operator is still a valid chromosome. This is not the case if one uses a permutation to represent the solution of a problem, which is the natural representation for a sequencing problem. For example, if the parents are as shown below,

Parent 1: $(2,1,3,4,5,6,7)$	Offspring 1: $(2,1,3,2,5,7,6)$
Parent 2: $(4,3,1,2,5,7,6)$	Offspring 2: $(4,3,1,4,5,6,7)$

it is clear that choosing the crossover point between the third and the fourth symbol, the offsprings are *illegitimate* because symbols do not appear once in the sequence. Different representation are possible, but it seemed easier to modify the idea of crossover and keep the permutation representation. The crossover (C) chose for the proposed algorithm consists of randomly selecting one point X along the parents' chromosomes, taking the pre–X section of the first parent, and filling up the chromosome by taking in order each *legitimate* symbol from the second parent. The application of this rule to previous example yields:

Parent 1: $(2,1,3,4,5,6,7)$	Offspring 1: $(2,1,3,4,5,7,6)$
Parent 2: $(4,3,1,2,5,7,6)$	Offspring 2: $(4,3,1,2,5,6,7)$

The idea behind C is that it preserves the absolute positions of the symbols taken from the first parent, and the relative positions of those from the second. This would provide enough scope for modification of the chromosome without excessively disrupting it. Preliminary experiments showed that this crossover rule tended to converge rather rapidly to a population of almost identical chromosomes (premature convergence), so a mutation operation is also required. In

other words the crossover C emphasizes *exploitation* instead of *exploration*; using a mutation provides a way to restore the balance. In the proposed evolutionary algorithm, a chromosome represents directly a complete sequence of of the edges of the graph G related to an instance of the problem (as reported in Figure 3). In particular, each *gene* will represent a single edge of the graph and it will have the same label. Also, the gene contains information about vertices and weight of the represented edge. The fitness of each chromosome is straightforward obtained scanning the sequence and calculating the number of changeovers. These characteristics make this representation preferable with respect to those based on a Random Key Encoding (RKE). Moreover, experiments underline three main drawbacks of RKE: a) the sorting process to obtain the sequence is time consuming; b) the encoding does not preserve the adjacency of edges in a given sequence; c) the crossover operation may lose more important information.

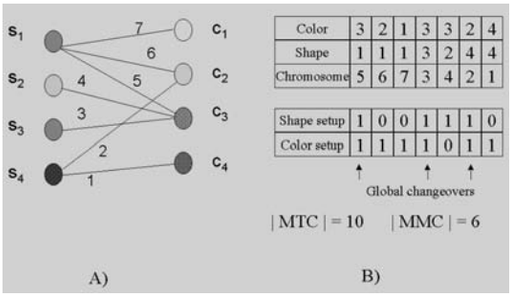

Fig. 3. The representation: A) the instance; B) a chromosome and its fitness.

- *Reproduction mechanism* - The selection of parents for the reproduction process (by means of the crossover operator), is a simple ranking mechanism for the first parent, while the second parent is selected randomly over the current population. At this aim, the population is ordered on the basis of the fitness of its elements. However, the ranking selection mechanism involves only K elements of the population with a better fitness than the current average fitness. Such a mechanism works in accordance with the probability distribution $p([i]) = \frac{2i}{K(K+1)}$, where $[i]$ is the ith chromosome in ascending order of fitness (for example, descending order of MTC or MMC), and K is the number of elements involved in this selection. This implies that the median value has a probability of $\frac{1}{K}$ of being selected, while the Kth (the fittest element) has a probability of $\frac{2}{(K+1)}$, about twice that of the median. The ranking mechanism is also used for choosing elements to be eliminated at each iteration, i.e. substituted by a offspring. Since we already have a ranking of the chromosomes, it is easy to choose from elements which have a bad fitness.

- *Mutation operation* - A mutation operator was used to reduce the rate of convergence. In traditional GAs, mutation of a chromosome is performed by

flipping each of its symbols from 0 to 1 (or vice versa) with a small probability. Using a permutation representation, mutation needs to be defined differently. Three types of mutation are considered: a) *exchange* mutation, was a simple exchange of two symbols in a permutation, chosen at random; b) *shift* mutation, i.e. a shift of one symbol (again, chosen randomly) a random number of places to the right or to the left; c) *reversal* mutation consisting in change a subsequence of the selected chromosome with its reverse. In the latter case, choosing the subsequence between two global changeovers, the fitness value of the chromosome will not be affected by the mutation operator. The premature convergence problem is one that has been noted by several workers in applying GAs, and various proposals have been made to prevent it. The method applyed here is to use an adaptive mutation rate p_m. At the start of algorithm an high probability of mutation is allowed; this probability is slowly decreased as long as a reasonable diverse population exists. If the diversity becomes too small, the original high value is restored. Several measures of diversity may be adopted, our choice is to reset the mutation rate if the ratio $\frac{f_{min}}{f_{avg}}$ is below a value D_{th}, where f is the fitness value of each chromosome (that is directly the objective of the problem to solve).

- Initial population - Another consideration deals with the selection of the initial population. The basic GA scheme assumes that the initial population is chosen completely at random. It may be profitable trying the effect of *seeding* the initial population with a good solution generated by a constructive heuristic. In the case of the problems under study, MTC my be tackled with heuristics based on DOMWALKS [1] or DOMTREE [2], while for MMC we have not a specific heuristic, and we may try with our best MTC solution as upper bound for this problem. This suggests an algorithmic scheme consisting of two different subsequent stages, one dedicated to MTC and another to MMC. Experimental results show that the seeding operation allows to quite reduce the computational time (see also Figure 4).

- Stopping criteria - The basic GA scheme has not a natural stopping point. In order to avoid too long computation time it is useful to introduce some stopping criteria. The simplest method consist to limit the number of algorithm iterations or the total computational time. This method may be refined considering also the evolution of the algorithm and starting iteration/time counters only when iteration does not improve the population fitness (e.g. the best or average values). The stopping criteria introduced in the proposed evolutionary scheme takes into account some theoretical lower and upper bounds on the objective function [12]. The first avoids to continue in the search when an optimal sequence is already found, while the second may force the algorithm to continue the search in order to improve the current solution. Both bounds allow to evaluate the quality of the current individuals in the population.

- Evolutionary strategy - Some parts in the algorithm have been modified in order to improve its performances. The changes are valuable especially on large instances. A particular evolution scheme is implemented in which two GAs evolve in parallel (starting with different initial population) in the first phase.

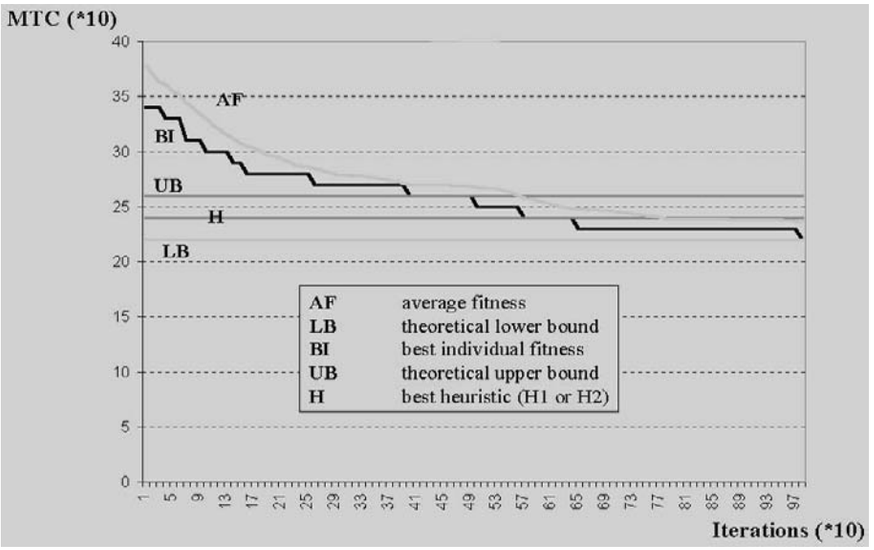

Fig. 4. Typical behaviour of the EA.

This phase ends when the average fitness of each population converges between two theoretical bounds. In the second phase the populations are merged and a new GA may start. This scheme may be useful also in the case of the bi-criteria problem to find solution when both objectives (MMC and MTC) are assumed to be minimized. In addition to the set of randomly generated individuals, two seeding heuristics (i.e. DOMWALKS and DOMTREE) are used. The resulting scheme of the proposed EA is reported in Figure 5. In Figure 4 are described the typical behaviour of the basic EA (without seeding): it appears clear as the seeding operation (i.e. using heuristics DOMWALKS or DOMTREE) may yields a sensible time saving. The characteristics of the *reverse* mutation suggest us a simple way to improve also the crossover operator: when the chomosomes to mate are almost identical (i.e., when the number of identical elements is greater than a fixed value S_{th}), one of them is reversed before the crossing over. After a preliminary phase for testing and tuning the algorithm, a particular setting of parameters is adopted and another improvement is introduced with respect to the basic GA scheme.

To improve the convergence, a local search (LS) procedure is incorporated into the algorithm. At any generation, LS is applied on the elitary chromosomes (i.e., chromosomes with good values of fitness in each population) except for which LS has been already applied in the previous generations. In order to eliminate a global changeover, the LS procedure scans a chromosome to find two subsequences σ_1 and σ_2 characterized as follows: a) both σ_1 and σ_2 are exactly delimited by two global changeovers; b) a starting (ending) subsequence α_i (β_i) exists in σ_i in which one of the two nodes of the edges does not change; c) subse-

quences α_1, β_1, α_2 and β_2 contain edges that allow to link up the subsequences σ_1 and σ_2. After the generation of offsprings, the population is updated and LS is applied on new elitary elements. When the best (current) solution changes in one population, it is introduced in the other population as soon as it is updated. It takes advantage of good genetic information and propagate them in each population.

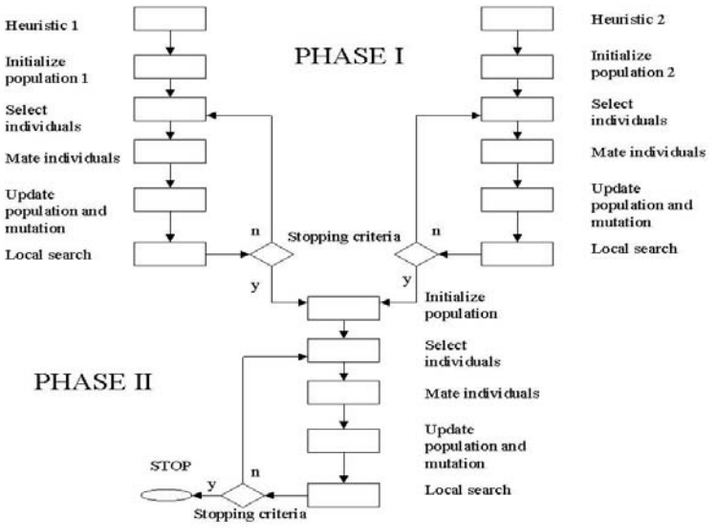

Fig. 5. The scheme of the evolutionary algorithm.

6 Experimental Results

A set of real instances as well as a wide set of artificial instances are considered in order to study the performance of the proposed approach. The real cases are characterized by several *cutting classes*(from 140 to 235) and *color classes* (from 20 to 30) with a density from 18 to 60%. The production of the plant follows an assembly-to-order strategy. Every week a new production plan is computed on the basis of the actual orders. The tests refer to 32 instances related to 32 different production weeks.

The EA was coded in C language and was tested on a PC Pentium II, 200 MHz with 64MB ram.

In all 32 instances the optimal solution (MTC) has been found by the EA. It takes always less than 150 seconds with an average computing time of 93 seconds. Using the seeding operation, these computing times decrease by a factor of about 5. Moreover, the use of the heuristics allows to reduce the *random* behaviour of

the EA (Figures 6 and 7 report the best/worst results on 5 different runs). In Figure 8 are reported the best/worst performance with respect to MMC objective of the EA using the seeding operation. In this case the effects are less appreciable as both the heuristics pursuing the MTC minimization. The results on artificial instances provide a way to carry out the tuning of the parameters of the algorithm. Moreover, these experiments point out the complementary behaviour of the EA and heuristics (DOMWALKS and DOMTREE). In fact, EA performs well when the graph has low density or a small size in terms of number of edges. DOMWALKS gives good results when the graph's density is high and may fail when the graph is particularly sparse; while DOMTREE performs well on cases with low density (or tree-like graphs) independently of the number of edges.

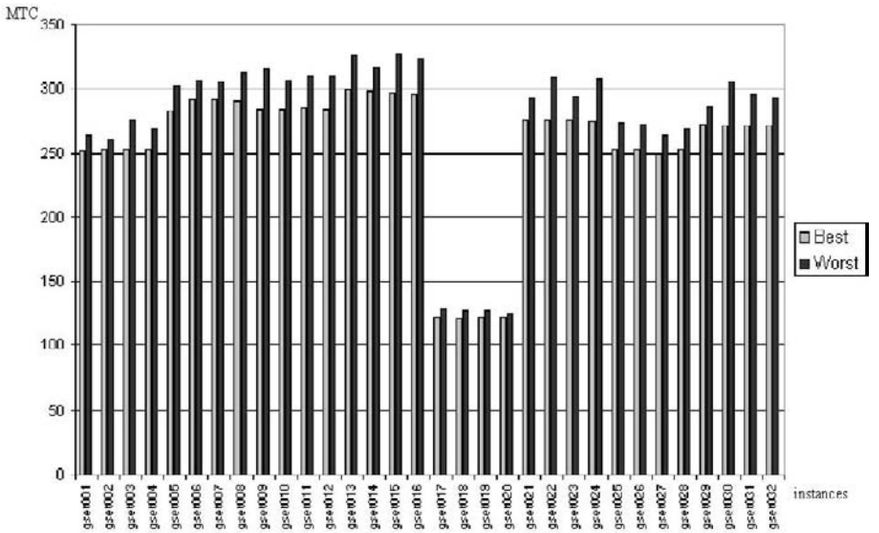

Fig. 6. Experimental results 1: EA with MTC objective and random initialization.

7 Conclusions

Experiments are made on a wide set of instances also from real industrial cases. The results allow us to point out the effect of seeding and local search operations with respect to both the quality of the solutions and the computation time.

The proposed EA appears to be a fast and powerful algorithm for the considered problems. The results obtained suggest some way to adapt the evolutionary scheme in order to tackle other issues such as: i) the case in which setups have different costs as those occurring when, for instance, it takes less time to switch

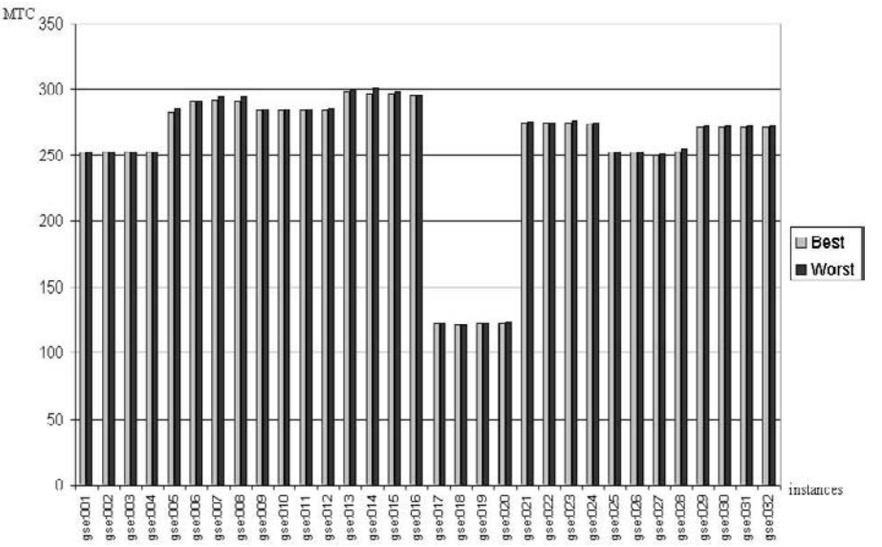

Fig. 7. Experimental results 2: EA with MTC objective and seeding operation.

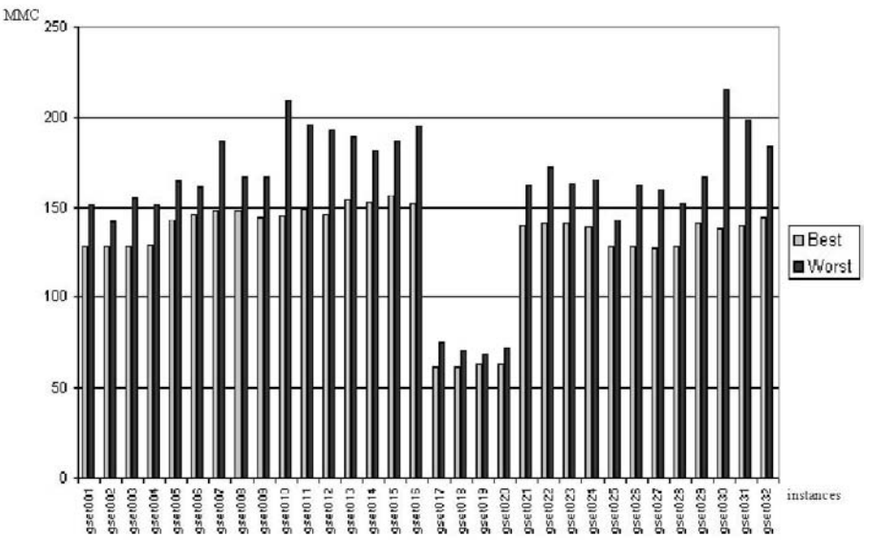

Fig. 8. Experimental results 3: EA with MMC objective and seeding operation.

from a lighter to a darker color than viceversa; ii) the multicriteria problems in which the objective account for both the total and the maximum number of setups; iii) the introduction of technological precedence constraints either on shapes or colors.

Acknowledgements

The author gratefully acknowledge the helpful comments and suggestions of the anonymous referees.

References

1. Agnetis, A., Detti, P., Meloni, C., Pacciarelli, D., 1999, Set-up coordination between two stages of a supply chain. Tech.Rep. n. 32-99, Dipartimento di Informatica e Sistemistica, Università La Sapienza, Roma.
2. Agnetis A., Detti P., Meloni C., Pacciarelli D., 2001, A linear algorithm for the hamiltonian completion number of the line graph of a tree. *Information Processing Letters* 79, 17-24.
3. Agnetis A., Detti P., Meloni C., Pacciarelli D., 2001, The minimum cardinality dominating trail set problem (MDTS). *Proceedings of the AIRO 2001 conference*, CUEC, Cagliari.
4. Bertossi, A.A., 1981, The edge hamiltonian problem is NP-complete, *Information Processing Letters*, (13), 157–159.
5. Chatterjee, S., Carrera, C., Lynch, L.A., 1996, Genetic algorithms and traveling salesman problems, *European Journal of Operational Research* (93), 490-510.
6. Crama, Y., Oerlemans, A.G., Spieksma, F.C.R., 1996, Production planning in automated manufacturing, Springer, Berlin.
7. Detti, P., Meloni, C., 2001, A linear algorithm for the Hamiltonian completion number of the line graph of a cactus. In: H. Broersma, U. Faigle, J. Hurink and S. Pickl (Eds.) Electronical Notes in Discrete Mathematics vol 8 (2001), Elsevier Science Publishers.
8. Hertz, A., Kobler, D., 2000, A framework for the description of evolutionary algorithms, *European Journal of Operational Research* (126), 1-12.
9. Holland, J.H., 1976, Adaption in natural and artificial systems, University of Michigan Press, Ann Arbor MI.
10. Jawahar, N., Aravindan, P., Ponnambalam, S.G., Aravind Karthikeyan, A., 1998, A genetic algorithm-based scheduler for setup-constrained FMC, *Computers in Industry* (35), 291-310.
11. Korte, B., Vygen, J., 2000, Combinatorial Optimization: Theory and Algorithms, Springer, Berlin.
12. Meloni, C., 2001, The splittance of a graph and the D-Trails problem. Preprint n.449, Centro Vito Volterra, Università Tor Vergata, Roma.
13. Reeves, C.R., 1993, Modern heuristic tecniques for combinatorial problems, Blackwell Scientific Pubblications, Oxford.
14. Reeves, C.R., 1995, A genetic algorithm for flowshop sequencing, Computers And Operations Research, 22, 5–13.
15. Wolsey, L.A., 1998, Integer Programming, Wiley-Interscience Publication, New York.

Evolutionary Search for Smooth Maps in Motor Control Unit Calibration

Jan Poland, Kosmas Knödler, Alexander Mitterer*, Thomas Fleischhauer*, Frank Zuber-Goos*, and Andreas Zell

Universität Tübingen, WSI-RA, Sand 1, D - 72076 Tübingen, Germany
poland@informatik.uni-tuebingen.de,
http://www-ra.informatik.uni-tuebingen.de

Abstract. We study the combinatorial optimization task of choosing the smoothest map from a given family of maps, which is motivated from motor control unit calibration. The problem is of a particular interest because of its characteristics: it is NP-hard, it has a direct and important industrial application, it is easy-to-state and it shares some properties of the wellknown Ising spin glass model. Moreover, it is appropriate for the application of randomized algorithms: for local search heuristics because of its strong 2-dimensional local structure, and for Genetic Algorithms since there is a very natural and direct encoding which results in a variable alphabet. We present the problem from two points of view, an abstract view with a very simple definition of smoothness and the real-world application. We run local search, Genetic and Memetic Algorithms. We compare the direct encoding with unary and binary codings, and we try a 2-dimensional encoding. For a simple smoothness criterion, the Memetic Algorithm clearly performs best. However, if the smoothness citerion is more complex, the local search needs many function evaluations. Therefore we prefer the pure Genetic Algorithm for the application.

Keywords: Genetic Algorithm, NP-hard, Control Unit Calibration, Variable Alphabet Coding, Hybrid GA, Smooth Maps, Combinatorial Optimization.

1 Introduction

This paper deals with a combinatorial optimization problem that is motivated from the calibration of electronic control units (ECUs) for combustion engines. We briefly sketch the situation. During engine operation, each engine parameter[1] is being controlled by a map that is stored in the ECU as a lookup table. In this way, the engine is adjusted to the actual operating situation, which is necessary to obtain optimal fuel consumption, exhaust emissions, etc. Technically, an operating situation is a combination of engine speed and relative air mass flow and

* BMW Group, 80788 München, Germany.

[1] engine parameters are controlled variables that determine the behaviour of the engine, e.g. ignition timing angle

K. Steinhöfel (Ed.): SAGA 2001, LNCS 2264, pp. 107–116, 2001.

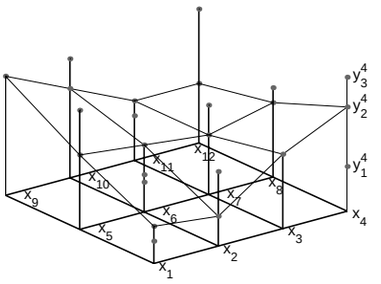

Fig. 1. The context: grid, candidates, and a sample map

thus corresponds to a point in a rectangular domain, the *operating range*. Moreover, the maps in the ECU are defined by a grid $(x_j)_{j=1}^n$ on the operating range, the *operating points*, and the respective co-domain values. If there are m engine parameters, then the control unit must store m maps $\left((y^{(j)(\mu)})_{j=1}^n\right)_{\mu=1}^m$. For operating points that are not grid points, the ECU will interpolate the parameter values. (See e.g. [1] for the technical background).

Our problem now arises in the final phase of the lookup table design. For each operating point x_j, we are given a set of candidates $\{(y_k^{(j)(\mu)})_{\mu=1}^m\}_{k=1}^{n_j}$, each of which defines a combination of values for the engine parameters. These candidates have been obtained by a previous optimization process separately for each operating point, and they have approximately the same quality (fuel-consumption etc.). In order to populate the lookup table, one candidate has to be selected at each operating point. The constraint is the fact that some of the parameters are adjusted *mechanically* and thus have a certain inertia. Hence, one is interested in *smooth* maps for these parameters, in order that the value can be adjusted swiftly when the operating point changes.

Note that this is a *multiple objective optimization* task unless $m = 1$, since we have m maps to smooth and choosing a candidate means fixing *all* parameter values at the corresponding operating point. Thus, the problem is not separable, i.e. it is not possible to optimize each map separately.

Clearly the meaning of "smoothness" has to be defined precisely. The first part of this paper will assume a very simple smoothness criterion. This turns the problem into an easy-to-state combinatorial optimization problem with several interesting properties. The second part of this paper deals with the application.

We use local search heuristics and Genetic Algorithms to find smooth maps. Even with a simple smoothness definition, it can be seen that a pure heuristic is not powerful enough to perform an efficient global optimization. On the other hand, the application implies more complex smoothness criteria. Here, the local search becomes less efficient, and other methods that require deeper insight into the problem structure are very expensive to develop or even intractable. However, we will see that a Genetic Algorithm can still produce good solutions.

2 The Abstract Case

In this section, we will consider an abstract version of our problem and a very simple smoothness definition.

Let $G = (x_j)_{j=1}^n$ be a set of points defined by a rectangular P by Q grid in $\mathbb{R}^2$. For each grid point x_j, a number of candidates $\{y_k^{(j)}\}_{k=1}^{n_j} \subset \mathbb{R}$ is given (see Fig. 1). Then, $M(x_j) = y_{k_j}^{(j)}$ defines a map from G into $\mathbb{R}$, this is the map obtained by choosing the candidate $y_{k_j}^{(j)}$ at the grid point x_j with $1 \leq k_j \leq n_j$ (Fig. 1 shows an example map). Clearly, there are $\prod_{j=1}^n n_j$ different maps, we denote the family of all these maps by $\mathcal{M}$.

This situation is quite similar to the wellknown spin glass model in two dimensions. Therefore, we will use a similar terminology and refer to the *energy* of a map instead of its smoothness or steepness. Then, the problem of minimizing the energy of a map corresponds to the problem of finding an exact ground state of a spin glass model (see e.g. [2]). Of course, there are also relations to other fields and problems with a two-dimensional structure, e.g. image processing.

Definition 1. *For two neighbouring grid points $x_i, x_j \in G$ and a map M, we define the energy of the connection $x_i - x_j$ in the map as*

$$E_{(i,j)}(M) = \mathbf{1}_{|y^{(i)} - y^{(j)}| > c},$$

i.e. the connection energy is 1 if the ordinate distance is greater than a constant threshold $c > 0$, and 0 otherwise. We may fix $c = \frac{1}{2}$ without loss of generality.

Two grid points are neighbouring if they are adjacent in the grid.

Definition 2. *For a map M, we define its energy (i.e. steepness) by the sum over all connection energies:*

$$E(M) = \sum_{i<j \text{ are neighbours}} E_{(i,j)}(M).$$

The above definition has a natural interpretation in terms of inert engine parameters: If the distance of two neighbouring grid points x_i and x_j is too large, then the adjusting time for the engine parameter is larger than a certain limit when the operating point changes from x_i to x_j. The limit could be derived e.g from the time resolution of the control unit.

Since the smoothest map has the lowest energy, our problem can be stated as follows.

Problem ("SmoothMap"): Find the map $M_0 \in \mathcal{M}$ with minimal energy, i.e.

$$E(M_0) = \min_{M_0 \in \mathcal{M}} E(M).$$

Although the above energy definition is very simple, and we have restricted to only one optimization objective ($m = 1$), the resulting abstract problem SmoothMap has very interesting features:

1. It is NP-complete in the grid dimension, even if the number of candidates n_j is limited to 2 for all $1 \leq j \leq n$. This is shown in [3].
2. It admits a PTAS (polynomial time approximation scheme) for certain "non-degenerate" instances. (Instances with a very small energy of the optimal solution are called degenerate, since in this case the approximation quality of the PTAS cannot be assured.) However, the PTAS does not produce good solutions in reasonable time, on the other hand, there is no fully PTAS .
3. If one grid dimension is equal to 1 (the *one-dimensional* case), there is a polynomial time algorithm. This is extendible to the case where one grid dimension is bounded. See [3] for these results.
4. Because of the simple neighbourhood definition, the problem has a strong 2-dimensional local structure.
5. A very appropriate encoding for Genetic Algorithms is the direct encoding, which leads to a variable alphabet. This will be discussed in the sequel.

For the spin glass model, certain cases have been proved to be NP-hard, while other cases are solvable in polynomial time (see [4]), including the simple two-dimensional case. Although the present problem seems to be very similar, its structure is quite different, and so is the proof of its complexity.

2.1 Variable Alphabet Coding

The direct encoding of a map is just a vector with n components $c = (c_j)_{j=1}^n$, each component c_j defining the candidate chosen at the grid point x_j. In terms of Genetic Algorithms, each x_j corresponds to one gene c_j of the chromosome c, and we have

$$c \;=\; (c_j)_{j=1}^n \;\in\; \bigotimes_{j=1}^{n} \{1, \ldots, n_j\}.$$

If this variable alphabet coding is used for Genetic Algorithms, we observe that the conditions and hence the assertions of the schema theorem are not at all affected (cf. [5], [6]).

Since the genes correspond to grid points, a simple 2-dimensional arrangement of the genes resulting in a 2-dimensional chromosome would be natural. Appropriate crossover operators have been suggested e.g. in [7]. On the other hand, a conventional one-dimensional chromosome can be considered as a reference.

In order to show the benefits of the direct encoding, we will additionally try a standard binary encoding as well as a unary encoding. (Here, "unary encoding" means bit-counting, i.e. a sufficiently large part of the bit string is reserved for a grid point and the number of ones in this part define the candidate chosen.)

2.2 A Local Search Operator

Because of its 2-dimensional local structure, the problem admits a very canonical and powerful local search heuristic.

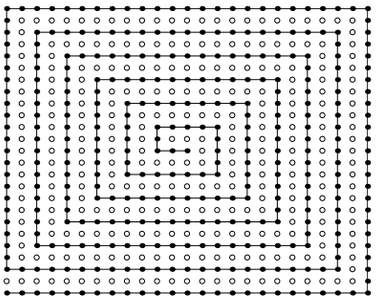

Fig. 2. Construction of a spiral line test instance

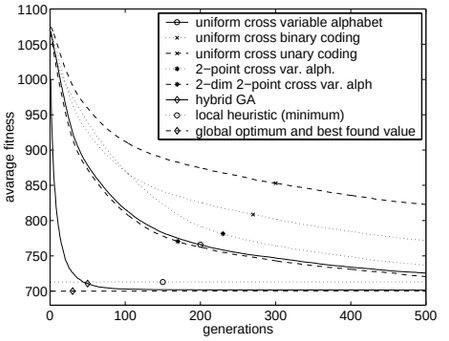

Fig. 3. Fitness curves for the spiral line test instance

Algorithm.
Repeat N times
  Choose $j \in \{1, \ldots, n\}$ randomly
  Find all $k \in \{1, \ldots, n_j\}$ such that the resulting choice has minimal energy
  Choose randomly one of theese k
End Repeat

In order to obtain reference solutions, a good choice for number of loops is $N = 10 \cdot P \cdot Q$, where P and Q are the grid dimensions. On the other hand, we will use the heuristic as mutation operator for the GA thus obtaining a Memetic Algorithm (hybrid GA). In this case, $N = P \cdot Q$ is sufficient.

2.3 Test Instances

We will use three types of test instances of the abstract problem SmoothMap.

1. Random test instances.
   For each grid point j, both the number of candidates and the candidate values are randomly generated. This is the easiest test instance type, the test instances are almost surely nondegenerate. On the other hand, the global optimum is unknown and intractable to compute, so one cannot check if an algorithm succeeds to find it.
2. Line test instances.
   This test instance type permits to compute the global optimum. One starts by placing a line onto the grid, such that the line does not touch itself, i.e. neighbouring grid points that belong to the line are neighbouring in the line. In Fig. 2, the line is spiral-formed, other forms are possible as well, e.g. meander. For the line points, the number of candidates and the candidate values are randomly generated. Then the optimal choice for these points is calculated with the efficient one-dimensional case algorithm (see [3]). Finally the

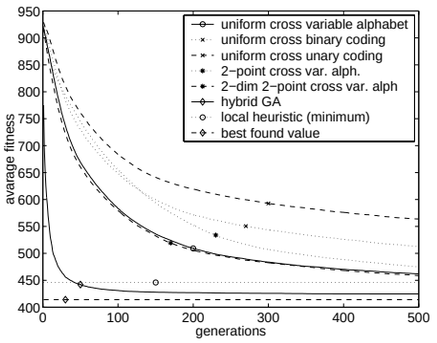

Fig. 4. Fitness curves for the random test instance

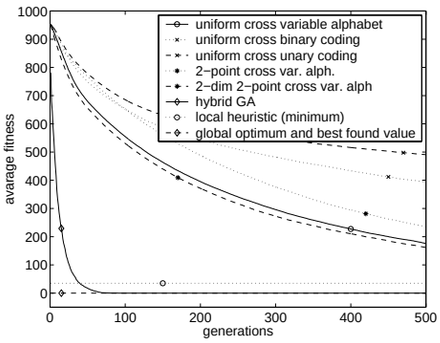

Fig. 5. Fitness curves for the noisy plane test instance

remaining grid points (the "separating" points) are randomly endowed with candidates, where clearly one has to assure that the optimal choice for the line points does not change. The resulting test instance has a global optimum that is easy to compute only if the line is exploited. Moreover, it is almost surely nondegenerate.

3. Noisy plane test instances.
   A very simple way to produce test instances with known global optimum 0 is the following: Start with one candidate for each grid point and arrange these candidates such that the respective neighbouring differences are small, i.e. they form a noisy plane with energy 0. Then add more randomly placed candidates for each grid point, clearly the optimum remains 0. Placing these additional candidates far away from the noisy plane probably raises the optimization difficulty for the local heuristic. The resulting test instances are of course degenerate, and the optimum is easy to find if the existence of the noisy plane is known.

2.4 Experimental Results

The experiments have been performed with a spiral line instance, a random instance and a noisy plane instance. All three of them are defined on a 25×25 grid and contain from 3 to 7 candidates per grid point. These relatively large numbers were chosen in order to obtain difficult test instances and thus a good observation of the convergence speed. The following representations and crossover operators are compared:

1. direct encoding with uniform crossover,
2. binary coding with uniform crossover,
3. unary coding ("bit counting") with uniform crossover,
4. direct encoding with 2-point crossover,

5. 2-dimensional direct encoding with 2-point crossover.

In order to obtain an undistorted measure of the convergence properties of the different representations, we use the pure GA. Additionally, a Memetic Algorithm and the pure local heuristic have been executed.

All code has been implemented in MATLAB. The following GA parameters were used: five parallel populations, each of size $\mu = 200$ (one population of size $\mu = 100$ for the hybrid GA), number of generations $t_{max} = 500$, tournament selection with $q = 5$, crossover and mutation probability $p_{cross} = 0.6$ and $p_{mut} = 1/625$ (for the hybrid GA the local search is performed with probability $p_{mut} = 0.01$). For each setting, 30 runs have been executed. The fitness curves in the plots (Figs. 3 - 5) display the average fitness values over the generations, except for the pure heuristic runs, here the minimum value is shown for reference purpose.

The Memetic Algorithm converges much faster than any of the pure GAs for all test data sets, even with less function evaluations per generation. This was not unexpected because of the strong local structure of the problem. However the pure GA reaches the solution quality of the Memetic Algorithm after a sufficient number of generations, since the combination of conventional mutation and selection can simulate the local heuristic.

The direct encoding performs considerably better than both the binary and the unary coding. This is not only valid for the avarage values: The *best* binary or unary results are even worse than the *worst* variable alphabet results after generation 35 for all three test data sets. The 2-point crossover reduces the convergence speed for the 1-dimensional encoding, while in combination with the 2-dimensional natural encoding the performance improves a little.

The average running time (with a Pentium III, 500 Mhz) is about 24 min for the variable alphabet GA, 89 min for the binary and 57 min for the unary coded GA. This is mainly due to the fact that the fitness function is relatively cheap, while the decoding of the binary or unary bit strings takes some time.

The plots for the first two data sets look very similar. This indicates that the line test instance is good in the sense that its hardness is similar to the random instance, if the line information is not exploited. However, the global optimum 700 was found 7 times by the Memetic Algorithm, while for the random instance the best ever found value 414 was obtained only once. This suggests that the random instance is still harder to optimize. The last test instance is apparently easier for the local heuristic. However, even for this data set, the pure heuristic fails to find the global optimum in any run.

3 The Application

After having investigated the abstract problem, we turn to the application. That is, we are given a set of *operating points* and *candidates* and try to find a candidate choice that results in m *simultaneously* smooth maps.

As already mentioned, the operating range is spanned by the engine speed and the relative air mass flow. We will consider $m = 2$ maps here, one for the inlet valve spread and one for the exhaust valve spread. Both valve spreads are

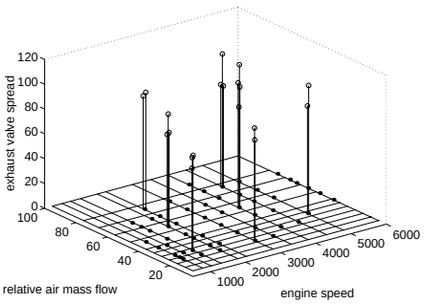

Fig. 6. The application: Operating range, operating points, grid (i.e. ECU operating points) and some exhaust valve spread candidates

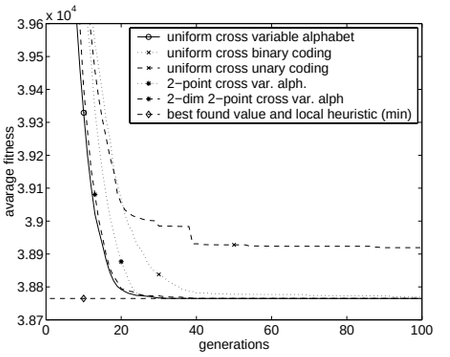

Fig. 7. Fitness curves for the application

mechanically adjusted actuators. Unfortunately, the operating points needed for the lookup tables are no more the equal to the operating points at which the candidates are available (see Fig. 6). Moreover, for the different candidates at each operating point, the location in the operating range varies slightly. I.e. two different candidates for one operating point have not only different exhaust valve spread values, but also slightly different engine speed and air mass values, as can be observed in Fig. 6.

As a consequence, when a selection of candidates is given, one has to apply appropriate interpolation and extrapolation algorithms in order to obtain values for the grid points. After that, one can apply the fitness function for evaluating the map. Here the next question arises: Which is the right smoothness criterion? Clearly there are many possibilities. We will consider a very simple criterion by integrating the square gradients of the two maps:

$$E = \int_{\substack{operating\\range}} \nabla y_{inlet}^2 + \int_{\substack{operating\\range}} \nabla y_{exhaust}^2.$$

Even with this simple formula, the local structure of the energy function becomes unclear with respect to the operating points because of the interpolation and extrapolation process. Thus a local search operator is much less efficient than before, since it cannot save computation time by directly exploiting the local structure. Instead it has to use complete evaluations of the energy function, which is quite expensive.

In terms of multi-objective optimizition, we thus use a simple aggregation method. Clearly, other more sophisticated multi-objective techniques can be expected to produce further interesting results, in particular for growing number of maps m. This should be subject of subsequent research.

We present the experimental results for the data set sketched in Fig. 6. There are 55 operating points endowed with candidates, the maximum number of candidates for one operating point is 4, the grid has dimension 12×12. This is a relatively small data set from the real-world application. Again, we compare variable alphabet coding with uniform crossover, 2-point crossover and 2-dimensional 2-point crossover as well as unary and binary coding with uniform crossover. Since the operating points do not form a grid this time, we used the grid construction algorithm from [8] in order to obtain the 2-dimensional arrangement.

The following GA parameters were used: population size $\mu = 100$, number of generations $t_{max} = 100$, tournament selection with $q = 5$, crossover and mutation probability $p_{cross} = 0.6$ and $p_{mut} = 1/55$. Again, 30 runs have been performed for each setting. We didn't use a Memetic Algorithm, since the local search is quite expensive in this case: One call needs more function evaluations than a whole GA generation, and the local search cannot efficiently exploit the local structure of the problem, as discussed above. Nevertheless we include the results of the pure local search.

Figure 7 shows the average GA fitness curves. This smaller instance is obviously easier to optimize than the 25×25 test instances. Again, direct encoding performs considerably better than both unary and binary coding. This time the best binary or unary results are worse than the worst variable alphabet results after generation 37. The optimal value is found by the variable alphabet GA with any crossover operator in all of the 30 runs. Hence we assume this value to be the global optimum. The binary coded GA obtains it in 27, the unary coded GA in 20 out of 30 runs. The pure local search finds the optimum in 20 out of 30 runs. This seems quite good, but the heuristic does not scale up nicely, it fails to optimize larger instances. And it needs quite many function evaluations, as already mentioned.

This time, the plot indicates that uniform crossover performs slightly better than the 2-dimensional 2-point crossover. This may be due to the fact that the 2-dimensional structure of the operating points is weaker than before. The average running time of the GA is about 7 min, independently of the representation. This is a consequence of the quite complex fitness function, where the decoding time becomes negligible.

4 Discussion

Our study covers two different aspects. On the one hand, the GA is able to optimize the application we started with in a satisfactory way. The quality of the resulting maps is similar or even better compared to the maps that were previously obtained manually by an engineer, a process which took several hours. On the other hand, we presented an easy-to-state problem with interesting features. We studied the performance of different codings and showed that a direct encoding is more suitable for a GA. We also tried a 2-dimensional encoding and saw that it can slightly accelerate the GA performance. This confirmes prior results of different researchers, who find that a d-dimensional encoding can result in a

moderate, but no significant performance gain. Finally, hybridization of a GA with a local search yields superior results, if the local search can be performed efficiently.

Acknowledgments

This research has been supported by the BMBF (grant no. 01 IB 805 A/1).

References

1. A. Mitterer. *Optimierung vielparametriger Systeme in der Antriebsentwicklung, Statistische Versuchsplanung und Künstliche Neuronale Netze in der Steuergeräteauslegung zur Motorabstimmung.* PhD thesis, Lehrstuhl für Meßsystem- und Sensortechnik, TU München, 2000.
2. C. De Simone, M. Diehl, M. Jünger, P. Mutzel, G. Reinelt, and G. Rinaldi. Exact ground states of Ising spin glasses: New experimental results with a branch and cut algorithm. *Journal of Statistical Physics*, 80:487–496, 1995.
3. J. Poland. Finding smooth maps is NP-complete. Preprint, 2001.
4. F. Barahona. On the computational complexity of ising spin glass model. *J.Phys:A: Math.Gen.*, 15:3241–3253, 1982.
5. D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989.
6. J. Holland. *Adaptions in Natural and Artificial Systems.* Ann Arbor: The University of Michigan Press, 1975.
7. T. N. Bui and B. R. Moon. On multidimensional encoding/crossover. In *6th International Conference on Genetic Algorithms*, pages 49–56, 1995.
8. J. Poland, K. Knödler, and A. Zell. On the efficient arrangement of given points in a rectangular grid. In E. J. W. Boers et al., editor, *Applications of Evolutionary Computation (LNCS 2037)*, pages 110–119, 2001.

Some Notes on Random Satisfiability

Gregory B. Sorkin

IBM T.J. Watson Research Center, Yorktown Heights NY 10598,
sorkin@watson.ibm.com, www.research.ibm.com/people/s/sorkin

Abstract. 3-SAT is a canonical NP-complete problem: satisfiable and unsatisfiable instances cannot generally be distinguished in polynomial time. However, random 3-SAT formulas show a phase transition: for any large number of variables n, sparse random formulas (with $m \leqslant 3.145n$ clauses) are almost always satisfiable, dense ones (with $m \geqslant 4.596n$ clauses) are almost always unsatisfiable, and the transition occurs sharply when m/n crosses some threshold. It is believed that the limiting threshold is around 4.2, but it is not even known that a limit exists. Proofs of the satisfiability of sparse instances have come from analyzing heuristics: the better the heuristic analyzed, the denser the instances that can be proved satisfiable with high probability. To date, the good heuristics have all been extensions of unit-clause resolution, all expressible within a common framework and analyzable in a uniform manner through the differential equation method. Any algorithm expressible in this framework can be "tuned" optimally. This tuning requires extending the analysis via the differential equation method, and making use a "maximum-density multiple-choice knapsack" problem. The structure of optimal knapsack solutions elegantly characterizes the choices made by an optimized algorithm. Optimized algorithms result in improving the known satisfiability bound from density 3.145 to 3.26. Many open problems remain. It is non-trivial to extend the methods to 4-SAT and beyond. If results are to be applicable to "real-world" 3-SAT instances, then the theory should be extended to formulas that need not be uniformly random, but obey some weaker conditions. Also, there is theoretical evidence that in the unsatisfiable regime it is difficult to prove the unsatisfiability of a given formula, while in the known region of satisfiability, linear-time algorithms produce satisfying assignments with high probability. Is the unsatisfiable regime truly hard, and is the whole of the satisfiable regime truly easy? In particular, as the scope of myopic, local algorithms is expanded so that they examine more and more variables, can such algorithms solve random instances arbitrarily close the the threshold density?

Keywords: 2-SAT; 3-SAT; phase transition; complexity; satisfiability; randomized analysis of algorithms; average-case analysis; differential equation method.

K. Steinhöfel (Ed.): SAGA 2001, LNCS 2264, pp. 117–130, 2001.

1 Introduction

Random 3-SAT formulas show a fascinating "phase transition" as a function of their "density" – the ratio of the number of clauses to the number of variables. A random formula with density less than 3.145 is almost always satisfiable, while one with density greater than 4.596 is almost always unsatisfiable. It is conjectured that the transition occurs around 4.2, but it is not even known that all formulas with many variables have a single, common density threshold.

To make this precise, let $F_k(n,m)$ be a random k-SAT formula formed by selecting uniformly, independently, and with replacement, m clauses among all $2^k\binom{n}{k}$ non-trivial clauses of length k, i.e., clauses with k distinct, non-complementary literals.[1] Let us say that a sequence of events $\mathcal{E}_n$ holds *with high probability* (w.h.p.) if $\lim_{n\to\infty}\Pr[\mathcal{E}_n]=1$ and *with positive probability* if $\liminf_{n\to\infty}\Pr[\mathcal{E}_n]>0$. The following was first put forward in [7] and has become a folklore conjecture.

Satisfiability Threshold Conjecture *For every $k\geqslant 2$, there exists a constant r_k such that for all $\varepsilon>0$,*
- If $m\leqslant(r_k-\varepsilon)n$ then w.h.p. $F_k(n,m)$ is satisfiable.
- If $m\geqslant(r_k+\varepsilon)n$ then w.h.p. $F_k(n,m)$ is unsatisfiable.

The satisfiability threshold conjecture has attracted attention in computer science, mathematics, and more recently in mathematical physics [16,17,18]. Below we briefly review the current state of knowledge.

For the linear-time solvable $k=2$ case, the conjecture was settled independently, $r_2=1$, by Chvátal and Reed [7], Goerdt [12], and Fernandez de la Vega [9]. Further, Bollobás et al. [3] recently determined the problem's "scaling window": with $m=r_2n$, and $r_2=1+\lambda n^{-1/3}$, any desired probability p of satisfiability can be achieved by an appropriate setting of $\lambda=\lambda(p)$ (independent of n, asymptotically).

For $k\geqslant 3$, it is not even known whether there exists a single threshold value r_k valid for all (asymptotically large n). By contrast, the existence of a threshold *sequence*, $r_k(n)$, is proved by a theorem of Friedgut [10].

Theorem 1 ([10]). *For every $k\geqslant 2$, there exists a sequence $r_k(n)$ such that for all $\varepsilon>0$,*
- If $m\leqslant(r_k(n)-\varepsilon)n$ then w.h.p. $F_k(n,m)$ is satisfiable.
- If $m\geqslant(r_k(n)+\varepsilon)n$ then w.h.p. $F_k(n,m)$ is unsatisfiable.

An immediate corollary of Theorem 1 is that if $F_k(n,nr)$ is satisfiable with positive probability then, for all $\varepsilon>0$, $F_k(n,(r-\varepsilon)n)$ is satisfiable w.h.p. Of course it remains open whether $r_k(n)$ converges; otherwise the satisfiability threshold conjecture would be resolved.

[1] While we adopt the $F_k(n,m)$ model throughout the paper, all the results presented hold in all standard models for random k-SAT, e.g. when clause replacement is not allowed and/or when each k-clause is formed by selecting k literals uniformly at random with replacement.

Existing proofs that dense formulas are typically unsatisfiable all rely on probabilistic counting arguments. The simplest of these forms a small part of Chvátal and Szemerédi [6]. The probability that any truth assignment satisfies a random clause is 7/8, the probability that it satisfies all m clauses in a random formula is $(7/8)^m$, and so the expected number of assignments satisfying a random formula is $2^n(7/8)^m$; for $m/n > \ln(2)/\ln(8/7) \approx 5.19$, the expected number of satisfying assignments (and the probability that there is any satisfying assignment) is small. After a series of improvements in the bound, the current best, due to Janson, Stamatiou and Vamvakari [13], is 4.596.

The greater part of Chvátal and Szemerédi [6] is to show that even random unsatisfiable formulas seem to be computationally intractable. Specifically, they show that for many unsatisfiable formulas (a polynomial fraction of them), every "resolution" proof of unsatisfiability is exponentially long, never mind the difficulty of finding it. This does not rule out the possibility of short unsatisfiability proofs of some other sort, but there has been limited progress in this regard. Friedman and Goerdt [11] show that random 3-SAT formulas with $\Omega(n^{3/2+\varepsilon})$ clauses can be certified unsatisfiable in polynomial time using spectral methods, but no such method is known for formulas of any linear size. This suggests that unsatisfiability of even random, linear-sized formulas is genuinely hard, like general (un)satisfiability.

By contrast, although 3-SAT is NP-complete, satisfiability for random formulas below threshold appears to be easy. In fact, proofs that sufficiently sparse formulas are satisfiable have all come from analyzing algorithms that satisfy them with positive or high probability, and these algorithms all run in linear time. The first high-probability heuristic was given by Broder, Frieze and Upfal [4] who proved that w.h.p. the *pure literal* heuristic succeeds on $F_3(n, rn)$ for $r \leqslant 1.63$, but fails for $r \geqslant 1.7$. This work was predated by Chao and Franco [5], who showed that a certain heuristic succeeds with positive probability for $r < 2.9$; unfortunately this had to await [10] before it could be interpreted as a satisfiability threshold bound.

2 Random 3-SAT below the Threshold

The more powerful the algorithm analyzed, the denser the formulas that can be proved satisfiable with high probability. The best algorithms have all been of a type Achlioptas and Sorkin [2] call "myopic", and they show how to optimally tune any of them. The technical details are supplied in [2], so the following is a still detailed but more conversational presentation.

In the course of the algorithm, the random formula F will repeatedly be "reduced": when a literal A is set to True (we will write $A = 1$), any clause in which A appears is satisfied, and may be discarded from F, while any in which $\bar{A}$ appears fails to be satisfied by $\bar{A}$ itself, and so may be shortened by discarding $\bar{A}$ from it (a 3-clause becomes a 2-clause, a 2-clause becomes a 1-clause, and a 1-clause becomes a 0-clause — meaning that F is unsatisfied). Letting m_1, m_2, and m_3 denote the number of "unit" (1-variable) clauses, 2-clauses, and 3-clauses

in F, and n as before the number of variables, we define in the obvious way a random formula $F \in \mathcal{F}(\setminus, \Updownarrow_\infty, \Updownarrow_\in, \Updownarrow_\ni)$.

2.1 A Myopic Algorithm

Rather than setting forth a formal definition, consider the following example of a "1-variable" myopic algorithm.

One **round**:

- **Free move**:
  - Choose a **random 2-clause** literal, A.
  - **Expose** all occurrences of A and $\bar{A}$, giving counts $\boldsymbol{A} = (A_2^+, A_2^-, A_3^+, A_3^-)$.
  - Set $A = \sigma(\boldsymbol{A})$. (1=true, 0=false.)
  - Reduce F.
- **Forced moves**:
  - While F has any **unit clauses**, choose one at random, set its literal $B = 1$, and reduce F.

That is, first, a random literal A is chosen from a 2-variable clause.[2] By "revealing" all occurrences of A and $\bar{A}$, we mean identifying (or equivalently, choosing) the clauses in which they appear and thereby conditioning the random formula F, but leaving F otherwise unconditioned. The counts $\boldsymbol{A} = (A_2^+, A_2^-, A_3^+, A_3^-)$ simply denote the number of occurrences of A in 2-clauses (at least 1, by definition), the number of occurrences of $\bar{A}$ in 2-clauses, and similarly the number of occurrences of A and $\bar{A}$ in 3-clauses. The assignment of True or False to A is made as a function σ of $\boldsymbol{A}$, $A = \sigma(\boldsymbol{A})$. *Choosing the "policy"* $\sigma : \mathbb{Z}^4 \to \{0, 1\}$ *optimally* is exactly what we mean by the optimal tuning of this myopic algorithm. In addition to its explicit argument, $\sigma(\boldsymbol{A})$ is also permitted to depend on the 2- and 3-clause densities $\rho_2 = m_2/n$ and $\rho_3 = m_3/n$; we may write $A = \sigma(n, \rho_2, \rho_3; \boldsymbol{A})$ to be more explicit. There is no role for m_1 because in the free move of each round, $m_1 = 0$.

If at any stage F contains an empty clause, then it is unsatisfied by the constructed assignment; the algorithm terminates in failure. Otherwise, rounds are made until F is empty; the empty formula is satisfied, and the original formula is satisfied by the constructed assignment. Variables set by the algorithm are not reversed (except in a slight variant of the standard algorithm, discussed soon).

[2] Initially, of course, F has only 3-clauses, no 2-clauses. This is a minor matter; for our purposes it suffices, for example, simply to add a random 2-clause to F if it has none. But after the beginning, it will have plenty, so this is a non-issue.

2.2 Issues in the Algorithm's Analysis

There are several obvious questions. What can be said about the intermediate formulas F' derived as the algorithm runs? Why is σ allowed to depend on ρ_2 and ρ_3, but not on all of n, m_2, and m_3? Why is the setting of $A = \sigma(\boldsymbol{A})$ not trivial — set A so as to satisfy as many clauses as possible? What is the algorithm's probability of succeeding? What does it mean to optimize the algorithm, and what choice of policy function σ achieves this?

2.3 Intermediate Formulas

First, if any *intermediate formula* F' in the algorithm's execution is satisfiable, then the original formula F is also satisfiable: Set each variable in F that is also present in F' the way it is set to satisfy F', and set the variables in F missing from F' the way they were set during the algorithm's execution. If F' is unsatisfiable, then the algorithm has failed; F may or may not be satisfiable.

The intermediate formulas produced as rounds continue are themselves random, as per the following lemma.

Lemma 1. *Let $F \in \mathcal{F}(n; m_2, m_3)$ be a random formula. Apply one round of any myopic algorithm to F, producing formula F'. Conditional upon the round setting n_0 variables, and F' having (m'_2, m'_3) 2- and 3-clauses, F' is a random formula in $\mathcal{F}(n - n_0; m'_2, m'_3)$.*

This is a simple fact: all that is known about the literals in F' are that they are not any of the literals that have been revealed.

2.4 Success and Failure

Next, what about *success or failure* of the algorithm? In a round, the algorithm will fail only if two contradictory unit clauses are generated, B and $\bar{B}$. But w.h.p. the number of literal occurrences exposed during any move is $O(1)$; also, as each unit clause is resolved, the expected number of new unit clauses generated is ρ_2, so *as long as* the 2-clause density $\rho_2 < 1$, w.h.p. the number of literal occurrences exposed in a round is $O(1)$. Thus, as long as $\rho_2 < 1$, w.h.p. the total number of unit clauses generated in a round is $O(1)$, and so the probability that any two are in contradiction is $O(1/n)$. That is, each round succeeds with probability $1 - O(1/n)$; the rounds are all independent; and so the algorithm's success probability over all $O(n)$ rounds is $\Theta(1)$. If ever the 2-clause density rises above 1, then it follows from the unsatisfiability of random 2-SAT with such density that the current working formula F is w.h.p. unsatisfiable, and so w.h.p. the algorithm must fail. The way it does so is that if $\rho_2 > 1$, then as each unit clause is satisfied, it creates on average more than one new unit clauses, the "branching process" explodes, many unit clauses are generated within a single round, and it is overwhelmingly likely that two of them contradict one another.

So as long as $\rho_2 < 1$ — and we will ensure that this is so — the algorithm fails with only small probability in a round, and only constant probability over

all. For convenience, we may ignore these chance failures, for either of two reasons. One reason is that Friedgut's result says that a random formula of any fixed density (away from the one critical value) is either satisfiable w.h.p. or is unsatisfiable w.h.p. If our algorithm succeeds with constant probability, then it must be that F is satisfiable w.h.p., which is what we originally wished to show. The other reason is that we can modify the algorithm to make it succeed with high probability, rather than merely with constant probability. This is a bit tricky, and not necessary to the understanding of the main results.

The modification requires a more exact specification of the algorithm: after the free move, set all unit-clause literals in this first "layer" simultaneously, then set all newly created unit-clause literals (in the second layer) simultaneously, and so forth, until a layer is generated which has contradictory clauses. The literals in the last layer were all there because they appeared, in 2-clauses, with negations of the literals in the second-last layer. Thus, reversing the settings of the literals in the second-last layer, the literal occurrences defining the last layer all lie in already-satisfied clauses. Similarly, reverse the settings of the literals in the third-last layer to satisfy the clauses containing the literal occurrences defining the second-last layer. Repeat, finally reversing the setting of the literal chosen in the free move. With the setting of the various literals reversed, a different set of unit clauses is generated; follow the usual procedure to resolve them, the new unit clauses they generate, and so forth. Note that the original number of unit clauses, and the new number, are w.h.p. both $O(1)$, so the probability that there is a conflict within the new unit clauses, or between the new ones and the re-set old ones, is $O(1/n)$. The probability that a round fails both the normal procedure and this backup procedure is $O(1/n^2)$; over all $O(n)$ rounds, the probability of any failure is $O(1/n)$, and so the algorithm succeeds w.h.p.

Combining the preceding remarks, we run the algorithm until either $\rho_2 \geqslant 1$ (in which case w.h.p. the algorithm has failed, and we know nothing about the original formula) or until we reach a point (ρ_2, ρ_3) where it is known that almost all formulas are satisfiable (in which case w.h.p. the algorithm has succeeded). One or the other outcome will always occur. Moreover, as will be seen, for a given initial pair of densities (ρ_2, ρ_3), one of the the outcomes will occur almost always, so for random formulas with given parameters, either w.h.p. we demonstrate satisfiability, or w.h.p. we fail to demonstrate anything.

2.5 Optimizing Policies, Optimal Algorithms, and Parametrization

Now we take up our last questions, about the optimal setting of the free-round variable A, and the optimal choice of the policy function σ. First, we cannot simply set A "to maximize the number of satisfied clauses" because one setting of A might satisfy more 2-clauses, while the opposite setting might satisfy more 3-clauses, and in such a case the decision is not clear. (The intuition is correct, that if one setting maximizes both, it should be used.) The function σ should of course depend on $(A_2^+, A_2^-, A_3^+, A_3^-)$, which contains all the information (revealed randomness) about the structure of the particular random formula F. We allow σ to depend additionally *not* on all of n, m_2, and m_3 (all the information there

is), but only on ρ_2 and ρ_3, because in two senses, **$(\boldsymbol{\rho_2, \rho_3})$ is the "correct" parametrization.**

First, with the particular functions $\sigma(\rho_2, \rho_3; A_2^+, A_2^-, A_3^+, A_3^-)$ that will be of interest to us, the parameters of the formulas F' produced during the execution of the algorithm will, almost surely, almost exactly, follow a predictable trajectory through (ρ_2, ρ_3)-space. (See Figure 1.) If the trajectory — which of course depends on the initial densities (ρ_2, ρ_3) — first passes through a region where formulas are known to be satisfiable w.h.p. (such as $\rho_2 < 1$ and $\rho_3 < 2/3$) then the original formula was satisfiable w.h.p.; if it first passes through a point with $\rho_2 \geqslant 1$, then we can say nothing.

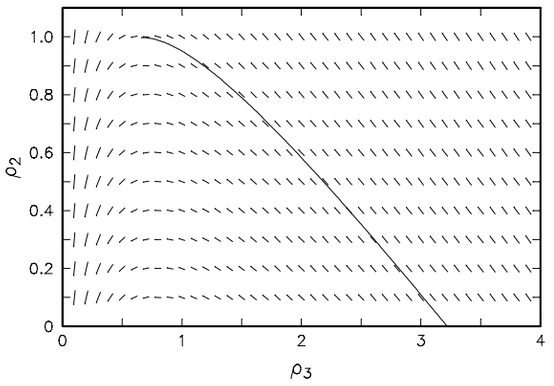

Fig. 1. Optimal gradients $d\rho_2/d\rho_3$ for "one variable at a time", and the trajectory starting from $(3.22, 0)$ and stopping when $\rho_3 < 2/3$.

Second, no algorithm of the same type, but using all the available information in a policy $A = \sigma(n, m_2, m_3; A_2^+, A_2^-, A_3^+, A_3^-)$, can give a better (lower) trajectory; therefore no such more general policy can allow us to reach a stronger conclusion about satisfiability. In other words, for our purposes, a policy $\sigma(\rho_2, \rho_3; \boldsymbol{A})$ is optimal.

Let us argue these points in a little more detail.

2.6 Fixed Policies

Suppose that the policy does not depend on (ρ_2, ρ_3), i.e., $\sigma(\rho_2, \rho_3; \boldsymbol{A}) = \sigma(\boldsymbol{A})$. Consider a round of the algorithm, when the working formula has densities (ρ_2, ρ_3). In the free move beginning the round, the probability distribution over tuples $\boldsymbol{A} = (A_2^+, A_2^-, A_3^+, A_3^-)$ can be calculated almost exactly. For example, $A_2 = 1 + X(\rho_2)$, where $X(\lambda)$ indicates a Poisson random variable with parameter λ; the other variables simply have Poisson distributions (no additive constants), and the variables are approximately independent.

The tuple determines, through the policy σ, the setting $A = \sigma(\boldsymbol{A})$, and in turn the number of eliminated and reduced clauses of each size (including the number of unit clauses formed). For each unit clause resolved (each forced move), again, it is straightforward to compute the distribution of the number of clauses of each size eliminated and reduced. The expected number of forced moves can also be calculated. In short, for each tuple $\boldsymbol{A}$, given $\sigma(\boldsymbol{A})$, we can calculate the expected changes to n, m_2, and m_3 in the course of the round. (Details of these calculations can be found in [2], but they are irrelevant to the thrust of the argument.) That is, we can calculate the expected change in the number of variables,

$$\mathbb{E}(\Delta n) = \sum_{\boldsymbol{A}} \Pr(\boldsymbol{A}) \mathbb{E}(\Delta n | \boldsymbol{A}, \sigma(\boldsymbol{A}))$$

and likewise for $\mathbb{E}(\Delta m_2)$ and $\mathbb{E}(\Delta m_3)$. All such changes will be small — just $O(1)$ — and so the changes to the parameters (ρ_2, ρ_3) will be just $O(1/n)$.

Since the key parameters (ρ_2, ρ_3) change very little in a round, we are free to repeat the analysis over a number of rounds ω, where $\omega = \omega(1)$ (large), but $\omega = o(n)$ (small enough that (ρ_2, ρ_3) will only change by $o(1)$). Over ω rounds, the number of times any tuple $\boldsymbol{A}$ occurs in the free move is determined almost exactly; given $\boldsymbol{A}$, the *expected* changes to n, m_2, and m_3 in the whole round are determined by $\boldsymbol{A}$ and $\sigma(\boldsymbol{A})$; and by a law of large numbers, almost surely, the actual change over ω rounds in the number of variables satisfies

$$\begin{aligned} \Delta_\omega n &= \omega \mathbb{E}(\Delta n)\,(1 + o(1)) \\ &= \omega \sum_{\boldsymbol{A}} \Pr(\boldsymbol{A}) \mathbb{E}(\Delta n | \boldsymbol{A}, \sigma(\boldsymbol{A}))\,(1 + o(1)), \end{aligned}$$

and similarly for $\Delta_\omega m_2$ and $\Delta_\omega m_3$. Since $\rho_2 = m_2/n$,

$$\mathbb{E}(\Delta \rho_2) = \mathbb{E}\left(\frac{m_2 + \Delta m_2}{n + \Delta n} - \frac{m}{n}\right) = \frac{1}{n}\mathbb{E}(\Delta m_2 - \rho_2 \Delta n)\,(1 + o(1)). \tag{1}$$

It follows that the change in ρ_2 over ω rounds is almost surely almost exactly equal to $\mathbb{E}(\Delta_\omega \rho_2)$, like the changes in n, m_2, and m_3. Likewise, of course, for ρ_3.

Moreover, because the function $\Pr(\boldsymbol{A})$ is smooth in ρ_2 and ρ_3, so are the expected changes $\mathbb{E}(\Delta_\omega \rho_2)$ and $\mathbb{E}(\Delta_\omega \rho_3)$: this pair defines a smooth *vector field*. (Such a vector field would be similar to the *line field* of Figure 1, but where a line field has only slopes at each point, a vector field has a slope and magnitude at each point.) Thus, the *differential equation method* [14,19], allows us to conclude that there is a trajectory almost surely almost exactly followed by (ρ_2, ρ_3) over many rounds, and the derivative of this trajectory matches the vector field (in the same way that the trajectory shown in Figure 1 matches the slopes of its line field).

2.7 Optimized Policies

Intuitively, the best choice of policy $\sigma(\boldsymbol{A})$ is one giving the "shallowest" slopes $\Delta\rho_2/\Delta\rho_3$ — i.e., since all the slopes are negative, maximizing them — for that

means that starting from a denser formula (a point further to the right on the X axis) we may still enter the known satisfiability region.

From (1),

$$\frac{\mathbb{E}(\Delta\rho_2)}{\mathbb{E}(\Delta\rho_3)} = \frac{\mathbb{E}(\Delta m_2 - \rho_2 \Delta n)}{\mathbb{E}(\Delta m_3 - \rho_3 \Delta n)}\,(1 + o(1))$$
$$= \frac{\sum_{\boldsymbol{A}} \Pr(\boldsymbol{A})[\mathbb{E}(\Delta m_2 \,|\, \boldsymbol{A}, \sigma(\boldsymbol{A})) - \rho_2 \mathbb{E}(\Delta t \,|\, \boldsymbol{A}, \sigma(\boldsymbol{A}))]}{\sum_{\boldsymbol{A}} \Pr(\boldsymbol{A})[\mathbb{E}(\Delta m_3 \,|\, \boldsymbol{A}, \sigma(\boldsymbol{A})) - \rho_3 \mathbb{E}(\Delta t \,|\, \boldsymbol{A}, \sigma(\boldsymbol{A}))]}(1 + o(1)). \quad (2)$$

The maximization can be performed efficiently through a "maximum-density multiple-choice knapsack" formulation, as detailed in [2].[3] Roughly, replacing "cases" $\boldsymbol{A}$ with i, and policy choices $A = \sigma(\boldsymbol{A})$ with $j = \sigma(i)$, (2) can be generalized as

$$\frac{\sum_i y(i, \sigma(i))}{\sum_i x(i, \sigma(i))}, \quad (3)$$

and the goal is, given all values $x(i, j)$ and $y(i, j)$, to choose the mapping $\sigma : i \mapsto j$ so as to maximize (3).

Returning to the SAT context, because $\Pr(\boldsymbol{A})$ depends on (ρ_2, ρ_3), so does the optimized policy $\sigma(\rho_2, \rho_3; \boldsymbol{A})$ — the optimized policy is *not* a fixed policy. This must be taken into account in our analysis, which extends over many rounds, during the course of which (ρ_2, ρ_3) changes, and thus σ changes. In fact, this is a real issue. For any two points (ρ_2, ρ_3) and (ρ_2', ρ_3'), however close, there is at least some "case" $\boldsymbol{A}$ for which the optimized policies differ; when this case occurs, two policies will yield different values $\mathbb{E}(\Delta n|\boldsymbol{A})$ and so forth. Since any case $\boldsymbol{A}$ occurs with positive probability, even at arbitrarily nearby points, the two optimized policies will produce *different* values $\mathbb{E}(\Delta\rho_2)$ and $\mathbb{E}(\Delta\rho_3)$ for the expected changes over a single round. That is to say, if in Figure 1 the optimized policy's line field $\mathbb{E}(\Delta\rho_2)/\mathbb{E}(\Delta\rho_3)$ were replaced by its vector field $(\mathbb{E}(\Delta\rho_2), \mathbb{E}(\Delta\rho_3))$, that vector field would be almost everywhere discontinuous.

Fortunately, it is in the nature of the optimized policy that the *slopes* $\mathbb{E}(\Delta\rho_2)/\mathbb{E}(\Delta\rho_3)$ change *smoothly*. Were the optimized slope much better at (ρ_2, ρ_3) than at a nearby point (ρ_2', ρ_3'), the optimized (ρ_2, ρ_3) policy used at (ρ_2', ρ_3') would give essentially the same slope as it did at (ρ_2, ρ_3) (fixed policies have smooth line fields), contradicting the supposed optimization of the

[3] In principle, since this maximization is just used for the algorithm-design phase, it need not be efficient; we could devote an arbitrary amount of computational resource to doing it once. But in practice it is very useful to have an efficient method, and that of [2] works in near-linear time. Here we measure run time as a function of the number of different cases $\boldsymbol{A}$ considered (cases i in the general version). In practice, we optimize the policy $\sigma(\boldsymbol{A})$ for a set of cases $\boldsymbol{A}$ covering most of the probability space, and take some default policy choice such as $\sigma(\boldsymbol{A}) = 1$ on the rest. Furthermore, the the method of solving the knapsack problem evinces structural aspects of the optimum, and these are interesting in their own right; for example, at each point in (ρ_2, ρ_3)-space, whether to set $\sigma(\boldsymbol{A}) = 0$ or 1 depends on the test $(A_2^+ - A_2^-) <> \lambda(A_3^+ - A_3^-)$, for some "constant" $\lambda(\rho_2, \rho_3)$.

different policy at (ρ_2', ρ_3'). That is, while an optimized policy's vector field is discontinuous, its line field is smooth. It follows that the optimized policy is again susceptible to the same sort of analysis as a fixed policy, with a slight extension to the differential equation method to work with the line field and not the vector field.

2.8 Optimal Policies

While we have argued intuitively for our method of optimizing policies by maximizing slopes, how do we know that such policies are indeed optimal? In particular, how can we compare them with policies that may depend on additional information (n, m_2, and m_3, rather than just ρ_2 and ρ_3), and that may not be well-behaved, and therefore not in any way susceptible to analysis via the differential equation method?

The answer is that, for an arbitrary policy, we can still compute the expected changes $\mathbb{E}(\Delta n)$ et cetera over one round, and therefore $\mathbb{E}(\Delta\rho_2)/\mathbb{E}(\Delta\rho_3)$ over a round. By definition, our "optimized" policy maximized this ratio. Let us take the optimal ratio and generously add a small value $\varepsilon > 0$ to it, to give ourselves an unfair edge. Then any other policy gives a strictly smaller slope. We can conclude that, over a large number of rounds, any competing policy produces a sequence of values (ρ_2', ρ_3') almost surely lying strictly above the trajectory given by our policy. That it can never cross back below again follows from real analysis, taking advantage of Lipschitz continuity of our optimized policy; no assumptions on the competing policy are needed. Again exploiting Lipschitz continuity, the optimized policy is only some $\delta(\varepsilon)$ worse than the ε-boosted version, and any competing policy is strictly worse than the boosted one, so it follows that the competing policy is no better than the (unboosted) optimized policy.

Thus, our optimized policy truly is optimal.

Optimizing the 1-variable algorithm presented at the beginning of this section gives precisely the outcome illustrated in Figure 1. Starting from $(\rho_2, \rho_3) = (0, 3.22)$, we come to a point $(1, 2/3 - \varepsilon)$ which is known to be feasible; thus, a random 3-SAT formulas with n variables and $3.22n$ or fewer clauses is almost surely satisfiable. This result, in [2], already improved on the previous best satisfiability bound.

2.9 Better Algorithms

Following exactly the same methods as above, we can consider other myopic algorithms. The first candidate is the algorithm considered by Achlioptas in [1], where instead of choosing one literal A from a 2-clause, we choose both its literals, say A and B.[4] Achlioptas showed that whereas the best density that had been proved satisfiable with a 1-variable algorithm was 3.003, with the 2-variable algorithm he could achieve a bound of 3.145.

[4] If we were to choose two literals independently, from different clauses, then because w.h.p. each has a finite "sphere of influence", there is an $O(1/n)$ chance that they interact, and this is essentially the same as the 1-variable algorithm.

The 2-variable algorithm can be optimized the same procedures; that the policy σ now depends on parameters $(A_2^+, A_2^-, A_3^+, A_3^-, B_2^+, B_2^-, B_3^+, B_3^-)$ does not change anything. Interestingly, the optimal policy for the 2-variable algorithm achieves a bound of 3.19 — better (of course) than Achlioptas' bound of 3.145, but not as good as the bound of 3.22 achieved with the optimized 1-variable algorithm. There is no obvious explanation for this, except that while selecting two variables at once can indeed help to reduce the number of clauses faster, it also certainly reduces the number of variables faster (by 2 instead of 1 in the first instance), and so it is plausible that the densities (clauses per variable) it achieves are less good. Another aspect may be that when one variable is chosen, there are two ways to set it; when two variables are chosen from the same clause, there are only 3 possible settings.

An algorithm that is clearly at least as good as both the 1- and 2-variable algorithms is a mixed algorithm. It randomly chooses a single literal A from a 2-clause and decides, as a function of the counts $\boldsymbol{A}$, whether to set $A = 1$, set $A = 0$, or look at A's companion literal B, observe its counts $\boldsymbol{B}$, and then decide how to set both A and B. It is at least as good as the 1-variable algorithm because it has the option of never looking at B, and at least as good as the 2-variable algorithm because it has the option of always looking at B. Optimizing the policy for this algorithm requires a small extension of the "knapsack" method, but can still be done in essentially the same way. It results in a satisfiability bound of 3.26, the best known to date.

2.10 Further?

The method for optimizing a policy for a myopic 3-SAT algorithm does not generalize obviously to k-SAT for any $k > 3$. The 3-SAT result relied on the fact that a random 3-SAT formula is completely parametrized by n, m_1, m_2, and m_3; that between rounds of a myopic algorithm $m_1 = 0$; that the remaining parameters n, m_2, and m_3 can be sufficiently represented by just the densities $\rho_2 = m_2/n$ and $\rho_3 = m_3/n$; and that optimization is connected with maximizing a single parameter, namely the slope $\Delta\rho_2/\Delta\rho_3$. Even for 4-SAT, while it is still true that a random formula can adequately be parameterized by its densities, now there are 3 densities (ρ_2, ρ_3, and ρ_4), and from each point in density space, there is a 2-parameter family of directions. It is not clear how to choose the best direction.

Even for 3-SAT, as remarked in the introduction, [6] provides theoretical evidence that in the unsatisfiable regime it is difficult to prove the unsatisfiability of a given formula. On the other hand, the only way that we know the *satisfiable* regime is through algorithms that succeed w.h.p. The best is the mixed algorithm described above, showing that formulas $F \in \mathcal{F}(n, 3.26n)$ are satisfiable w.h.p.; like all myopic algorithms, it runs in essentially linear time. So while in the unsatisfiable regime we know that there are many formulas whose unsatisfiability is hard to prove (by any known means), in all we know of the satisfiable regime, almost all formulas can be satisfied by a linear-time algorithm.

This suggests dual questions to which we have no answers: (1) Is it truly hard to prove that a random formula in the unsatisfiable regime is unsatisfiable? (2) Is it true that a random formula anywhere in the satisfiable regime (not just the part of the regime that we know today) can be satisfied in polynomial time, or even linear time? As the mixed algorithm, which looks at a broader "sphere of influence" than the 1-variable algorithm, increases the satisfiability bound from 3.22 to 3.26, can further extending the sphere of influence give algorithms which work right up to the satisfiability threshold, or are more sophisticated methods required?

3 Relaxing the Uniform Randomness Assumption

Average-case analysis of algorithms is notoriously tricky, because the meaning is not clear: real-world instances are not uniformly random. However, they are probably not worst-case, either.

A small step towards generalizing beyond uniform random formulas is taken by Cooper, Frieze, and Sorkin [8]. Here, rather than specifying only the number of clauses, the degree (number of appearances) of each literal is given.

This result can be seen as filling in an obvious gap. The phase transition in 2-SAT at $\rho_2 = 1$ is analogous to the birth of a giant component in a random graph as the density of edges per vertex passes $m/n = 1/2$; both results can be seen as the point when the average fanout of a branching process exceeds 1. (In the graph model, the branching process goes from a vertex to its neighbors, and so on; in the 2-SAT model, from a setting of a literal to its implications and so on.)

Molloy and Reed [15] generalized the standard $G(n, m)$ random graph model to random graphs with a specified degree sequence, asking what property of the degree sequence controlled whether the graph would, w.h.p., have a giant component. Again it is natural to ask the analogous question for 2-SAT, and [8] resolved that the answer too is analogous. The main result is as follows.

A degree sequence $\mathbf{d} = d_1, \bar{d}_1, \ldots, d_n, \bar{d}_n$ is *proper* if

- every $d_i, \bar{d}_i \leqslant n^\alpha$ where $\alpha < 1/13$ is a constant.
- $\sum_i (d_i + \bar{d}_i)$ is even.

Theorem 2. *Let* $\mathbf{d}$ *be proper and let* $0 < \varepsilon < 1$ *be constant. Then if* F *is chosen uniformly at random from all formulas having* $\mathbf{d}$ *as their degree sequence, then with* $D_1 = \sum_{i=1}^{n} (d_i + \bar{d}_i)$ *and* $D_2 = \sum_{i=1}^{n} d_i \bar{d}_i$,

(a) *if* $2D_2 < (1 - \varepsilon)D_1$ *then*

$$\Pr(F \text{ is satisfiable}) \to 1, \text{ and}$$

(b) *if* $2D_2 > (1 + \varepsilon)D_1$ *then*

$$\Pr(F \text{ is unsatisfiable}) \to 1.$$

In the case of $m = cn$ randomly chosen clauses, $D_1 = 2cn$ and w.h.p. $D_2 \approx c^2 n$, giving the familiar 2-SAT phase transition of [7].

The "satisfiable" half of the theorem is relatively easy to prove. The subcriticality of the branching process means that there are unlikely to be long paths (chains of implications), and there are likely to be no occurrences of a related structure, a "bicycle", shown by Chvátal and Reed [7] to be a necessary condition for unsatisfiability.

The "unsatisfiable" half is a bit harder. It is argued that a complementary pair of literals A, $\bar{A}$ can be found such that both $span(A)$ and $span(\bar{A})$ (the sets of implications of setting each True) are "large". Further, w.h.p., large spans contain complementary literals; that is, w.h.p. $A \to X$ and $A \to \bar{X}$, while $\bar{A} \to Y$ and $\bar{A} \to \bar{Y}$. If this is so, the formula is unsatisfiable.

4 Conclusions

A certain amount is known about random k-SAT formulas. Sparse formulas are almost all satisfiable, dense ones are almost all unsatisfiable, and there is a sharp threshold somewhere in between. The case of 2-SAT is understood well: it is known exactly where the threshold lies and how sharp it is. The 2-SAT threshold point can be generalized to random formulas with a specified degree sequence.

But much less is known for $k > 2$. Here it is not even known if the threshold density has a limit as $n \to \infty$, although it is expected that it does, for all k, and that for 3-SAT the limit is about 4.2. Although it is likely that both the known 3-SAT satisfiability bound of 3.26 and unsatisfiability bound of 4.596 can be improved, different methods will be needed to place either of them at the true threshold value. For $k > 3$, the methods of [2] no longer suffice to tune myopic satisfiability algorithms.

Nothing is known about relaxing the uniform randomness assumption for random 3-SAT formulas, and since in a 3-clause no variable's setting has any direct implications, generalizing the result of [8] is not straightforward.

A broader question is whether, as [6] suggests, random unsatisfiable formulas are (on average) truly hard to prove to be unsatisfiable, and — on the other side — whether stronger myopic algorithms suffice to solve random formulas in the satisfiable regime, whether other polynomial-time algorithms are required, or whether denser formulas within the satisfiable regime are hard.

References

1. Dimitris Achlioptas. Setting two variables at a time yields a new lower bound for random 3-SAT. In *32nd Annual ACM Symposium on Theory of Computing (Portland, OR, 2000)*, pages 28–37. ACM, New York, 2000.
2. Dimitris Achlioptas and Gregory B. Sorkin. Optimal myopic algorithms for random 3-SAT. In *41st Annual Symposium on Foundations of Computer Science*, pages 590–600. IEEE Comput. Soc. Press, Los Alamitos, CA, 2000.

3. Béla Bollobás, Christian Borgs, Jennifer Chayes, Jeong Han Kim, and David B. Wilson. The scaling window of the 2-SAT transition. Manuscript, 1999.
4. Andrei Z. Broder, Alan M. Frieze, and Eli Upfal. On the satisfiability and maximum satisfiability of random 3-CNF formulas. In *4th Annual ACM-SIAM Symp. on Disc. Alg. (Austin, TX, 1993)*, pages 322–330. ACM, New York, 1993.
5. Ming-Te Chao and John Franco. Probabilistic analysis of two heuristics for the 3-satisfiability problem. *SIAM J. Comput.*, 15(4):1106–1118, 1986.
6. Vašek Chvátal and Endre Szemerédi. Many hard examples for resolution. *J. Assoc. Comput. Mach.*, 35(4):759–768, 1988.
7. Vasēk Chvátal and Bruce Reed. Mick gets some (the odds are on his side). In *33th Annual Symposium on Foundations of Computer Science (Pittsburgh, PA, 1992)*, pages 620–627. IEEE Comput. Soc. Press, Los Alamitos, CA, 1992.
8. Colin Cooper, Alan Frieze, and Gregory B. Sorkin. A note on random 2-SAT with prescribed literal degrees. In *Proceedings of the 13th Annual ACM–SIAM Symposium on Discrete Algorithms*. ACM, New York, 2002.
9. Wenceslas Fernandez de la Vega. On random 2-SAT. Manuscript, 1992.
10. Ehud Friedgut. Necessary and sufficient conditions for sharp thresholds of graph properties, and the k-SAT problem. *J. Amer. Math. Soc.*, 12:1017–1054, 1999.
11. Joel Friedman and Andreas Goerdt. Recognizing more unsatisfiable random 3-SAT instances efficiently. manuscript.
12. Andreas Goerdt. A threshold for unsatisfiability. *J. Comput. System Sci.*, 53(3):469–486, 1996.
13. Svante Janson, Yiannis C. Stamatiou, and Malvina Vamvakari. Bounding the unsatisfiability threshold of random 3-SAT. *Random Structures Algorithms*, 17(2):103–116, 2000.
14. Thomas G. Kurtz. Solutions of ordinary differential equations as limits of pure jump Markov processes. *J. Appl. Probability*, 7:49–58, 1970.
15. Mike Molloy and Bruce Reed. The size of the giant component of a random graph with a given degree sequence. *Combinatorics, Probability and Computing*, 7:295–305, 1998.
16. Rémi Monasson and Riccardo Zecchina. Entropy of the K-satisfiability problem. *Phys. Rev. Lett.*, 76(21):3881–3885, 1996.
17. Rémi Monasson and Riccardo Zecchina. Statistical mechanics of the random K-satisfiability model. *Phys. Rev. E (3)*, 56(2):1357–1370, 1997.
18. Rémi Monasson and Riccardo Zecchina. Tricritical points in random combinatorics: the $(2+p)$-SAT case. *J. Phys. A: Math. and Gen.*, 31(46):9209–9217, 1998.
19. Nicholas C. Wormald. Differential equations for random processes and random graphs. *Ann. Appl. Probab.*, 5(4):1217–1235, 1995.

Prospects for Simulated Annealing Algorithms in Automatic Differentiation

Uwe Naumann and Peter Gottschling

Department of Computer Science, University of Hertfordshire, Hatfield, UK
u.1.naumann@herts.ac.uk

Abstract. We present new ideas on how to make simulated annealing applicable to the combinatorial optimization problem of accumulating a Jacobian matrix of a given vector function using the minimal number of arithmetic operations. Building on vertex elimination in computational graphs we describe how simulated annealing can be used to find good approximations to the solution of this problem at a reasonable cost.

Keywords: Accumulation of Jacobian matrices, vertex elimination, simulated annealing, logarithmic cooling schedule, inhomogeneous Markov chains.

1 Problem Description

The requirement to compute accurate derivative information for mathematical models efficiently is central to many scientific, economical, and engineering problems. Without it the highly desirable step from simulation to optimization can often not be made. Especially first order derivatives of vector functions given as computer programs play an important role in modern numerical analysis.

Let $F' = F'(\mathbf{x}_0) = \left(\frac{\partial y_i}{\partial x_j}(\mathbf{x}_0)\right)_{j=1,\ldots,n}^{i=1,\ldots,m}$ denote the Jacobian matrix of a nonlinear vector function $F : I\!R^n \supseteq D \to I\!R^m : \mathbf{x} \mapsto \mathbf{y} = F(\mathbf{x})$ evaluated at some argument $\mathbf{x}_0$. Automatic Differentiation (AD) [7], [3], [10], [6] enables us to compute F' numerically with machine accuracy. F is assumed to be given as a computer program which decomposes into a sequence of scalar elemental functions $(I\!R \ni) v_j = \varphi_j(v_i)_{i \prec j}$ where $j = 1, \ldots, q$ and $p = q - m$. $i \prec j$ denotes the direct dependence of v_j on v_i. We write $i \prec^* j$ if there exist $k_1, \ldots, k_p$ such that $i \prec k_1 \prec k_2 \prec \ldots \prec k_p \prec j$. So, $\{i | i \prec j\}$ is the index set of arguments of φ_j and we denote its cardinality by $|\{i | i \prec j\}|$. Within F we distinguish between three types of variables $V = X \cup Z \cup Y$, independent ($X \equiv \{v_{1-n}, \ldots, v_0\}$), intermediate ($Z \equiv \{v_1, \ldots, v_p\}$), and dependent ($Y \equiv \{v_{p+1}, \ldots, v_q\}$). We set $x_i \equiv v_{i-n}$, $i = 1, \ldots, n$, and $y_j \equiv v_{p+j}$, $j = 1, \ldots, m$. The numbering $\mathcal{I} : V \to \{(1-n), \ldots, q\}$ of the variables of F is expected to induce a topological order with respect to the dependence "$\prec$", i.e. $i \prec^* j \Rightarrow \mathcal{I}(v_i) < \mathcal{I}(v_j)$.
Since the differentiation of F is based on the differentiability of its elemental functions it will be assumed that the φ_j, $j = 1, \ldots, q$, have jointly continuous partial derivatives $c_{ji} \equiv \frac{\partial}{\partial v_i}\varphi_j(v_k)_{k \prec j}$, $i \prec j$, on open neighborhoods $\mathcal{D}_j \subset I\!R^{n_j}$,

K. Steinhöfel (Ed.): SAGA 2001, LNCS 2264, pp. 131–144, 2001.

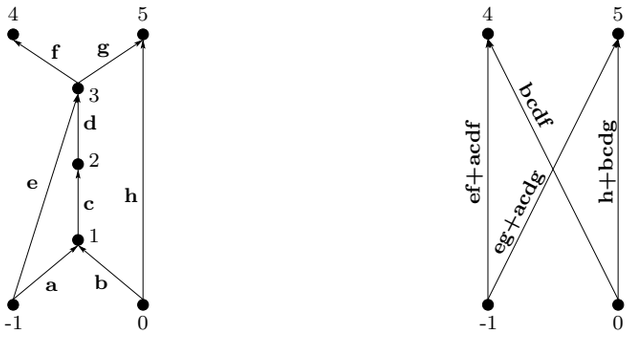

Fig. 1. G and **G**′

$n_j \equiv |\{i|i \prec j\}|$, of their domain. The computational graph (or c-graph) $\mathbf{G} = (V, E)$ of F is a directed acyclic graph with $V = \{i|v_i \in F\}$ and $(i, j) \in E$ if $i \prec j$. We assume $\mathbf{G}$ to be linearized in the sense that the numerical values of all partial derivatives of the elemental functions are attached to their corresponding edges, i.e. (i, j) is labelled with c_{ji}. Fig. 1 shows an example of a c-graph with $n = 2$, $p = 3$, and $m = 2$. The local partial derivatives $c_{j,i}$, $i \prec j$, are denoted by $\mathbf{a}, \ldots, \mathbf{h}$. Assuming, for example, that $v_1 = v_{-1} \cdot v_0$ and $v_4 = \sin(v_3)$ we get

$$\mathbf{a} = \frac{\partial(v_{-1} \cdot v_0)}{\partial v_{-1}} = v_0 \qquad \text{and} \qquad \mathbf{f} = \frac{\partial(\sin(v_3))}{\partial v_3} = \cos(v_3).$$

The accumulation of F' can be regarded as the process of transforming $\mathbf{G}$ into a subgraph $\mathbf{G}'$ of the complete bi-partite graph $K_{n,m}$ [11]. This transformation will be denoted by $\mathbf{G} \to \mathbf{G}'$. Different application sequences of the *chain rule* to $\mathbf{G}$ may yield drastically varying computational costs.

By the chain rule an entry $F'(i, j)$ of the Jacobian can be computed by multiplying the edge labels over all paths connecting the minimal vertex j with the maximal vertex i followed by building the sum over all these products [16]. For example, the Jacobian of the c-graph in Fig. 1 is given by

$$F' = \begin{pmatrix} \mathbf{ef} + \mathbf{acdf} & \mathbf{bcdf} \\ \mathbf{eg} + \mathbf{acdg} & \mathbf{h} + \mathbf{bcdg} \end{pmatrix},$$

which is equivalent to $\mathbf{G}' = K_{2,2}$ as shown in Fig. 1 on the right. Assuming that modern floating-point units can compute an addition on top of a multiplication at virtually no extra cost [14] we will take the number of multiplications as the objective function to be minimized. The resulting combinatorial optimization problem is called Optimal Jacobian Accumulation (OJA) problem [22] and it is conjectured to be NP-complete [11], [5]. This conjecture is based on similarities between OJA and the Gaussian elimination problem considered in [27]. Herley [13] could show that the closely related Fill-in problem is NP-complete [9] under vertex elimination. The proofs of corresponding results for *edge* and *face* elimination [24] could not be given so far.

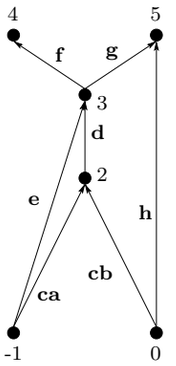

Fig. 2. Elimination of vertex 1

The successive elimination of all (vertex-)paths of length > 2 in the above example would take 14 multiplications. An optimal application sequence of the chain rule would give us F' at the cost of only 7 multiplications. Simply pre-accumulate $r = \mathbf{cd}$, $s = \mathbf{a}r + \mathbf{e}$, and $t = \mathbf{b}r$ and compute

$$F' = \begin{pmatrix} s\mathbf{f} & t\mathbf{f} \\ s\mathbf{g}\ t\mathbf{g} + \mathbf{h} \end{pmatrix}.$$

In fact, this optimal way of computing F' is equivalent to the vertex elimination sequence $(2, 1, 3)$ in $\mathbf{G}$. Vertex elimination will be introduced in the following section.

2 Simulated Annealing

2.1 Configuration

Our discussion will be based on vertex elimination which was first used in [11] to optimize the accumulation of the Jacobian by applying the *Markowitz rule* adapted from the theory on the solution of sparse linear systems.

Rule 1 *To eliminate a vertex j from $\mathbf{G}$ we perform for all $i \prec j$ and all k with $j \prec k$ the following steps:*

1. *Compute $d = c_{k,j} \cdot c_{j,i}$.*
2. *If $(i, k) \in \mathbf{G}$ then*

   – *set $c_{k,i} = c_{k,i} + d$*

   else

   – *introduce a new edge $(i, k) \in \mathbf{G}$;*
   – *label it with d.*

A graphical illustration of Rule 1 can be found in Fig. 2 which shows $\mathbf{G}$ after the elimination of vertex 1. Assuming that $\forall i,j \in Y : i \not\prec j \wedge j \not\prec i$ and $\forall i,j \in X : i \not\prec j \wedge j \not\prec i$ we get the following

Proposition 1 *The transformation $\mathbf{G} \to \mathbf{G}'$ can be regarded as the elimination of the p intermediate vertices in $\mathbf{G}$ using Rule 1.*

PROOF: The proof of Prop. 1 follows immediately from the chain rule and it can be found in [11]. □

Our configuration space is the set of all feasible vertex elimination sequences in $\mathbf{G}$. It will be denoted by $\mathcal{V}$ and it is finite with $|\mathcal{V}| = p!$. A vertex elimination sequence σ is a permutation of the indices of the intermediate vertices in $\mathbf{G}$, i.e.

$$\sigma = (\pi_1, \ldots, \pi_p), \quad \text{where } \forall i,j, i \neq j : 1 \leqslant \pi_i \leqslant p, 1 \leqslant \pi_j \leqslant p, \pi_i \neq \pi_j.$$

According to Rule 1 the elimination of a vertex $j \in Z$ involves $|P_j^\sigma| \cdot |S_j^\sigma|$ scalar multiplications. $|P_j^\sigma|$ and $|S_j^\sigma|$ denote the numbers of predecessors and successors of j, respectively. Both values depend on the order within the current elimination sequence σ. We aim to minimize the objective function

$$\mathcal{C} = \mathcal{C}(\sigma) \equiv \sum_{j=1}^{p} |P_{\pi_j}^\sigma| \cdot |S_{\pi_j}^\sigma|$$

over $\mathcal{V}$. Notice, that $\mathcal{C} \in I\!N$ for all $\sigma \in \mathcal{V}$. The associated combinatorial optimization problem will be referred to as the VERTEX ELIMINATION (VE) problem. The set of optimal vertex elimination sequences is defined by

$$\mathcal{V}_{\mathtt{min}} = \{\sigma | \forall \sigma' \in \mathcal{V} : \mathcal{C}(\sigma) \leqslant \mathcal{C}(\sigma')\}.$$

Proposition 2 *The complexity of evaluating $\mathcal{C}$ over $\mathcal{V}$ is $O(p^3)$.*

PROOF: Without loss of generality, let $m = n$ and $k = 2 \cdot n + p$ with p odd. Considering the most expensive vertex elimination sequence possible we observe that the overall cost adds up to

$$\left(\frac{k-1}{2}\right)^2 + \frac{k-1}{2} \cdot \left(\frac{k-1}{2} - 1\right) + \left(\frac{k-1}{2} - 1\right)^2 + \ldots + n^2 = O(p^3)$$

for a given n. □ Obviously, the evaluation of $\mathcal{C}$ is polynomial in n and m too.

VE is not equivalent to OJA [19]. There are cases where the optimal vertex elimination sequence does not minimize the number of scalar multiplications required to compute F'. However, building on a large number of numerical tests [19], we conjecture that this *vertex discrepancy* does not exceed a small constant factor. This makes vertex elimination a sensible approach to approximating the solution of OJA. Refer to Sec. 4 for further remarks.

2.2 Neighborhood Relations

The effect of simulated annealing on the solution of VE has been investigated first in [23] by combining rearrangements suggested in [15] with various cooling schedules. The corresponding algorithms were applied to a selection of widely used test problems in numerical simulation and optimization showing very promising results. In this paper we will propose new ideas on how to make logarithmic cooling schedules for inhomogeneous Markov chains [12] work on vertex elimination sequences in c-graphs. Furthermore, the results of this approach will be compared with standard pre-accumulation techniques [11] as well as with a simple fixed-temperature Metropolis [17] algorithm for an a priori fixed stopping time.

Neighborhood relations on $\mathcal{V}$ are defined by rearrangements transforming $\sigma \in \mathcal{V}$ into $\sigma' \in \mathcal{N}_\sigma \subseteq \mathcal{V}$ where $\mathcal{N}_\sigma$ denotes the neighborhood of σ. These *transitions* will be denoted by $[\sigma, \sigma']$ and the probability of generating σ' from σ via a corresponding sequence of transitions by $G[\sigma, \sigma']$. For simplicity, transitions will be defined such that

$$G[\sigma, \sigma'] = \frac{1}{|\mathcal{N}_\sigma|}. \tag{1}$$

For this purpose we will denote a uniformly distributed random number over the closed interval $[1, p] \subset \mathbb{N}$ by $r \in \mathcal{R}([1, p])$. Acceptance probabilities are associated with all feasible transitions $[\sigma, \sigma']$. They are defined by

$$A[\sigma, \sigma'] = \begin{cases} 1, & \text{if } \mathcal{C}(\sigma') \leqslant \mathcal{C}(\sigma), \\ e^{-\frac{\mathcal{C}(\sigma') - \mathcal{C}(\sigma)}{t}}, & \text{otherwise,} \end{cases} \tag{2}$$

where the control parameter t can be interpreted as the *temperature* in the annealing process [17], [29]. The probability of a transition $[\sigma, \sigma']$ actually being executed is

$$\mathcal{P}[\sigma, \sigma'] = \begin{cases} G[\sigma, \sigma'] \cdot A[\sigma, \sigma'], & \text{if } \sigma \neq \sigma', \\ 1 - \sum_{\sigma' \neq \sigma} G[\sigma, \sigma'] \cdot A[\sigma, \sigma'], & \text{otherwise.} \end{cases} \tag{3}$$

Building on (3) one can define the probability of being in a configuration σ after k steps:

$$p_\sigma(k) = \sum_{\sigma'} p_{\sigma'}(k-1) \cdot \mathcal{P}[\sigma', \sigma]. \tag{4}$$

We will consider inhomogeneous Markov chains defined by (4) where the temperature t is lowered in (2) at each single step. First of all we need to define suitable neighborhood relations as, for example, the following three:

Neighborhood 1 (N1) *The transition* $[\sigma, \sigma']$ *is defined by* $r, r' \in \mathcal{R}([1, p])$, $r < r'$, *such that*

$$\sigma = (\pi_1, \ldots, \pi_{r-1}, \pi_r, \pi_{r+1}, \ldots, \pi_{r'-1}, \pi_{r'}, \pi_{r'+1}, \ldots, \pi_p),$$
$$\sigma' = (\pi_1, \ldots, \pi_{r-1}, \pi_{r'}, \pi_{r'-1}, \ldots, \pi_{r+1}, \pi_r, \pi_{r'+1}, \ldots, \pi_p).$$

σ' is derived from σ by reversing the subsequence $(\pi_r, \ldots, \pi_{r'})$.

Neighborhood 2 (N2) *The transition* $[\sigma, \sigma']$ *is defined by* $r, r' \in \mathcal{R}([1,p])$, $r < r'$, *such that*

$$\sigma = (\pi_1, \ldots, \pi_{r-1}, \pi_r, \pi_{r+1}, \ldots, \pi_{r'-1}, \pi_{r'}, \pi_{r'+1}, \ldots, \pi_p),$$
$$\sigma' = (\pi_1, \ldots, \pi_{r-1}, \pi'_r, \pi_{r+1}, \ldots, \pi_{r'-1}, \pi_r, \pi_{r'+1}, \ldots, \pi_p).$$

The two elements π_r and $\pi_{r'}$ are exchanged in σ to get σ'.

Neighborhood 3 (N3) *The transition* $[\sigma, \sigma']$ *is defined by* $r \in \mathcal{R}([1,p])$ *as*

$$\sigma = (\pi_1, \ldots, \pi_r, \ldots, \pi_p),$$
$$\sigma' = \begin{cases} (\pi_1, \ldots, \pi_{r+1}, \pi_r, \pi_{r+2}, \ldots, \pi_p), & \textit{if } r < p, \\ (\pi_p, \pi_2, \ldots, \pi_{p-1}, \pi_1), & \textit{if } r = p. \end{cases}$$

Two neighboring vertices are exchanged in σ to give σ'. The neighborhood is made cyclic by exchanging π_1 and π_p if $r = p$. If σ' can be obtained from σ by some sequence of given transitions then we write $\sigma' \in \mathcal{N}^*_\sigma$.

In order to evaluate the computational complexity of each of the moves one should notice that the structure of the c-graph is modified by vertex elimination. This implies that either any new elimination sequence has to be run "from scratch" to evaluate its cost or all intermediate versions of the graph generated by the successive elimination of vertices have to be stored. For **N2** the latter approach would enable us to start the evaluation of σ' with the graph resulting from the elimination of $(\pi_1, \ldots, \pi_{r-1})$ from $\mathbf{G}$ at the cost of a significant increase in memory requirement. We have decided to pursue the first strategy resulting in a computational complexity regarding the number of scalar floating-point multiplications performed which is quadratic in the number $|V|$ of vertices in $\mathbf{G}$. In fact, the worst case complexity is $\binom{|V|-1}{2}$ for the complete graph $K_{|V|}$.

We will show that **N2** satisfies all requirements for logarithmic simulated annealing as described in [12]. It is straight forward to show similar results for **N1** and **N3**. First of all **N2** is required to be *reversible*.

Proposition 3

$$\forall \sigma, \sigma' \in \mathcal{V} : \sigma' \in \mathcal{N}_\sigma \Leftrightarrow \sigma \in \mathcal{N}_{\sigma'}.$$

Proof: This result is obvious and follows immediately from the definition of **N2**. □

Knowing that our transitions are reversible we need to show that there exist sequences of transitions transforming σ into σ' for all pairs of configurations $\sigma, \sigma' \in \mathcal{V}$.

Proposition 4

$$\forall \sigma, \sigma' \in \mathcal{V}\ \exists \sigma_0, \ldots, \sigma_k \in \mathcal{V}\ (\sigma_0 = \sigma \wedge \sigma_k = \sigma') : G[\sigma_i, \sigma_{i+1}] > 0, i = 0, \ldots, (k-1).$$

PROOF: By (1) $G[\sigma_i, \sigma_j] > 0$ for all $\sigma_i, \sigma_j \in \mathcal{V}$ with $\sigma_j \in \mathcal{N}_{\sigma_i}$. Furthermore, by exchanging two vertices at a time any permutation of the p intermediate vertices can be obtained, i.e. $\forall \sigma \in \mathcal{V} : \mathcal{N}_\sigma^* = \mathcal{V}$. □ An immediate consequence of Prop. 3 and Prop. 4 is that **N2** is *identically reversible*, i.e. if $\sigma_0, \ldots, \sigma_k \in \mathcal{V}$ are any elimination sequences as in Prop. 4 to obtain σ' from σ then σ can be obtained from σ' via $\sigma' = \sigma_k, \ldots, \sigma_0 = \sigma$.

Definition 1. *$\sigma' \in \mathcal{V}$ is reachable at height h from $\sigma \in \mathcal{V}$ if $\exists \sigma_0, \ldots, \sigma_k \in \mathcal{V}$ $(\sigma_0 = \sigma \wedge \sigma_k = \sigma')$ such that $G[\sigma_i, \sigma_{i+1}] > 0, i = 0, \ldots, (k-1)$, and $\mathcal{C}(\sigma_i) \leqslant h$ for all $i = 0, \ldots, k$.*

The height of a sequence of transitions transforming σ into σ' will be noted by $\mathcal{H}[\sigma, \sigma']$.

Proposition 5

$$\forall h : \mathcal{H}[\sigma, \sigma'] \leqslant h \Leftrightarrow \mathcal{H}[\sigma', \sigma] \leqslant h.$$

PROOF: With $G[\sigma_i, \sigma_j] > 0$ for all $\sigma_i, \sigma_j \in \mathcal{V}$ this is a direct consequence of **N2** being identically reversible. □

Definition 2. *Let $\sigma_{\min}$ denote a local minimum, i.e. $\sigma_{\min} \in \mathcal{V} \setminus \mathcal{V}_{\min}$ and $\mathcal{C}(\sigma_{\min}) < \mathcal{C}(\sigma)$ for all $\sigma \in \mathcal{N}_{\sigma_{\min}} \setminus \sigma_{\min}$. The escape depth $\mathcal{D}(\sigma_{\min})$ of $\sigma_{\min}$ is the smallest Δh such that there exists a $\sigma' \in \mathcal{V}$ where $\mathcal{C}(\sigma') < \mathcal{C}(\sigma_{\min})$ which is reachable at height $\mathcal{C}(\sigma_{\min}) + \Delta h$.*

According to HAJEK's result [12] asymptotic convergence of the logarithmic simulated annealing algorithm is guaranteed if the temperature in (2) is chosen as

$$t = t(k) = \frac{\Gamma}{\ln(k+2)}, \quad \text{for } k = 0, 1, \ldots,$$

with $\Gamma \geqslant \max_{\sigma_{\min}} \mathcal{D}(\sigma_{\min})$. The detailed investigation of the energy landscape of VE is subject to ongoing research (see Sec. 4). The main outcome of this should be a better understanding of how to choose Γ appropriately. In the following section we will present some preliminary test results obtained by implementing various cooling schedules based on heuristic choices for Γ.

3 Numerical Results

The first test function was introduced in [19] and referred to as *absorption function*. It is given as $y = f(x)$ with $f : \mathbb{R}^n \to \mathbb{R}$ defined by

$$y = \prod_{i=0}^{n-1} \left(\varphi_i(x_i) \cdot \prod_{j=1}^{e_i} x_i \right) \tag{5}$$

where $\varphi_i : \mathbb{R} \to \mathbb{R}$ and $e_i \geqslant 1$ for $i = 0, \ldots, (n-1)$. Fig. 3 shows an instance of (5). Our discussion will be based on c-graphs $\mathbf{G}$ of this shape with variable parameters n and $e_i, i = 0, \ldots, (n-1)$. We will consider two special cases defined by

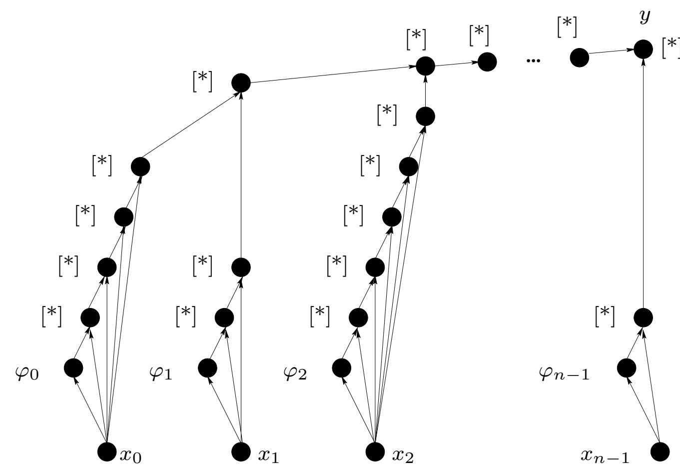

Fig. 3. Absorption Function

$e_i = i + 1, i = 0, \ldots, 9$, (**ABS1**) and $e_i = 3$, $i = 0, \ldots, 29$, (**ABS2**) respectively. One interesting property of (5) with respect to VE is that its optimal solution is a well-defined combination of forward and reverse vertex elimination applied to certain parts of **G**. In fact, the products $w_i(e_i) = \varphi_i \cdot x_i \cdot \ldots \cdot x_i, i = 0, \ldots, (n-1)$, have to be eliminated in forward mode followed by the elimination of the intermediate vertices of $w_1 \cdot w_2 \cdot \ldots \cdot w_{n-1}$ in reverse mode. This leaves us with n vertices (representing the $w_i, i = 0, \ldots, (n-1)$) to be eliminated at the cost of one multiplication each. Consequently, we can immediately provide expressions for the cost of running a forward vertex elimination sequence (**VFM**: $n + \sum_{i=0}^{n-1} e_i + \binom{n}{2} - 1$), a reverse vertex elimination sequence (**VRM**: $2 \cdot (n-2) + 2 \cdot \sum_{i=0}^{n-1} e_i + n$), and an optimal vertex elimination sequence (**VOPT**: $n + \sum_{i=0}^{n-1} e_i + 2 \cdot (n-2)$). For **ABS1** we get the values 109, 136, and 81, respectively. **ABS2** results in 554, 266, and 176.

Our simulated annealing algorithm starts with the worst choice out of **VFM** and **VRM**. Of course, we could as well have chosen a random elimination sequence as starting point. In any case we do not want the cost of this initial sequence to be close to the minimum already. By picking the maximum out of **VFM** and **VRM** we can satisfy this requirement in most cases. Additionally, min{**VFM**, **VRM**} often gives us a useful reference value for comparison with the result delivered by the annealing process. (See, for example, annealing of Chebychev Quadrature problem [1] in [23].)

We have combined each of the three neighborhood relations **N1**, **N2**, and **N3** with a cooling schedule defined by $\Gamma = 2$ leading to simulated annealing algorithms **SAV1**, **SAV2**, and **SAV3**. Fig. 4 and Fig. 5 show the development of the objective function over the first one thousand iterations for **ABS1** and **ABS2**, respectively. It took approximately five seconds on an INTEL Pentium-III 233 MHz system to complete one annealing process. This runtime could be

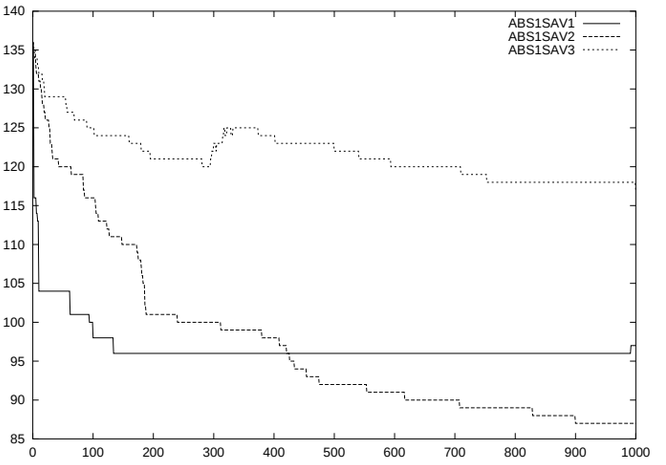

Fig. 4. Simulated annealing of ABS1

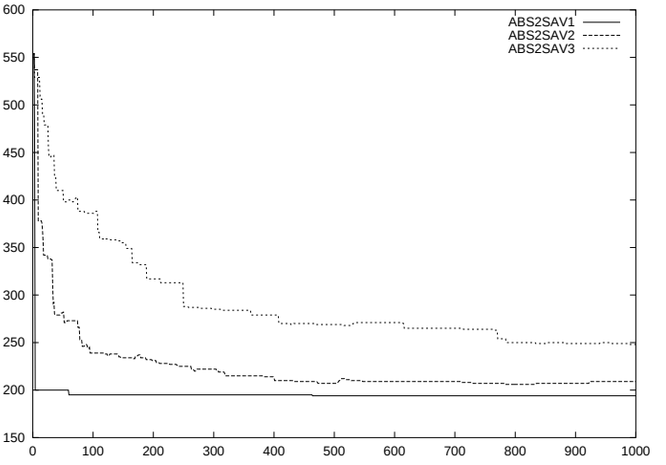

Fig. 5. Simulated annealing of ABS2

respectable even when thinking of simulated annealing as a compiler optimization pass (see Sec. 4). No significant improvements were achieved by increasing the number of iterations to five or even ten thousand. While **N1** and **N2** seem to perform well in both cases **N3** results in a slower convergence, especially for **ABS1**. This suggests that it may either be a poor neighborhood choice or that the cooling was in fact too quick. In order to investigate this problem further we

have applied a slower cooling schedule with

$$t = t(k) = \frac{\Gamma}{\ln(\sqrt{k+1})}, \quad \Gamma = 2, \;\; k = 0, \ldots, 5000,$$

to **ABS1**. After a considerably increased runtime of approximately 30 seconds the algorithm returned 108. This result does not represent a significant improvement. The values delivered by **SAV1** and **SAV2** were not reached. We conclude that simulated annealing based on **N3** in combination with logarithmic cooling schedules for inhomogeneous Markov chains is less suitable for our purposes.

Research in the field of optimizing the accumulation of Jacobian matrices by elimination techniques in c-graphs [24] has only recently been starting to deliver both theoretical and numerical results. First attempts to use vertex elimination in connection with the greedy Markowitz heuristic have lead to first numerical results published in [11]. However, most results suitable for comparison with the simulated annealing approach were presented in [19]. Standard techniques of AD such as forward and reverse vector modes [10], seed matrix compression techniques [8], [25], or sparse forward and reverse modes [4] are all aimed towards the minimization of the computational effort for accumulating derivatives. However, in virtually all cases their performance can not keep up with elimination techniques, provided that the latter are applicable (see Sec. 4). The following table shows how the simulated annealing algorithms compare with **VFM**, **VRM**, and with the known optimum. The cost of the start configuration and the minimum achieved by simulated annealing are highlighted.

	VFM	**VRM**	**SAV1**	**SAV2**	**SAV3**	**VOPT**
ABS1	109	**136**	96	**87**	117	81
ABS2	**554**	266	**194**	206	248	176

The logarithmic cooling schedule was introduced because it is possible to prove its asymptotic convergence [12]. However, the schedule is widely regarded as impractical. Often comparable or even better results can be obtained by a simple fixed-temperature Metropolis algorithm. We have implemented the latter for various temperatures t while fixing the number of iterations to be performed (5000 in this case). The algorithm was applied to **ABS1** and the development of the objective function is illustrated in Fig. 6. For $t = 0.5$ the algorithm does not seem to converge. "By luck" the case $t = 0.4$ produces a good elimination sequence early in the annealing process. However, it diverges from this solution soon due to the acceptance of worse sequences as the algorithm proceeded. The variant where $t = 0.1$ appears to be too restrictive in the acceptance of new elimination sequences resulting in a slower convergence. Both $t = 0.2$ and $t = 0.3$ turn out to be good choices for the temperature. Again, the algorithm with $t = 0.2$ generates a very good sequence early in the annealing process which results in a slightly better final solution. These results correspond to the ones obtained using logarithmic cooling schedules. Fig. 7 shows the development of the temperature over the first one thousand iterations with $\Gamma = 2$. Most of the time the algorithm runs at a temperature between 0.35 and 0.29.

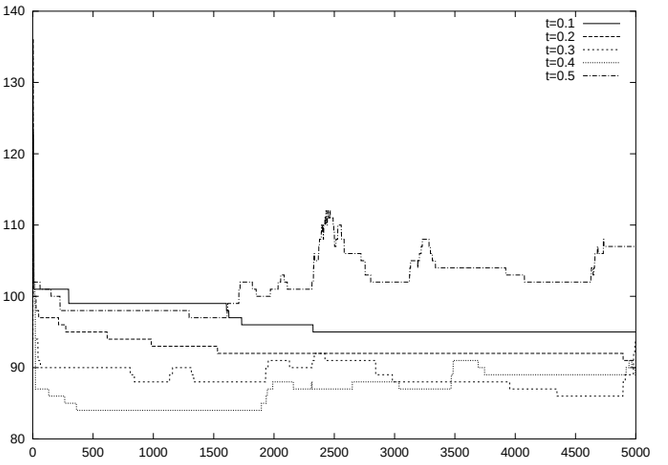

Fig. 6. Fixed-temperature Metropolis Algorithm for ABS1

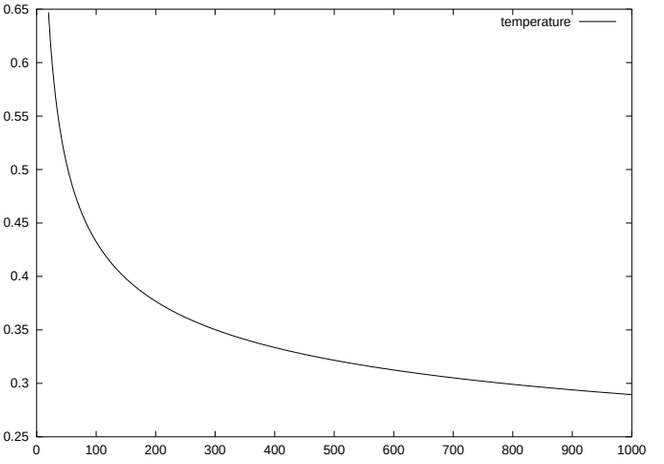

Fig. 7. Development of Temperature for $\Gamma = 2$

In order to test the robustness of **N2** in a less predictable environment we have generated four c-graphs at random with the following specifications: **RG1** ($n = 1$, $p = 100$, $m = 10$, $i_{\texttt{min}} = 2$, and $i_{\texttt{max}} = 4$), **RG2** ($n = 10$, $p = 100$, $m = 10$, $i_{\texttt{min}} = 2$, and $i_{\texttt{max}} = 2$), **RG3** ($n = 20$, $p = 100$, $m = 5$, $i_{\texttt{min}} = 2$, and $i_{\texttt{max}} = 3$), **RG4** ($n = 5$, $p = 200$, $m = 5$, $i_{\texttt{min}} = 2$, and $i_{\texttt{max}} = 3$), where $i_{\texttt{min}}$ and $i_{\texttt{max}}$ denote the minimal and maximal numbers of successors of any independent and intermediate vertex in the c-graph, respectively. The same logarithmic cooling schedule with $\Gamma = 2$ was applied to all four problems for one thousand steps and a start configuration **START** which, again, was chosen as the worst out of **VFM**

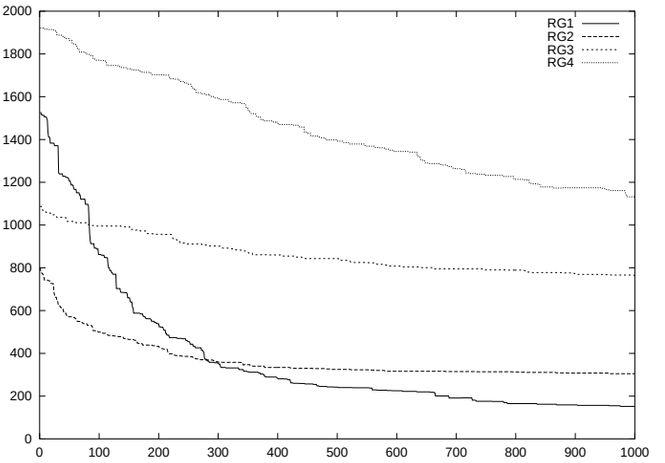

Fig. 8. Simulated annealing of randomly generated c-graphs

and **VRM**. The minima obtained by simulated annealing (**VSA**) for **RG2** and **RG3** lay already below the best values achieved by other known algorithms [20], [21] (**VMIN**) and are highlighted in the table below. The development of the value of the objective function is illustrated in Fig. 8.

	START	**VSA**	**VMIN**
RG1	1526	152	95
RG2	810	**305**	344
RG3	1087	**765**	865
RG4	1925	1132	529

4 Outlook

The generation of optimized derivative code for mathematical models given as computer programs is considered as a compile-time activity. Thinking of differentiation enabled compilers [18] the runtime of algorithms for optimizing elimination sequences represents an important factor. In this situation simulated annealing is likely to come into play only if the runtime of the algorithm can be fixed a priori as in Sec. 3 or if the generated code will be used repeatedly over a long period of time. This is when the effort of experimenting with different neighborhood relations and cooling schedules might pay off in form of a highly efficient derivative code.

In order to be able to apply elimination techniques the c-graph of the function has to be built at compile time. In general, computer programs contain branches and loops with variable bounds so that this requirement can not be met. However, c-graphs of *basic blocks* [2] can often be obtained easily unless

effects such as *aliasing* [2] occur. The exploitation of the theoretical results in this field is subject to ongoing research and development [28].

So is the theoretical investigation of OJA, in general. In [22] we could show that vertex elimination is not general enough to yield the solution of OJA for all c-graphs. Neither is edge elimination as introduced in [19]. A quantitative characterization of the *vertex* and *edge discrepancies* remains one of the unsolved problems in AD. We are currently trying to develop simulated annealing algorithms for both *edge* and *face elimination* as defined in [24]. So far, both problems turned out to be much harder to handle than pure vertex elimination. While vertex elimination sequences are simply permutations of the intermediate vertices both edge and face elimination sequences change dynamically due to new edges being introduced (*fill-in*).

An important part of our research is dedicated to the development of robust and efficient algorithms (both sequential and parallel) for finding nearly optimal elimination sequences. Colleagues from the University of Cranfield / Royal Military College of Sciences, Shrivenham, UK, are developing a compiler infrastructure which will allow us to parse real world application programs into internal representations suitable for elimination techniques. A quasi-standard interface between the compiler and our optimization module is being developed to allow for easy communication of c-graphs and elimination sequences [26].

Finally, we are working towards a comprehensive complexity analysis of the various elimination problems. Herley [13] could show that the problem of minimizing the fill-in is NP-complete under vertex elimination. The closely related VE is conjectured to exhibit the same property. Further research is required to prove analogous results for more general elimination techniques as introduced in [24].

Acknowledgement

This work is supported by the Engineering and Physical Sciences Research Council (EPSRC) under grant number GR/R/38101/01.

References

1. B. Averik and R. Carter and J. More, *The Minpack-2 Test Problem Collection (Preliminary Version)*, Mathematical and Computer Science Division, Argonne National Laboratory, Technical Report 150, 1991.
2. A. Aho, R. Sethi, and J. Ullman, *Compilers. Principles, Techniques, and Tools*, Addison Wesley, 1986.
3. M. Berz, C. Bischof, G. Corliss, and A. Griewank, eds., *Computational Differentiation: Techniques, Applications, and Tools*, SIAM, 1996.
4. C. Bischof and A. Carle and P. Khademi and A. Maurer, *The ADIFOR 2.0 System for Automatic Differentiation of Fortran 77 Programs*, IEEE Comp. Sci. & Eng., 3, 3, 1996, 18–32.
5. C. Bischof and M. Haghighat, *Hierarchical Approaches to Automatic Differentiation*, in [3], 83–94.

6. G. Corliss and C. Faure and A. Griewank and L. Hascoet and U. Naumann, eds., *Automatic Differentiation - From Simulation to Optimization*, LNCS, Springer, 2001, *to appear*.
7. G. Corliss and A. Griewank, eds., *Automatic Differentiation: Theory, Implementation, and Application*, SIAM, 1991.
8. A. Curtis, M. Powell and J. Reid, *On the Estimation of Sparse Jacobian Matrices*, J. Inst. Math. Appl., 13, 1974, 117–94.
9. M. Garey and D. Johnson, *Computers and Intractability - A Guide to the Theory of NP-completeness*, W. H. Freeman and Company, 1979.
10. A. Griewank, *Evaluating Derivatives, Principles and Techniques of Algorithmic Differentiation*, Frontiers in Appl. Math., 19, SIAM, 2000.
11. A. Griewank and S. Reese, *On the Calculation of Jacobian Matrices by the Markowitz Rule*, in [7], 126-135.
12. B. Hajek,*Cooling Schedules for Optimal Annealing*, Mathem. Oper. Res., 13, 1998, 311-329.
13. K. Herley, *Presentation at: Theory Institute on Combinatorial Challenges in Computational Differentiation*, MCSD, Argonne National Laboratory, 1993.
14. R. Jessani and M. Putrino, *Comparison of Single- and Dual-Pass Multiply-Add Fused Floating-Point Units*, IEEE Transactions on Computers, 47, 9, 1998.
15. S. Lin, Bell System Technical Journal, 44, 1965, 2245-2269.
16. W. Miller and C. Wrathall, *Software for Roundoff Analysis of Matrix Algorithms*, Academic Press, 1980.
17. N. Metropolis and A.W. Rosenbluth and M.N. Rosenbluth and A.H. Teller and E . Teller, *Equation of State Calculations by Fast Computing Machines*, J. Chem. Phys., 21, 1953, 1087-92.
18. U. Naumann and B. Christianson, *Differentiation Enabled Compiler Technology*, MOD/DERA/EPSRC grant, GR/R55252.
19. U. Naumann, *Efficient Calculation of Jacobian Matrices by Optimized Application of the Chain Rule to Computational Graphs* Ph.D. thesis, Technical University Dresden, 1999.
20. U. Naumann, *Optimized Jacobian Accumulation Techniques*, in N. Mastorakis, Ed., *Problems in Modern Applied Mathematics*, WSES Press, 2000.
21. U. Naumann, *An Enhanced Markowitz Rule for Accumulating Jacobians Efficiently,* Proceedings of Algorithmy 2000, Slovak University of Technology, 2000.
22. U. Naumann, *Elimination Techniques for Cheap Jacobians*, in [6].
23. U. Naumann, *Cheaper Jacobians by Simulated Annealing*, to appear in SIAM J. Opt.
24. U. Naumann, *Elimination Methods for Computational Graphs. On the Optimal Accumulation of Jacobian Matrices - Conceptual Framework.*, submitted to Math. Prog.
25. G. Newsam and J. Ramsdell, *Estimation of Sparse Jacobian Matrices*, SIAM J. Alg. Dis. Meth., 4, 1983, 404–417.
26. U. Naumann, S. A. Forth, M. Tadjouddine, and J. D. Pryce, *A Standard Interface for Elimination Sequences in Automatic Differentiation*, AMOR Report 2001/03, Cranfield Univ. (RMCS Shrivenham), 2001.
27. D. J. Rose and R. E. Tarjan, *Algorithmic Aspects of Vertex Elimination on Directed Graphs*, SIAM J. Appl. Math., 34, 1, 1978, 176-197.
28. M. Tadjouddine, S. A. Forth, and J. D. Pryce, *AD Tools and Prospects for Optimal AD in CFD Flux Jacobian Calculations*, in [6].
29. P. Van Laarhoven and E. Aarts, *Simulated Annealing: Theory and Applications*, Reidel, 1988.

Optimization and Simulation: Sequential Packing of Flexible Objects Using Evolutionary Algorithms

Henning Behnke, Michael Kolonko, Ulrich Mertins, and Stefan Schnitter

Institut für Mathematik, Technical University Clausthal, D-38678
Clausthal-Zellerfeld, Germany

Abstract. We want to fill a given two-dimensional closed contour as accurately as possible with a fixed number of identical, flexible objects. These objects have to be packed sequentially. They adapt themselves to the surface they are packed on, but their deformation can only be simulated. This type of problem is the two-dimensional cross-section of manufacturing processes where soft material is wound onto a mandrel. We formulate this as a problem of dynamic programming with simulated law of motion. It allows an evolutionary algorithm approach that successfully produces approximate solutions in a non-sequential fashion.

Keywords: Evolutionary algorithms, optimization and simulation, packing problems, dynamic programming.

1 Introduction

We consider the problem of filling a given two-dimensional container with a fixed number N of objects of identical type. The objects are smooth and adapt themselves to the surface their are packed on. The packing has to start at the bottom of the container and the objects have to be placed sequentially, see Fig. 1. The aim is to fill the container, which is given as a closed two dimensional contour, as accurately as possible. The particular additional constraint here is that the exact behaviour of the objects, their deformation function, is not available in a closed analytical form. Instead, it has to be simulated based on assumptions about properties of the material.

Hence, we have to solve an optimization problem with a simulated target function. We use a dynamic programming framework to give a precise definition of our problem. The dynamics of the model depend on the simulated deformation of the objects, which rules out most solution methods of dynamic programming like backward induction. Instead, we suggest an approximate procedure that consists of a genetic algorithm framework optimizing the relative position of the objects using the results of the simulation as fitness.

The main point here is to find a coding of the feasible solutions and of the cost function (i.e. the deviation of the final layout from the target contour) such that genetic algorithm techniques can be applied efficiently. With our approach

K. Steinhöfel (Ed.): SAGA 2001, LNCS 2264, pp. 145–154, 2001.

standard genetic operators yield feasible solutions without any problem specific adaptations. Also, these operations behave 'locally' enough to produce offspring that inherits properties of their parents leading to astonishingly good solutions in very short time for a number of real world problems.

The problem described is the two-dimensional cross-section of certain three-dimensional packing problems that arise in the manufacturing of rotational symmetrical bodies like tubes, rings or tanks from fibre-reinforced plastic. Here, a continuous fibre bundle impregnated with a polymere is wound around a rotating mandrel, see e.g. [6] for technical details. The shape of the mandrel determines the inner contour of the workpiece, the outer contour is formed by the layers of the composite fibre bundles after hardening or cooling. So the winding robot has to put more layers at places where the side of the workpiece is to be stronger and less where there is to be a groove. Generally the aim is to control the robot in such a way, that a predefined outer contour is filled as accurately as possible with a given number of windings. As the spooled material is soft and sticky, it will adapt to the surface or is even pressed on to it to prevent air inclusion.

Due to the rotational symmetry, we can restrict the problem to (one half of) the 2-dimensional cross-sections of the workpiece and of the rope to be spooled, see Fig. 1. Here, (a) shows the cross-section of a ring. The cross-section of the rope when wound onto an even surface will look like Fig. 1 (b), this will be referred to as an *object*. Fig. 1(c) shows one half of the ring (the *target shape*) with four objects placed, i.e. with four rounds of rope already applied. The target shape consists of the lower *starting* and the upper *target contour* . We assume that the axis of the mandrel is parallel to the $x-$axis as indicated in the Figue. 1. We assume further that the number of objects is chosen appropriately (i.e. the area of the target shape is N times the area of the objects). Note that we neglect possible problems in the original 3-dimensional problem that may occur when 'changing the lane' during winding.

Classical problems of packing and cutting, see e.g. [2] mostly consider objects of fixed shape, an exception is [4]. Methods for solving include dynamic programming, integer programming, problem specific algorithms and several heuristics, see [2]. In [3], simulated annealing is applied to a simple packing problem.

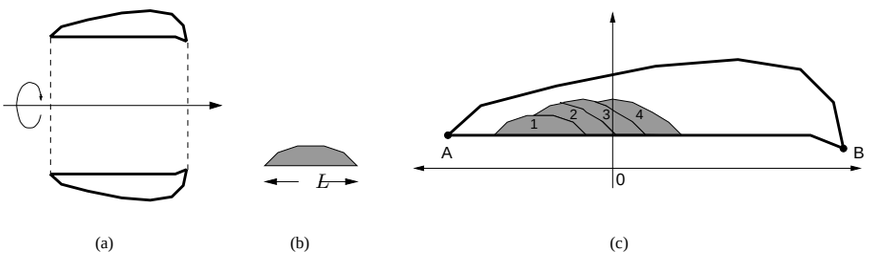

Fig. 1. Target shape and objects.

The paper is organized as follows. In Section 2, we give a formulation of the problem in a dynamic programming set-up and define the solution. In Section 3 we sketch how to simulate the deformation of a single object and how to evaluate the cost of a complete solution approximately. In Section 4 we then present a genetic algorithm that produces good solutions in a non-sequential way. In the final Section 5 we report on some practical experiences with an implementation of the algorithm.

2 The Mathematical Model

Recall that a dynamic programming model (see [1]) consists of states s , actions a , a transition function $T(s, a)$ that maps state-action-pairs into new states, a cost structure with local costs $c(s, a)$ connected to each transition and a terminal cost function $V_0(s)$ for the final state.

Our packing problem fits into this framework if we regard the decision where to place the next object as an action. Then $N :=$ no. of objects to be placed equals the number of optimization stages. The state of the packing has to contain all information relevant for the further placement, in particular, the present upper contour formed by the objects already placed. This will be made precise in the following sub-sections.

2.1 Feasible Contours

We describe the contour by two functions for its upper and lower half. The set of possible contour functions is

$$\mathfrak{C} \;:=\; \{C : [\alpha, \beta] \to \mathbb{R}_+ \mid C \text{ bounded, continuous and piecewise differentiable}\}$$

with $\alpha < \beta, \alpha, \beta \in \mathbb{R}$. We assume that the given *starting contour* C_0 and the *target contour* $\overline{C}$ belong to $\mathfrak{C}$ with $C_0(x) < \overline{C}(x)$ for $x \in (\alpha, \beta)$ and $y_\alpha := C_0(\alpha) = \overline{C}(\alpha),\ y_\beta := C_0(\beta) = \overline{C}(\beta)$, see Fig. 2. During the placement, the starting contour C_0 is changed to contours $C_1 \le C_2 \le \ldots \le C_N \in \mathfrak{C}$ by adding on the objects.

We characterize each point on a contour $C \in \mathfrak{C}$ by its distance from the left endpoint of C measured in arc length, '$C-$distance' for short. To do so, let

$$\Lambda_C(x) \;:=\; \int_\alpha^x \sqrt{1 + C'(z)^2}\, dz, \qquad x \in [\alpha, \beta],\ C \in \mathfrak{C}$$

denote the $C-$distance of the point $(x, C(x))$ from the left starting point $(\alpha, C(\alpha))$ measured in arc length along C.

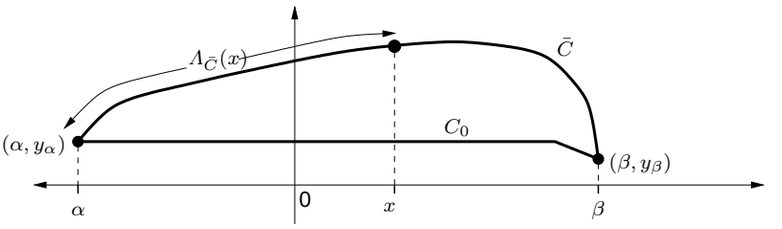

Fig. 2. A target shape with starting contour C_0 and target contour $\bar{C}$, point $(x, \bar{C}(x))$ has distance $\Lambda_{\bar{C}}(x)$ from the left endpoint.

2.2 The State Space

Next we define the state for the dynamic programming model described above.

We assume that the winding robot moves from one side of the target shape to the other and back again, i.e. it produces complete layers of material and may not change its direction freely. This restricts the space of possible solutions but simplifies matters considerably.

For a given starting contour $C_0 \in \mathfrak{C}$ with $y_\alpha = C_0(\alpha), y_\beta = C_0(\beta)$ we define the *state space*

$$S := \{(C,t,d) \mid C \in \mathfrak{C},\ C(\alpha) = y_\alpha, C(\beta) = y_\beta,\ C(x) \geq C_0(x), \alpha < x < \beta, \text{ and } 0 < t < \Lambda_C(\beta),\ d \in \{-1,+1\}\ \}.$$

Here, the state $s = (C,t,d)$ indicates that the objects placed so far form an upper contour C and that the location of the last object placed is represented on C by t (in $C-$distance). $d = +1$ indicates that the present layer runs from left to right, $d = -1$ indicates the opposite direction. Note that the last object was *not* placed *on* C, point t only represents that location, see details below. In the starting state (C_0, t, d), t indicates an arbitrary reference point for the first placement.

2.3 Action and Placement

As described above, the placement of an object has to be formulated as action in the dynamic programming model.

We assume that the objects all have identical shape before placement (i.e. the rope of material has a constant cross-section) and that the length L of their lower contour (*base line*) and the volume of the cross-section when placed on a surface are constant, see Fig. 1 and 3. As reference point for the placement (*basepoint*) we take the middle of this base line, it will be marked by a black triangle in the figures below.

Let $\lambda > 0$ be the maximal $C-$distance between two successive objects that can be managed by the winding machine. Then, *action* $a \in [0,1]$ applied to state (C,t,d) means to place the next object with its basepoint at a $C-$distance of

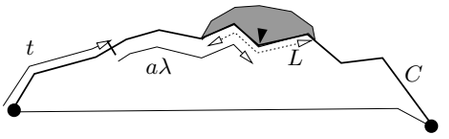

Fig. 3. An object is placed on C with its basepoint at $C-$distance $t + a\lambda$.

$da\lambda$ from t, i.e. at $C-$distance $t + da\lambda$ from the left endpoint, see Fig. 3. Near the left and right border of the target shape this may lead to infeasible positions $\notin [0, \Lambda_C(\beta)]$. In this case the direction d is changed to $-d$ and the object is placed at the next feasible position, hence the exact position of the new object is at a $C-$distance $\tau(s, a)$ where

$$\tau(s,a) = \tau((C,t,d),a) \;:=\; \begin{cases} t + da\lambda & \text{if } L/2 \le t + da\lambda \le \Lambda_C(\beta) - L/2 \\ L/2 & \text{if } t + da\lambda < L/2 \\ \Lambda_C(\beta) - L/2 & \text{if } t + da\lambda > \Lambda_C(\beta) - L/2 \end{cases} \tag{1}$$

and the direction after placement is given by

$$\delta(s,a) = \delta((C,t,d),a) \;:=\; \begin{cases} d & \text{if} \quad L/2 \le t + da\lambda \le \Lambda_C(\beta) - L/2 \\ -d & \text{else} \end{cases} .$$

Note that this notion of feasible placement does not prevent objects to be placed completely outside of the target shape, see Fig. 6 below for an example.

2.4 The State Transition

The state transition function has to describe the contour after a placement of a new object. Let the local deformation function for an object

$$l \mapsto \gamma(C, r, l)$$

be a bounded, continuous and piecewise differentiable function that gives the height of the object l units (in $C-$distance) from its left endpoint if the object is placed on C at a $C-$distance r, see Fig. 4. In the next Section, we shall give an idea how to simulate γ.

The new contour after placing an object in state $s = (C, t, d) \in S$ with a relative offset of $a \in [0, 1]$ is now given by

$$C_{s,a}(x) \;:=\; \begin{cases} C(x) & \text{for } \Lambda_C(x) \le \tau(s,a) - L/2 \\ C(x) + \gamma(C, \tau(s,a), l) & \text{for } \Lambda_C(x) = \tau(s,a) - L/2 + l,\ 0 \le l \le L \\ C(x) & \text{for } \Lambda_C(x) \ge \tau(s,a) + L/2 \end{cases} \tag{2}$$

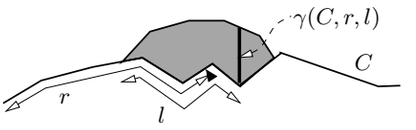

Fig. 4. The object is placed at $C-$distance r. Its height l units from its left endpoint is given by $\gamma(C,r,l)$.

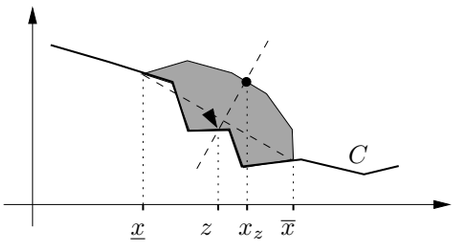

Fig. 5. The reference point for the next offset is marked by a black dot. It is a projection of the basepoint (black triangle) of the last placement onto the new surface.

for $\alpha \le x \le \beta$. Note that the region outside $\tau(s,a) \pm L/2$ refers to that part of the contour that is left untouched by the newly placed object. Obviously, $C_{s,a}$ belongs to $\mathfrak{C}$ for all s, a.

We finally have to determine how the location $\tau(s,a)$ at which the last object was placed on C is represented on the new contour $C_{s,a}$ as reference point for the next placement. We use the projection of the basepoint of the last object onto the new contour where the projection is perpendicular to the line connecting left and right endpoint of the object, see Fig. 5. From simple geometric considerations we obtain that the $x-$coordinate $x_z = x_z(s,a)$ of this point is the smallest solution y of

$$(y-z)(\overline{x}-\underline{x})/(C(\underline{x})-C(\overline{x}))+C(z) \;=\; C_{s,a}(y). \tag{3}$$

where $\underline{x}, \overline{x}$ and z are as in Fig. 5. Now the complete *transition function* for the dynamic programming model with local deformation functions γ can be defined from (2) and (3) as

$$T(s,a) \;=\; T((C,t,d),a) \;:=\; (C_{s,a}, \Lambda_{C_{s,a}}(x_z(s,a)), \delta(s,a)). \tag{4}$$

2.5 The Cost Structure

We have no local costs in our model but a *terminal cost function*

$$V_0(s) \;=\; V_0((C,t,d)) \;:=\; \int_\alpha^\beta \mid C(x) - \overline{C}(x) \mid \, dx. \tag{5}$$

which gives the deviation from the target contour in the final state.

Now the dynamic programming problem is completely determined. A *solution* to this problem is a sequence of offsets

$$a_0, \ldots, a_{N-1} \ \in \ [0,1]^N$$

for the consecutive placement of the N objects on the starting contour C_0. Note that *any* sequence from $[0,1]^N$ constitutes a feasible solution. Its costs are given by

$$\begin{aligned} V_N(s_0, a_0, \ldots, a_{N-1}) \ &= \ V_0(T(\ldots(T(T(s_0, a_0), a_1), \ldots, a_{N-1}) \\ &= \ \int_\alpha^\beta \mid C_N(x) - \overline{C}(x) \mid \ dx \end{aligned} \tag{6}$$

where C_N is the final upper contour after applying $a_0, \ldots, a_{N-1}$ to s_0. An *optimal solution* for starting state $s_0 = (C_0, t, d)$ is a sequence $a_0^*, \ldots, a_{N-1}^* \ \in \ [0,1]^N$ that minimizes (6) over $[0,1]^N$.

3 Simulating the Local Deformation

We only sketch the simulation very briefly as it is not in the focus of the present paper.

In our experiments it turned out to be sufficiently accurate to use piecewise linear functions for the contours as well as for the objects, and hence for the deformation function γ. This greatly simplifies the calculation described above and also the representation of a contour on the computer.

Let $\gamma_0 : [0, L] \to \mathbb{R}_+$ denote the piecewise linear upper contour of the object when placed on an even surface. We add γ_0 to the present contour at the location given by $\tau(s, a)$. Let Γ denote that part of the new contour $C_{s,a}$ that is formed by γ_0.

We use two different levels of simulation to improve Γ. The first level assumes that the surface of an object will always tend to form a section of a circle. From simple geometric considerations we can determine a circle that runs through the two endpoints of the object placed at $\tau(s, a)$ ($\underline{x}, \overline{x}$ in Fig. 5) and encloses the exact volume of the object. We then approximate the relevant section of the circle by a polygone which is slighly changed afterwards to enclose the correct volume. This yields an improved deformation function γ_1.

This is a very fast procedure that works well as long as the surface C is not too ragged. Otherwise the circle may intersect with C. This has to be detected and then the second more detailed level of simulation has to be called. Here, the polygone describing Γ is optimized to yield smallest length and minimal distortion with the correct enclosed volume. This is done using a standard library for non-linear optimization. The result is a simulated contour $C_{s,a}$ which takes into account more physical properties of the material.

For a given starting state s_0 and a sequence of off-sets $a_0, \ldots, a_{N-1} \in [0,1]$ we evaluate the cost function $V_N(s_0, a_0, \ldots, a_{N-1})$ as given in (6) successively by N calls of the simulation.

4 An Evolutionary Algorithm

We shall now describe an evolutionary heuristic algorithm that yielded excellent solutions in our experiments. Let us briefly collect the main ingredients of genetic algorithms (see e.g. [5]). A starting population of solutions ('individuals') is created at random. It is subject to random genetic operations that produce off-spring solutions. Standard operators are crossover of two randomly selected solutions and random mutation of a solution. Repeated application of these operators enlarges the population which is then reduced by a selection mechanism to its former size. This cycle (' generation') is repeated for a certain number of times. The selection prefers solutions with low costs. Thus the members of the successive populations tend to become good solutions of the optimization problem. There are many different ways to select solutions for the operations and to take into account their 'fitness', i.e. the cost of the solution.

In the preceding Sections we have formulated our problem such that the structure of a solution $a_0, \ldots, a_{N-1}$ is extremely simple : it lies in the unit rectangle $[0,1]^N$ and each vector in this space represents a feasible placement of N objects. Therefore we may use simple standard genetic operators without any problem specific mending operation. The fitness of a solution $a_0, \ldots, a_{N-1}$ is measured by the cost function $V_N(s_0, a_0, \ldots, a_{N-1})$ as defined in (6). Note that we minimize fitness.

For *crossover* we use a one-point or uniform operator. The one-point crossover takes two randomly selected solutions $(a_0, \ldots, a_{N-1})$ and $(a'_0, \ldots, a'_{N-1})$ as parents, selects a random position $l \in \{0, \ldots, N-1\}$ and returns the solution $(a_0, \ldots, a_l, a'_{l+1}, \ldots, a'_{N-1})$. In terms of controlling the winding machine this means to use off-sets from the first solution for the first l rounds and then to follow the second solution. Uniform crossover selects each position in the resulting solution independently from one of the parents.

Note that we restricted changes of the direction d to the lateral borders where it is calculated by the cost function. Including changes of direction into the solution would cause problems with crossover operators, as the impact of a change in direction depends heavily on its *absolute* location within a placement which in general will be changed during a crossover. Then the offspring individuum will be quite different from its parents which would make the algorithm search rather arbitrarily in the solution space.

We apply *mutation* to each new offspring solution created by crossover. Mutation operators randomly change single offsets of a solution. Again, even a small change of a single offset may lead to a completely different solution, as all subsequent objects are shifted. We therefore use a 'local' mutation that also changes the next offset to neutralize the widerange effect of the mutated offset. The procedure is as follows: select an index $0 \le i \le N-2$ randomly with a bias towards higher values. In the solution to be mutated, let $\sigma = a_i + a_{i+1}$. Select a new value a'_i uniformly distributed over

$$\Big[\max\{1,\sigma\} - 1,\ \min\{1,\sigma\}\Big]$$

and put $a'_{i+1} = \sigma - a'_i$. Then $a'_i, a'_{i+1} \in [0,1]$ and $a'_i + a'_{i+1} = \sigma$, i.e. the position of objects $i+2, i+3, \ldots$ on the their contour is unchanged. Note however, that the contours themselves will have changed due to the moving of objects i and $i+1$. This may be illustrated by looking at the $a_0, \ldots, a_{N-1}$ as distances of beads on a string. Mutation then moves exactly one of the beads and leaves the others unchanged.

Let M be the population size. Crossover and mutation are repeated for a fixed number M' of times enlarging the population to $M + M'$ elements . The selection mechanism then takes M solutions from the enlarged population to form the next population. These may be the M solutions with lowest (simulated) costs V_N , or they may be drawn with a probability density proportional to the V_N-values or proportional to their rank with respect to these cost values. Again these are standard selection procedures.

We provided the system with a graphical user interface , that allows to display the evolution of the population with their cost values. The evolution may be stopped and the individuals may be inspected graphically as shown in Fig. 6 and 7 below.

5 Experimental Results

We applied our model to several contours supplied by our industrial partner. Typically there were about 60 objects to be placed. Target contour and the cross-section of the object were given as polygones, similar to Fig. 1.

A starting population was created randomly, good results were obtained with individuals that had constant offsets $a_0, \ldots, a_{N-1}$ with different random values for different individuals. As starting state $(C_0, t, -1)$ we used the lower contour C_0 with its middle point as reference point for the first object.

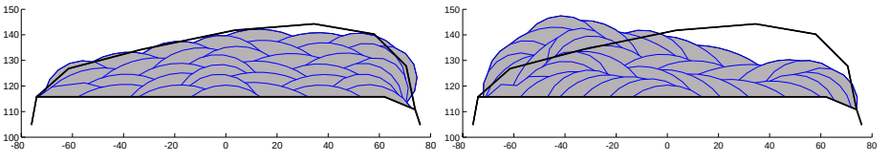

Fig. 6. Two individuals from a starting population.

Fig. 6 shows two individuals from a starting population. The one on the left was produced with econstant offsets, the other with random offsets. Note that these are obtained by placing a random vector of offsets into the contour using the simulation of the cost function as described in Section 3. The much improved results in Fig. 7 show the two best individuals obtained after about 300 generations of the genetic algorithm which took about 30 sec of cpu time on a 650 MHz Pentium.

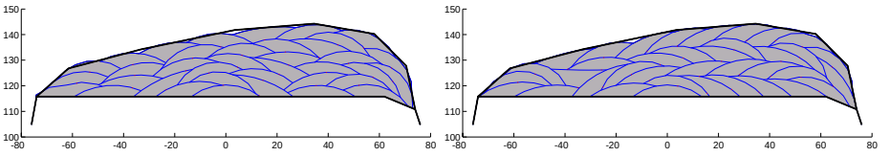

Fig. 7. Two individuals from a later population.

Results from this system were fed into a real winding machine. The contours obtained in reality were in very good accordance with our simulated contours. This shows that our simple model of deformation reflects reality with sufficient accuracy.

6 Conclusions

We modelled a complex technical process of placing flexible objects into a given contour as a deterministic dynamic programming problem. The classical sequential solution methods are not applicable as the main ingredient of the model, the state transition function is not known. It can only be approximated by a separate local optimization that simulates the physical properties of the flexible material.

We therefore took a non-sequential approach, that optimizes all placement decisions at once with a genetic algorithm. The formulation of the model allowed to produce feasible solutions with standard genetic operators without the need of problem specific mending operations.

In real world applications with about 60 objects this algorithm produced astonishingly good results in just a few seconds.

References

1. Bertsekas, Dimitri P. *Dynamic programming : deterministic and stochastic models* Englewood Cliffs, N. J.: Prentice-Hall, 1987.
2. Dowsland, Kathryn A. and William B. Dowsland *Packing Problems* Europ. J. Operational Research, 56 (1992), 2-14.
3. Dowsland, Kathryn A. *Some Experiments with Simulated Annealing Techniques for Packing Problems* Europ. J. Operational Research, 68 (1993), 389-399.
4. Scheithauer, G. *The Solution of Packing Problems with Pieces of Variable Length and Additional Allocation Constraints* Optimization, 34 (1995), 81-86.
5. Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolutionary Programs.* Springer Verlag, 1992.
6. T. J. Reinhart. *Composites.* Engineered materials handbook, ASM International / Handbook Committee, Metals Park, Ohio : ASM International, 1988.

Stochastic Finite Learning

Thomas Zeugmann

Institut für Theoretische Informatik, Med. Universität zu Lübeck,
Wallstraße 40, 23560 Lübeck, Germany
thomas@tcs.mu-luebeck.de

Abstract. Recently, we have developed a learning model, called *stochastic finite learning*, that makes a connection between concepts from PAC learning and inductive inference learning models. The motivation for this work is as follows. Within Gold's (1967) model of learning in the limit many important learning problems can be formalized and it can be shown that they are algorithmically solvable *in principle*. However, since a limit learner is only supposed to converge, one never knows at any particular learning stage whether or not it has already been successful. Such an uncertainty may be not acceptable in many applications. The present paper surveys the new approach to overcome this uncertainty that potentially has a wide range of applicability.

Keywords: Inductive inference, average-case analysis, stochastic finite learning, conjunctive concepts, pattern languages.

1 Introduction

Let us assume that we have to deal with the following problem. We are given a concept class $\mathcal{C}$ and should design a learner for it. Next, suppose we already know or could prove $\mathcal{C}$ not to be PAC learnable. But it can be shown that $\mathcal{C}$ is learnable within Gold's [6] model of learning in the limit. Consequently, there is a learner behaving as follows. When fed any of the data sequences allowed in this model, it converges in the limit to a hypothesis correctly describing the target concept. Nothing more is known. Since it is either undecidable whether or not the sequence of hypotheses has already converged or, even if it decidable, it practically infeasible to do so, one never knows whether or not the learner has already converged. But such an *uncertainty* may not be tolerable in many applications. So, how can we recover?

In general, there may be no way to overcome this uncertainty at all, or at least not efficiently. But if the learner satisfies some additional assumptions outlined below then there is a rather general way to transform learning in the limit into *stochastic finite learning*. Within this survey, we aim to describe our method developed and to exemplify it by using some well-known concept classes.

It should also be noted that our approach may be beneficial even in case that the considered concept class is PAC learnable.

K. Steinhöfel (Ed.): SAGA 2001, LNCS 2264, pp. 155–171, 2001.

2 Learning in the Limit

Gold's [6] model of learning in the limit allows one to formalize a rather general class of learning problems, i.e., learning from examples. For defining this model we assume any recursively enumerable set $\mathcal{X}$ and refer to it as the *learning domain*. By $w.p.\ (\mathcal{X})$ we denote the power set of $\mathcal{X}$. Let $\mathcal{C} sw.p.\ (\mathcal{X})$, and let $c \in \mathcal{C}$ be non-empty; then we refer to $\mathcal{C}$ and c as a *concept class* and a *concept*, respectively. Let c be a concept, and let $t = (x_j)_{j \in \mathbb{N}}$ be any infinite sequence of elements $x_j \in c$ such that $\mathrm{range}(t) =_{df} \{x_j \mid j \in \mathbb{N}\} = c$. Then t is said to be a *positive presentation* or, synonymously, a *text* for c. By $text(c)$ we denote the set of all positive presentations for c. Moreover, let t be a positive presentation, and let $y \in \mathbb{N}$. Then, we set $t_y = x_0, \ldots, x_y$, i.e., t_y is the initial segment of t of length $y+1$, and $t_y^+ =_{df} \{x_j \mid j \leqslant y\}$. We refer to t_y^+ as the *content* of t_y.

Alternatively, one can also consider *complete presentations* or, synonymously, *informants*. Let c be a concept; then any sequence $i = (x_j, b_j)_{j \in \mathbb{N}}$ of labeled examples, where $b_j \in \{+, -\}$ such that $\{x_j \mid j \in \mathbb{N}\} = \mathcal{X}$ and $i^+ = \{x_j \mid (x_j, b_j) = (x_j, +),\ j \in \mathbb{N}\} = c$ and $i^- = \{x_j \mid (x_j, b_j) = (x_j, -),\ j \in \mathbb{N}\} = \mathcal{X} \setminus c$ is called an informant for c. For the sake of presentation, the following definitions are only given for the text case, the generalization to the informant case should be obvious. We sometimes use the term *data sequence* to refer to both text and informant, respectively.

An *inductive inference machine* (abbr. IIM) is an algorithm that takes as input larger and larger initial segments of a text and outputs, after each input, a hypothesis from a prespecified *hypothesis space* $\mathcal{H} = (h_j)_{j \in \mathbb{N}}$. The indices j are regarded as suitable finite encodings of the concepts described by the hypotheses. A hypothesis is said to describe a concept c iff $c = h$.

Definition 1. *Let $\mathcal{C}$ be any concept class, and let $\mathcal{H} = (h_j)_{j \in \mathbb{N}}$ be a hypothesis space for it. $\mathcal{C}$ is called learnable in the limit from text iff there is an IIM M such that for every $c \in \mathcal{X}$ and every text t for c,*

(1) *for all $n \in \mathbb{N}^+$, $M(t_n)$ is defined,*
(2) *there is a j such that $c = h_j$ and for all but finitely many $n \in \mathbb{N}^+$, $M(t_n) = j$.*

By LIM we denote the collection of all concepts classes $\mathcal{C}$ that are learnable in the limit from text. Note that instead of LIM sometimes EX is used.

It can be shown that many interesting concepts classes are learnable in the limit from text and informant, respectively. However, it should be noted that there are also some major *points of concern*. First, in the definition given above, the limit learner has always access to the whole actual initial segment of the data sequence provided. Clearly, in practice such a learner would run out of space pretty soon. This problem can be overcome be considering *incremental* learning algorithms. We do not develop this topic here any further. Instead, the reader is referred to [3,13].

Second, since a limit learner is only supposed to converge, one never knows at any particular learning stage whether or not it has already been successful.

Such an uncertainty may be not acceptable in many applications. Clearly, one could modify the definition of learning in the limit by adding the requirement that convergence should be *decidable*. Then one would arrive at finite learning (cf. Gold [6]). However, the power of finite learning is much smaller than that of learning in the limit. Finally, it seems difficult to define an appropriate measure for the complexity of learning in the limit (cf. Pitt [15]).

Note that the latter points of concern are really serious, since a major task of algorithmic learning theory consists in providing *performance guarantees*. Thus, it is only natural to ask why we not simply refrain to consider learning the limit. Alternatively, we could try to use the well-known PAC model (cf. [20]). Our answer is twofold. During the last decade many interesting concept classes have been proven to be *not* PAC learnable. Moreover, even if a concept class is PAC learnable, the performance guarantees provided by this model are often much too pessimistic. So, how can we recover?

Though in general there may be no way to recover, we recently discovered a rather general method that allows one to overcome the difficulties described above. In the remaining part of this paper, we outline this approach. The key ingredient is an *average-case* analysis of the learning algorithms considered as initiated by Zeugmann [22]. As it turned out, the most promising complexity measure to be considered is the so called *total learning time*, i.e., the overall time taken by a learner until convergence. Formally, the total learning time is defined as follows (cf. Daley and Smith [4]). Let $\mathcal{C}$ be any concept class, and let M be any IIM that learns $\mathcal{C}$ in the limit. Then, for every $c \in \mathcal{C}$ and every text t for c, let

$$Conv(M,t) =_{df} \text{the least number } m \in \mathbb{N}^+ \text{ such that for all } n \geqslant m,\ M(t_n) = M(t_m)$$

denote the *stage of convergence* of M on t (cf. [6]). Note that $Conv(M,t) = \infty$ if M does not learn the target concept from its text t. Moreover, by $T_M(t_n)$ we denote the time to compute $M(t_n)$. We measure this time as a function of the length of the input and call it the *update time*. Finally, the total learning time taken by the IIM M on successive input t is defined as

$$TT(M,t) =_{df} \sum_{n=1}^{Conv(M,t)} T_M(t_n).$$

Clearly, if M does not learn the target language from text t then the total learning time is *infinite*.

It has been argued elsewhere that within the learning in the limit paradigm a learning algorithm is invoked only when the current hypothesis has some problem with the latest observed data. Clearly, if this viewpoint is adopted, then our definitions of learning and of the total learning time seem inappropriate. Note however, that there may be no way at all to decide whether or not the current hypothesis is not correct for the latest piece of data received. But even if one can decide whether or not the latest piece of data obtained is correctly reflected

by the current hypothesis, such a test may be computationally infeasible. For example, consider the well-known class of all pattern languages (cf. Angluin [1]). The membership problem for the pattern languages is $\mathcal{NP}$-complete. Thus, directly testing consistency would immediately lead to a non-polynomial update time unless $\mathcal{P} = \mathcal{NP}$. Moreover, at least for the informant case there is no polynomial time update learning algorithm at all that infers the whole class of pattern languages (cf. [21]). Thus, adopting the definition of the total learning time given above is reasonable.

Next, one has to assume an appropriate class of probability distributions $\mathcal{D}$ over the relevant learning domain. Ideally, this class should be parameterized. In what follows, it is assumed that the data sequences are drawn at random with respect to any of the distributions from $\mathcal{D}$. Next, it is advantageous to introduce a random variable $CONV$ for the stage of convergence. Note that $CONV$ can be also interpreted as the total number of examples read by the IIM M until convergence. The first major step of the average-case analysis consists now in determining the expectation $E[CONV]$. Clearly, $E[CONV]$ should be finite for all concepts $c \in \mathcal{C}$ and all distributions $D \in \mathcal{D}$.

Next, Markov's inequality provides us with the following tail bounds:

$$\Pr(CONV \geq t \cdot E[CONV]) \leq \frac{1}{t} .$$

Having reached this stage, learning in the limit can be transformed into *stochastic finite learning*.

3 Stochastic Finite Learning

A stochastic finite learner is successively fed data about the target concept. Note that these data are generated randomly with respect to one of the probability distributions from the class of underlying probability distributions. Additionally, the learner takes a confidence parameter δ as input. But in contrast to learning in the limit, the learner itself decides how many examples it wants to read. Then it computes a hypothesis, outputs it and stops. The hypothesis output is correct for the target with probability at least $1 - \delta$.

Some more remarks are mandatory here. The description given above explains how it works, but not why it does. Intuitively, the stochastic finite learner simulates the limit learner until an upper bound for twice the expected total number of examples needed until convergence has been met. Assuming this to be true, by Markov's inequality the limit learner has now converged with probability $1/2$. All what is left, is to decrease the probability of failure. This can be done by using again Markov's inequality, i.e., increasing the sample complexity by a factor of $1/\delta$ results in a confidence of $1 - \delta$ for having reached the stage of convergence.

Finally, it remains to explain how the stochastic finite learner can calculate the upper bound for $E[CONV]$. This is precisely the point where we need the parameterization of the class $\mathcal{D}$ of underlying probability distributions. Since in

general, it is not know which distribution from $\mathcal{D}$ has been chosen, one has to assume a bit of *prior knowledge* or *domain knowledge* provided by suitable upper and/or lower bounds for the parameters involved. A more serious difficulty is to incorporate the unknown target concept into this estimate. This step depends on the concrete learning problem on hand, and requires some extra effort. We shall point to some examples at the end of this section.

Next, we formally define stochastic finite learning.

Definition 2. *Let $\mathcal{D}$ be a set of probability distributions on the learning domain, $\mathcal{C}$ a concept class, $\mathcal{H}$ a hypothesis space for $\mathcal{C}$, and $\delta \in (0,1)$. $(\mathcal{C},\mathcal{D})$ is said to be* stochastically finitely learnable with δ-confidence *with respect to $\mathcal{H}$ iff there is an IIM M that for every $c \in \mathcal{C}$ and every $D \in \mathcal{D}$ performs as follows. Given any random data sequence θ for c generated according to D, M stops after having seen a finite number of examples and outputs a single hypothesis $h \in \mathcal{H}$. With probability at least $1-\delta$ (with respect to distribution D) h has to be correct, that is $c = h$.*

If stochastic finite learning can be achieved with δ-confidence for every $\delta > 0$ then we say that $(\mathcal{C},\mathcal{D})$ can be learned stochastically finite *with high confidence*.

Note that our model of stochastic finite learning differs in several ways from the PAC model. First, it is not completely distribution independent. Thus, from that perspective, this variant is weaker than the PAC-model. But the hypothesis computed is exactly correct with high probability. That is, we do *not* measure the quality of the hypothesis with respect to the underlying probability distribution. For seeing the difference, consider a target concept class $\mathcal{C}$ over some fixed learning domain $\mathcal{X}$ containing the concept $\mathcal{X}$. Furthermore, suppose $\mathcal{X}$ to be learnable from positive data only. Thus, it is reasonable to assume that all negative examples have probability 0. Consequently, a PAC learner could simply output a finite description for $\mathcal{X}$ independent of the particular $c \in \mathcal{C}$ to be learned. It would even learn from 0 examples without any error (where error is measured as usual in the PAC model). But this may be hardly what one really wants to learn. In contrast, a stochastic finite learner would always output a hypothesis h such that $c = h$ with any desired probability. Last but not least, in the PAC model the sample complexity depends exclusively on the VC dimension of the target concept class and the error and confidence parameters ε and δ, respectively (cf. [2]). In contrast, a stochastic finite learner decides itself how many examples it wishes to read.

Finally, if the limit learner M is additionally known to be *conservative* and *rearrangement independent* much more efficient stochastic finite learners are available. A learner is said to be rearrangement independent if its output depends exclusively on the range and length of its input (cf. [12] and the references therein). Furthermore, a learner is conservative if it exclusively performs mind changes that can be justified by an inconsistency of the abandoned hypothesis with the data received so far (cf. [1] for a formal definition). In this case, we could even prove *exponentially* shrinking tail bounds for $E[CONV]$.

Theorem 1. *(*Rossmanith and Zeugmann [19]*.) Let* $CONV$ *be the sample complexity of a conservative and rearrangement-independent learning algorithm. Then*

$$\Pr(CONV) \geq 2t \cdot E[CONV]) \leq 2^{-t} \text{ for all } t \in \mathbb{N} .$$

Having this theorem, the sample complexity of the resulting stochastic finite learner is only by a factor of $\log(1/\delta)$ larger than that of the original limit learner. Consequently, Theorem 1 puts the importance of conservative and rearrangement-independent learners into the right perspective. As long as the learnability of indexed families is concerned, these results have a wide range of potential applications, since every conservative learner can be transformed into a learner that is both conservative and rearrangement-independent provided the hypothesis space is appropriately chosen (cf. Lange and Zeugmann [12]).

Since the distribution of $CONV$ decreases geometrically, all higher moments of $CONV$ exists, too. Thus, instead of applying Theorem 1 directly, one can hope for further improvements by applying even sharper tail bounds using for example Chebyshev's inequality.

Our model of stochastic finite learning differs to a certain extent from the PAC model. First, it is *not* completely distribution independent, since a bit of *additional knowledge* concerning the underlying probability distributions is required. Thus, from that perspective, stochastic finite learning is weaker than the PAC model. But with respect to the quality of its hypotheses, it is stronger than the PAC model by requiring the output to be *probably exactly correct* rather than *probably approximately correct*. That is, we do *not* measure the quality of the hypothesis with respect to the underlying probability distribution. Note that exact identification with high confidence has been considered within the PAC paradigm, too (cf., e.g., Goldman *et al.* [7]).

Furthermore, in the uniform PAC model as introduced in Valiant [20] the sample complexity depends exclusively on the VC dimension of the target concept class and the error and confidence parameters ε and δ, respectively. This model has been generalized by allowing the sample size to depend on the concept complexity, too (cf., e.g., Blumer *et al.* [2] and Haussler *et al.* [9]). Provided no upper bound for the concept complexity of the target concept is given, such PAC learners decide themselves how many examples they wishes to read (cf. [9]). This feature is also adopted to our setting of stochastic finite learning. However, all variants of PAC learning we are aware of require that all hypotheses from the relevant hypothesis space are uniformly polynomially evaluable. Though this requirement may be necessary in some cases to achieve (efficient) stochastic finite learning, it is not necessary in general as we shall see below.

4 Learning Conjunctive Concepts

In this section we exemplify the general approach outlined above by using the class of all concepts describable by a monomial. For all details omitted the reader is referred to Reischuk and Zeugmann [17].

To define the classes of concepts we deal with in this paper let $\mathcal{L}_n = \{x_1, \bar{x}_1, x_2, \bar{x}_2 \ldots, x_n, \bar{x}_n\}$ be a set of literals. x_i is a *positive* literal and $\bar{x}_i$ a *negative* one. A conjunction of literals defines a ***monomial***. For a monomial m let $\#(m)$ denote its length, that is the number of literals in it.

m describes a subset $L(m)$ of $\mathcal{X}_n$, in other words a concept, in the obvious way: the concept contains exactly those binary vectors for which the monomial evaluates to 1, that is $L(m) := \{b \in \mathcal{X}_n \mid m(b) = 1\}$. The collection of objects we are going to learn is the set $\mathcal{C}_n$ of all concepts that are describable by monomials over $\mathcal{X}_n$. There are two trivial concepts, the empty subset and $\mathcal{X}_n$ itself. $\mathcal{X}_n$, which will also be called "TRUE", can be represented by the empty monomial. The concept "FALSE" has several descriptions. To avoid ambiguity, we always represent "FALSE" by the monomial $x_1\bar{x}_1 \ldots x_n\bar{x}_n$. Furthermore, we often identify the set of all monomials over $\mathcal{L}_n$ and the concept class $\mathcal{C}_n$. Note that $|\mathcal{C}_n| = 3^n + 1$.

For the concept class $\mathcal{C}_n$ we choose as hypothesis space the set of all monomials over $\mathcal{L}_n$. We shall distinguish learning from positive data only and learning from both positive and negative data. Note that when considering learning from positive data only, one cannot decide whether or not the learner has already converged. When learning from positive and negative data is considered, the stage of convergence is *decidable*, but one would have to read the data sequence until all Boolean vectors did appear. Thus, for any interesting n decidability is practically infeasible.

The learner used is Haussler's [8] Wholist algorithm. Note that this learner computes a new hypothesis by using only the most recent example received and his old hypothesis. Such learners are called *iterative* (cf. [13]). For the sake of completeness, we include the Wholist learning algorithm here. By $data(c)$ we denote any data sequence for the concept c.

Algorithm $\mathcal{P}$:
On input sequence $\langle(b_1, c(b_1)), (b_2, c(b_2)), \ldots\rangle$ do the following:

Initialize $h_0 := x_1\bar{x}_1 \ldots x_n\bar{x}_n$.
for $i = 1, 2, \ldots$ **do**
let h_{i-1} denote $\mathcal{P}$'s internal hypothesis produced before receiving $(b_i, c(b_i))$;
when receiving $(b_i, c(b_i))$ test whether or not $h_{i-1}(b_i) = c(b_i)$.
if $h_{i-1}(b_i) = c(b_i)$ **then** set $h_i := h_{i-1}$ and output h_i.
else for $j := 1$ **to** n **do**
if $b_i^j = 1$ **then** delete $\bar{x}_j$ in h_{i-1} **else** delete x_j in h_{i-1};
let h_i be the resulting monomial and output h_i.
end.

Now, it is easy to see that Algorithm $\mathcal{P}$ learns the set of all monomials in the limit. If the target monomial is the concept "FALSE", then Algorithm $\mathcal{P}$ immediately converges. Thus, we call "FALSE" the minimal concept. If the

target concept contains precisely n literals, then one positive example suffices. This positive example is unique.

In the general case, we call the literals in a monomial m ***relevant***. All other literals in $\mathcal{L}_n$ are said to be ***irrelevant*** for m. There are $2n - \#(m)$ irrelevant literals. We call bit i **relevant** for m if x_i or $\bar{x_i}$ is relevant for m. By $\boldsymbol{k = k(m)} = n - \#(m)$ we denote the number of irrelevant bits.

First, we consider learning from positive data. To avoid the trivial cases, we let $c = L(m) \in \mathcal{C}_n$ be a concept with monomial $m = \bigwedge_{j=1}^{\#(m)} \ell_{i_j}$ such that $k = k(m) = n - \#(m) > 0$. There are 2^k positive examples for c. For the sake of presentation, we assume these examples to be ***binomially distributed***. That is, in a random positive example all entries corresponding to irrelevant bits are selected independently of each other. With some probability p this will be a 1, and with probability $q := 1 - p$ a 0. We shall consider only nontrivial distributions where $0 < p < 1$. Note that otherwise the data sequence does not contain all positive examples. We aim to compute the expected number of examples taken by $\boldsymbol{\mathcal{P}}$ until convergence. Again we use $CONV$ to denote a random variable counting the number of examples read till convergence.

The first example received forces $\boldsymbol{\mathcal{P}}$ to delete precisely n of the $2n$ literals in h_0. Thus, this example always plays a *special* role. Note that the resulting hypothesis h_1 depends on b_1, but the number k of literals that remain to be deleted from h_1 until convergence is *independent* of b_1. Using tail bound techniques, we can show the following theorem.

Theorem 2. *Let $c = L(m)$ be a non-minimal concept in $\mathcal{C}_n$, and let the positive examples for c be binomially distributed with parameter p. Define $\boldsymbol{\psi} := \min\{\frac{1}{1-p}, \frac{1}{p}\}$. Then the expected number of positive examples needed by algorithm $\boldsymbol{\mathcal{P}}$ until convergence can be bounded by $E[CONV] \leq \lceil \log_{\psi} k(m) \rceil + 3$.*

Proof. Let $k := k(m)$. The first positive example contains ν times a 1 and $k - \nu$ many 0 with probability $\binom{k}{\nu} p^{\nu} q^{k-\nu}$ at the positions not corresponding to a literal in the target monomial m. Now, assuming any such vector, we easily see that h_1 contains ν positive irrelevant literals and $k - \nu$ negative literals. Therefore, in order to achieve convergence, the algorithm $\boldsymbol{\mathcal{P}}$ now needs positive examples that contain at least one 0 for each positive irrelevant literal and at least one 1 for each negative irrelevant literal. Thus, the probability that at least one irrelevant literal survives μ subsequent positive examples is bounded by $\nu q^{\mu} + (k - \nu)p^{\mu}$. Therefore,

$$\Pr[CONV - 1 > \mu] \leq \sum_{\nu=0}^{k} \binom{k}{\nu} p^{\nu} q^{k-\nu} \cdot \left(\nu q^{\mu} + (k - \nu)p^{\mu}\right) .$$

Next, we derive a closed formula for the sum given above.

Claim 1. $\displaystyle\sum_{\nu=0}^{k} \binom{k}{\nu} p^{\nu} q^{k-\nu} \cdot \nu = kp \quad \text{and} \quad \sum_{\nu=0}^{k} \binom{k}{\nu} p^{\nu} q^{k-\nu} \cdot (k - \nu) = kq$

The first equality can be shown as follows.

$$\begin{aligned}
\sum_{\nu=0}^{k}\binom{k}{\nu}p^{\nu}q^{k-\nu}\cdot\nu \;&=\; \sum_{\nu=1}^{k}\binom{k}{\nu}p^{\nu}q^{k-\nu}\cdot\nu \;=\; \sum_{\nu=0}^{k-1}\binom{k}{\nu+1}p^{\nu+1}q^{k-1-\nu}\cdot(\nu+1)\\
&=\; \sum_{\nu=0}^{k-1}k\cdot\binom{k-1}{\nu}p^{\nu+1}q^{k-(\nu+1)}\\
&=\; k\,p\cdot\sum_{\nu=0}^{k-1}\binom{k-1}{\nu}p^{\nu}q^{(k-1)-\nu}\\
&=\; k\,p\cdot(p+q)^{k-1} \;=\; k\,p\;.
\end{aligned}$$

The other equality can be proved analogously, which yields Claim 1.

Now, proceeding as above, we obtain

$$\begin{aligned}
E[CONV\;-\;1\,] &= \sum_{\mu=0}^{\infty}\Pr[CONV-1\;>\mu]\\
&\leq\; \lambda+\sum_{\mu=\lambda}^{\infty}\sum_{\nu=0}^{k}\binom{k}{\nu}p^{\nu}q^{k-\nu}\cdot\big(\nu q^{\mu}+(k-\nu)p^{\mu}\big)\\
&=\; \lambda+\sum_{\mu=\lambda}^{\infty}\sum_{\nu=0}^{k}\binom{k}{\nu}p^{\nu}q^{k-\nu}\cdot\nu q^{\mu}+\sum_{\mu=\lambda}^{\infty}\sum_{\nu=0}^{k}\binom{k}{\nu}p^{\nu}q^{k-\nu}\cdot(k-\nu)p^{\mu}\\
&=\; \lambda+\sum_{\mu=\lambda}^{\infty}q^{\mu}\cdot\underbrace{\sum_{\nu=0}^{k}\binom{k}{\nu}p^{\nu}q^{k-\nu}\nu}_{=kp\quad\text{by Claim 1}}+\sum_{\mu=\lambda}^{\infty}p^{\mu}\cdot\underbrace{\sum_{\nu=0}^{k}\binom{k}{\nu}p^{\nu}q^{k-\nu}\cdot(k-\nu)}_{=kq\quad\text{by Claim 1}}\\
&=\; \lambda+kp\cdot\sum_{\mu=\lambda}^{\infty}q^{\mu}+kq\cdot\sum_{\mu=\lambda}^{\infty}p^{\mu} \;=\; \lambda+k\cdot[q^{\lambda}+p^{\lambda}]\\
&\leq\; \lambda+k\;2\;\psi^{-\lambda}\;.
\end{aligned}$$

Finally, choosing $\lambda=\lceil\log_{\psi}k\rceil$ gives the statement of the theorem. □

Now, we can easily determine the expected total learning time.

Corollary 1. *For every binomially distributed text with parameter* $0<p<1$ *the average total learning time of algorithm* $\boldsymbol{\mathcal{P}}$ *for concepts in* $\mathcal{C}_n$ *with* μ *literals is at most* $O(n(\log(n-\mu+2))\,.$

The Wholist algorithm possesses the two favorable properties needed to apply Theorem 1, i.e., it is rearrangement-independent and conservative. Thus, we can conclude

$\Pr[CONV>2\,t\cdot E[CONV]\leqslant 2^{-t}$ for all $t\in\mathbb{N}\,.$

Next, we turn our attention to the design of a stochastic finite learner. We study the case that the positive examples are binomially distributed with parameter p. But we do not require precise knowledge about the underlying distribution. Instead, we reasonably assume that *prior knowledge* is provided by parameters p_{low} and p_{up} such that $p_{low} \leqslant p \leqslant p_{up}$ for the true parameter p. Binomial distributions fulfilling this requirement are called $\boldsymbol{(p_{low}, p_{up})}$ ***–admissible distributions***. Let $\boldsymbol{\mathcal{D}_n[p_{low}, p_{up}]}$ denote the set of such distributions on $\mathcal{X}_n$.

If bounds p_{low} and p_{up} are available, the Wholist algorithm can be transformed into a stochastic finite learner inferring all concepts with high confidence.

Theorem 3. *Let* $0 < p_{low} \leq p_{up} < 1$ *and* $\psi := \min\{\frac{1}{1-p_{low}}, \frac{1}{p_{up}}\}$. *Then* $(\mathcal{C}_n, \mathcal{D}_n[p_{low}, p_{up}])$ *is stochastically finitely learnable with high confidence from positive presentations. To achieve* δ *-confidence no more than* $O(\log_2 1/\delta \cdot \log_\psi n)$ *many examples are necessary.*

Proof. The learner is based on the Wholist algorithm and a counter for the number of examples already processed. If the Wholist algorithm is run for $\tau := \lceil \log_\psi n \rceil + 3$ many examples Theorem 2 implies that τ is at least as large as the expected convergence stage $E[CONV]$.

In order to achieve the desired confidence, the learner sets $\gamma := \lceil \log \frac{1}{\delta} \rceil$ and runs the Wholist algorithm for a total of $2\,\gamma \cdot \tau$ examples. The algorithm outputs the last hypothesis $h_{2\,\gamma \cdot \tau}$ produced by the Wholist algorithm and stops thereafter. The reliability follows from the tail bounds established in Theorem 1. □

4.1 Average-Case Analysis for Learning in the Limit from Informant

Finally, let us consider how the results obtained so far translate to the case of learning from informant. Since the Wholist algorithm does not learn anything from negative examples, one may expect that it behaves much poorer in this setting. Let us first investigate the uniform distribution over $\mathcal{X}_n$. Again, we have the trivial cases that the target concept is "FALSE" or m is a monomial without irrelevant bits. In the first case, no example is needed at all, while in the latter one, there is only one positive example having probability 2^{-n}. Thus the expected number of examples needed until successful learning is $2^n = 2^{\#(m)}$.

Theorem 4. *Let* $c = L(m) \in \mathcal{C}_n$ *be a nontrivial concept. If a data sequence for* c *is generated from the uniform distribution on the learning domain by independent draws the expected number of examples needed by algorithm* $\boldsymbol{\mathcal{P}}$ *until convergence is bounded by*

$$E[CONV] \;\leq\; 2^{\#(m)}\,(\lceil \log_2 k(m) \rceil + 3)\;.$$

Proof. Let $CONV+$ be a random variable for the number of positive examples needed until convergence. Every positive example is preceded by a possibly empty block of negative examples. Thus, we can partition the initial segment

of any randomly drawn informant read until convergence into $CONV+$ many blocks B_j containing a certain number of negative examples followed by precisely one positive example. Let Λ_j be a random variable for the length of block B_j. Then $CONV = \Lambda_1 + \Lambda_2 + \cdots + \Lambda_{CONV+}$, where the Λ_j are independently identically distributed. In order to compute the distribution of Λ_j, it suffices to calculate the probabilities to draw a negative and a positive example, respectively. Since the overall number of positive examples for c is 2^k with $k = k(m)$, the probability to generate a positive example is 2^{k-n}. Hence, the probability to draw a negative example is $1 - 2^{k-n}$. Consequently,

$$\Pr[\Lambda_j = \mu + 1] \;=\; \left(1 - 2^{k-n}\right)^{\mu} \cdot 2^{k-n} \;.$$

Therefore,

$$\begin{aligned} E[CONV] &= E[\Lambda_1 + \Lambda_2 + \cdots + \Lambda_{CONV+}] \\ &= \sum_{\zeta=0}^{\infty} E[\Lambda_1 + \Lambda_2 + \cdots + \Lambda_\zeta \mid CONV+ = \zeta] \;\cdot\; \Pr[CONV+ = \zeta] \\ &= \sum_{\zeta=0}^{\infty} \zeta \;\cdot\; E[\Lambda_1] \;\cdot\; \Pr[CONV+ = \zeta] \\ &= E[CONV+] \;\cdot\; E[\Lambda_1] \end{aligned}$$

By Markov's inequality, we have $E[CONV+] \leq \lceil \log_2 k \rceil + 3$, and thus it remains to estimate $E[\Lambda_1]$. A simple calculation shows

Lemma 1. *For every* $0 < a < 1$ *holds:*

$$\sum_{\mu=0}^{\infty} (\mu + 1) \cdot a^{\mu} \;=\; (1 - a)^{-2} \;.$$

Using this estimation we can conclude

$$\begin{aligned} E[\Lambda_1] &= \sum_{\mu=0}^{\infty} (\mu + 1) \;\cdot\; \Pr[\Lambda_1 = \mu + 1] \\ &= 2^{k-n} \sum_{\mu=0}^{\infty} (\mu + 1) \cdot \left(1 - 2^{k-n}\right)^{\mu} \;=\; 2^{n-k} \;, \end{aligned}$$

and thus the theorem follows. □

Hence, as long as the length of m is constant, and therefore $k(m) = n - O(1)$, we still achieve an expected total learning time of order $n \log n$. But if $\#(m)$ grows linearly the expected total learning becomes exponential. On the other hand, if there are many relevant literals then even h_0 may be considered as a not too bad *approximation* for c. Consequently, it is natural at this point to introduce an error parameter $\varepsilon \in (0, 1)$ as in the PAC model, and to ask

whether one can achieve an expected sample complexity for computing an ε-approximation that is bounded by a function depending on $\log n$ and $1/\varepsilon$.

To answer this question, let us formally define $error_m(h_j) = D(L(h_j) \triangle L(m))$ to be the error made by hypothesis h_j with respect to monomial m. Here $L(h_j) \triangle L(m)$ stands for the symmetric difference of $L(h_j)$ and $L(m)$ and D for the underlying probability distribution with respect to which the examples are drawn. Note that by Theorem 1 we can conclude $error_m(h_j) = D(L(m) \setminus L(h_j))$. We call h_j an $\boldsymbol{\varepsilon}$**−approximation** for m if $error_m(h_j) \leqslant \varepsilon$. Finally, we redefine the stage of convergence. Let $d = (d_j)_{j \in \mathbb{N}^+}$ be an informant for $L(m)$, then

$$\boldsymbol{CONV_\varepsilon(d)} =_{df} \text{the least number } j \text{ such that } error_m(P(d[i])) \leq \varepsilon \text{ for all } i \geq j.$$

Note that once the Wholist algorithm has reached an ε-approximate hypothesis all further hypotheses will also be at least that close to the target monomial.

The following theorem gives an affirmative answer to the question posed above.

Theorem 5. *Let $c = L(m) \in \mathcal{C}_n$ be a nontrivial concept. Assuming that examples are drawn independently from the uniform distribution, the expected number of examples needed by algorithm $\boldsymbol{\mathcal{P}}$ until converging to an ε-approximation for c can be bounded by*

$$E[CONV_\varepsilon] \;\leq\; \frac{1}{\varepsilon} \cdot (\lceil \log_2 k(m) \rceil + 3) \;.$$

Proof. It holds $error_m(h_0) = 2^{k(m)-n}$, since h_0 misclassifies exactly the positive examples. Therefore, if $error_m(h_0) \leqslant \varepsilon$, we are already done. Now suppose $error_m(h_0) > \varepsilon$. Consequently, $1/\varepsilon > 2^{n-k(m)}$, and thus the bound stated in the theorem is larger than $2^{n-k(m)}(\lceil \log_2 k(m) \rceil + 3)$, which, by Theorem 4 is the expected number of examples needed until convergence to a correct hypothesis. □

Thus, additional knowledge concerning the underlying probability distribution pays off again. Applying Theorem 1 and modifying the stochastic finite learner presented above *mutatis mutandis*, we get a learner identifying ε-approximations for all concepts in $\mathcal{C}_n$ stochastically with high confidence using $O(\frac{1}{\varepsilon} \cdot \log \frac{1}{\delta} \cdot \log n)$ many examples. Comparing this bound with the sample complexity given in the PAC model, we see that it is reduced exponentially, i.e., instead of a factor n now we have the factor $\log n$.

Finally, we can generalize the last results to the case that the data sequences are binomially distributed for some parameter $p \in (0, 1)$. This means that any particular vector containing ν times a 1 and $n - \nu$ a 0 has probability $p^\nu (1-p)^{n-\nu}$ since a 1 is drawn with probability p and a 0 with probability $1 - p$. First, Theorem 4 generalizes as follows.

Theorem 6. *Let $c = L(m) \in \mathcal{C}_n$ be a nontrivial concept. Let m contain precisely π positive literals and τ negative literals. If the labeled examples for c are independently binomially distributed with parameter p and $\psi := \min\{\frac{1}{1-p}, \frac{1}{p}\}$*

then the expected number of examples needed by algorithm $\boldsymbol{\mathcal{P}}$ *until convergence can be bounded by*

$$E[CONV] \;\leq\; \frac{1}{p^{\pi}(1-p)^{\tau}} \left(\lceil \log_{\psi} k(m) \rceil + 3 \right) .$$

Proof. Assuming the same notation as in the proof of Theorem 4, it is easy to see that we only have to recompute $E[\Lambda_1]$, and thus $\Pr[\Lambda_1 = \mu+1]$, too. Since m contains precisely π positive literals and τ negative literals, the probability to draw a positive example is clearly $p^{\pi}(1-p)^{\tau}$, and thus the probability to randomly draw a negative example is $1-p^{\pi}(1-p)^{\tau}$. Consequently,

$$\Pr[\Lambda_1 = \mu+1] \;=\; (1-p^{\pi}(1-p)^{\tau})^{\mu} \cdot p^{\pi}(1-p)^{\tau} \;,$$

and Lemma 1 gives $E[\Lambda_1] = \frac{1}{p^{\pi}(1-p)^{\tau}}$. □

Theorem 5 directly translates into the setting of binomially distributed inputs, too.

Theorem 7. *Let* $c = L(m) \in \mathcal{C}_n$ *be a nontrivial concept. Assume that the examples are drawn with respect to a binomial distribution with parameter* p*, and let* $\psi = \min\{\frac{1}{1-p}, \frac{1}{p}\}$*. Then the expected number of examples needed by algorithm* $\boldsymbol{\mathcal{P}}$ *until converging to an* ε*-approximation for* c *can be bounded by*

$$E[CONV] \;\leq\; \frac{1}{\varepsilon} \cdot (\lceil \log_{\psi} k(m) \rceil + 3) \;.$$

Finally, one can also get stochastic finite approximations with high confidence from informant with an exponentially smaller sample complexity.

Theorem 8. *Let* $0 < p_{low} \leq p_{up} < 1$ *and* $\psi := \min\{\frac{1}{1-p_{low}}, \frac{1}{p_{up}}\}$*. For* $(\mathcal{C}_n, \mathcal{D}_n[p_{low}, p_{up}])$ ε*-approximations are stochastically finitely learnable with* δ*-confidence from informant for any* $\varepsilon,\ \delta \in (0,1)$*.*

For this purpose, $O(\frac{1}{\varepsilon} \cdot \log_2 1/\delta \cdot \log_{\psi} n)$ *many examples suffice.*

5 Learning the Pattern Languages

Finally, we take a look at the class of all pattern languages. The results presented below have been obtained by Rossmanith and Zeugmann [19] and we refer the reader to this paper for all omitted details.

This class is not PAC learnable with respect to every hypothesis space having a membership problem uniformly decidable in polynomial time.

Following Angluin [1] we define patterns and pattern languages as follows. Let $\mathcal{A} = \{0, 1, \dots\}$ be any non-empty finite alphabet containing at least two elements. By $\mathcal{A}^*$ we denote the free monoid over $\mathcal{A}$. The set of all finite non-null strings of symbols from $\mathcal{A}$ is denoted by $\mathcal{A}^+$, i.e., $\mathcal{A}^+ = \mathcal{A}^* \setminus \{\lambda\}$, where λ denotes the empty string. Let $X = \{x_i \mid i \in \mathbb{N}\}$ be an infinite set of variables such that $\mathcal{A} \cap X = \emptyset$. *Patterns* are non-empty strings over $\mathcal{A} \cup X$, e.g.,

01, $0x_0111$, $1x_0x_00x_1x_2x_0$ are patterns. The length of a string $s \in \mathcal{A}^*$ and of a pattern π is denoted by $|s|$ and $|\pi|$, respectively. A pattern π is in *canonical form* provided that if k is the number of different variables in π then the variables occurring in π are precisely $x_0, \ldots, x_{k-1}$. Moreover, for every j with $0 \leqslant j < k-1$, the leftmost occurrence of x_j in π is left to the leftmost occurrence of x_{j+1}. The examples given above are patterns in canonical form. In the sequel we assume, without loss of generality, that all patterns are in canonical form. By Pat we denote the set of all patterns in canonical form.

If k is the number of different variables in π then we refer to π as to a *k-variable pattern*. By Pat_k we denote the set of all *k-variable patterns*. Furthermore, let $\pi \in Pat_k$, and let $u_0, \ldots, u_{k-1} \in \mathcal{A}^+$; then we denote by $\pi[x_0/u_0, \ldots, x_{k-1}/u_{k-1}]$ the string $w \in \mathcal{A}^+$ obtained by substituting u_j for each occurrence of x_j, $j = 0, \ldots, k-1$, in the pattern π. For example, let $\pi = 0x_01x_1x_0$. Then $\pi[x_0/10, x_1/01] = 01010110$. The tuple $(u_0, \ldots, u_{k-1})$ is called a *substitution*. Furthermore, if $|u_0| = \cdots = |u_{k-1}| = 1$, then we refer to $(u_0, \ldots, u_{k-1})$ as to a *shortest substitution*. Let $\pi \in Pat_k$; we define the *language generated by pattern* π by $L(\pi) = \{\pi[x_0/u_0, \ldots, x_{k-1}/u_{k-1}] \mid u_0, \ldots, u_{k-1} \in \mathcal{A}^+\}$. By PAT_k we denote the set of all *k-variable pattern languages*. Finally, $PAT = \bigcup_{k \in \mathbb{N}} PAT_k$ denotes the set of all pattern languages over $\mathcal{A}$.

Our pattern learner will directly output patterns as hypotheses. Note that this hypothesis space is not admissible for PAC learning unless $\mathcal{P} = \mathcal{NP}$. It uses Lange and Wiehagen's [11] pattern language learner as a main ingredient. We consider here learning from positive data only. Again, first an average-case analysis has to be performed. Every string of a particular pattern language is generated by at least one substitution. Therefore, it is convenient to consider probability distributions over the set of all possible substitutions. That is, if $\pi \in Pat_k$, then it suffices to consider any probability distribution D over $\underbrace{\mathcal{A}^+ \times \cdots \times \mathcal{A}^+}_{k-\text{times}}$. For $(u_0, \ldots, u_{k-1}) \in \mathcal{A}^+ \times \cdots \times \mathcal{A}^+$ we denote by $D(u_0, \ldots, u_{k-1})$ the probability that variable x_0 is substituted by u_0, variable x_1 is substituted by u_1, ..., and variable x_{k-1} is substituted by u_{k-1}.

In particular, we mainly consider a special class of distributions, i.e., *product distributions*. Let $k \in \mathbb{N}^+$, then the class of all product distributions for Pat_k is defined as follows. For each variable x_j, $0 \leqslant j \leqslant k-1$, we assume an arbitrary probability distribution D_j over $\mathcal{A}^+$ on substitution strings. Then we call $D = D_0 \times \cdots \times D_{k-1}$ product distribution over $\mathcal{A}^+ \times \cdots \times \mathcal{A}^+$, i.e., $D(u_0, \ldots, u_{k-1}) = \prod_{j=0}^{k-1} D_j(u_j)$. Moreover, we call a product distribution *regular* if $D_0 = \cdots = D_{k-1}$. Throughout this paper, we restrict ourselves to deal with *regular* distributions. We therefore use d to denote the distribution over $\mathcal{A}^+$ on substitution strings, i.e, $D(u_0, \ldots, u_{k-1}) = \prod_{j=0}^{k-1} d(u_j)$. We call a regular distribution *admissible* if $d(a) > 0$ for at least two different elements $a \in \mathcal{A}$.

We will express all estimates with the help of the following parameters: $E[\Lambda]$, α and β, where Λ is a random variable for the length of the examples drawn. α and β are defined below. To get concrete bounds for a concrete implementation one has to obtain c from the algorithm and has to compute $E[\Lambda]$, α, and β

from the admissible probability distribution D. Let $u_0, \ldots, u_{k-1}$ be independent random variables with distribution d for substitution strings. Whenever the index i of u_i does not matter, we simply write u or u'.

The two parameters α and β are now defined via d. First, α is simply the probability that u has length 1, i.e.,

$$\alpha = \Pr(|u| = 1) = \sum_{a \in \mathcal{A}} d(a).$$

Second, β is the conditional probability that two random strings that get substituted into π are identical under the condition that both have length 1, i.e.,

$$\beta = \Pr\big(u = u' \;\big|\; |u| = |u'| = 1\big) = \sum_{a \in cal A} d(a)^2 \Big/ \Big(\sum_{a \in \mathcal{A}} d(a)\Big)^2.$$

Then we can prove the following bound for the stage of convergence for every target pattern from Pat_k.

Theorem 9. *$E[CONV] = O\left(\frac{1}{\alpha^k} \cdot \log_{1/\beta}(k)\right)$ for all $k \geqslant 2$.*

The expected total learning time can be estimated by $O\left(\frac{1}{\alpha^k} E[\Lambda] \log_{1/\beta}(k)\right)$ for all $k \geqslant 2$.

The transformation of Lange and Wiehagen's [11] pattern language learner into a stochastic finite learner is the performed *mutatis mutandis*. The prior knowledge needed is now assumed concerning a lower bound for α and an upper bound for β. More precisely, we have the following theorem.

Theorem 10. *Let $\alpha_*,\ \beta_* \in (0,1)$. Assume $\mathcal{D}$ to be a class of admissible probability distributions over $\mathcal{A}^+$ such that $\alpha \geqslant \alpha_*$, $\beta \leqslant \beta_*$ and $E[\Lambda]$ finite for all distributions $D \in \mathcal{D}$. Then $(PAT, \mathcal{D})$ is stochastically finitely learnable with high confidence from text.*

The latter theorem allows a nice corollary which we state next. Making the same assumption as done by Kearns and Pitt [10], i.e., assuming the additional prior knowledge that the target pattern belongs to Pat_k, the complexity of the stochastic finite learner given above can be considerably improved. The resulting learning time is linear in the expected string length, and the constant depending on k grows only exponentially in k in contrast to the doubly exponentially growing constant in Kearns and Pitt's [10] algorithm. Moreover, in contrast to their learner, our algorithm learns from positive data only, and outputs a hypothesis that is correct for the target language with high probability.

Again, for the sake of presentation we shall assume $k \geqslant 2$. Moreover, if the prior knowledge $k = 1$ is available, then there is also a much better stochastic finite learner for PAT_1 (cf. [18]).

Corollary 2. *Let $\alpha_*,\ \beta_* \in (0,1)$. Assume $\mathcal{D}$ to be a class of admissible probability distributions over $\mathcal{A}^+$ such that $\alpha \geqslant \alpha_*$, $\beta \leqslant \beta_*$ and $E[\Lambda]$ finite for*

all distributions $D \in \mathcal{D}$ *. Furthermore, let* $k \geqslant 2$ *be arbitrarily fixed. Then there exists a learner* M *such that*

(1) M *learns* $(PAT_k, \mathcal{D})$ *stochastically finitely with high confidence from text, and*
(2) *The running time of* M *is* $O\big(\hat{\alpha}_*^k E[\Lambda] \log_{1/\beta_*}(k) \log_2(1/\delta)\big)$ *.*
(* *Note that* $\hat{\alpha}_*^k$ *and* $\log_{1/\beta_*}(k)$ *now are constants.* *)

Further results have been obtained for the class of all one-variable pattern languages. Note that even this class is not PAC learnable as shown in [14]. For the class of all one-variable pattern languages our stochastic finite learner has a total learning time that is essentially linear in the length of the target pattern (cf. [16,18]). Moreover, this stochastic finite learner works for all meaningful probability distributions.

These results provide at least some evidence that our new approach may lead to a new generation of learning algorithms that combine a feasible total learning time and proven performance guarantees.

References

1. D. Angluin, Finding Patterns common to a Set of Strings, *Journal of Computer and System Sciences* **21** (1980), 46–62.
2. A. Blumer, A. Ehrenfeucht, D. Haussler and M. Warmuth, Learnability and the Vapnik-Chervonenkis Dimension, *Journal of the ACM* **36** (1989), 929–965.
3. J. Case, S. Jain, S. Lange and T. Zeugmann, Incremental Concept Learning for Bounded Data Mining, *Information and Computation* **152**, No. 1, 1999, 74–110.
4. R. Daley and C.H. Smith. On the Complexity of Inductive Inference. *Information and Control* **69** (1986), 12–40.
5. T. Erlebach, P. Rossmanith, H. Stadtherr, A. Steger and T. Zeugmann, Learning one-variable pattern languages very efficiently on average, in parallel, and by asking queries, *Theoretical Computer Science* **261**, No. 1-2, 2001, 119–156.
6. E.M. Gold, Language identification in the limit, *Information and Control* **10** (1967), 447–474.
7. S.A. Goldman, M.J. Kearns and R.E. Schapire, Exact identification of circuits using fixed points of amplification functions. *SIAM Journal of Computing* **22**, 1993, 705–726.
8. D. Haussler, Bias, version spaces and Valiant's learning framework. "Proc. 8th National Conference on Artificial Intelligence" (pp. 564–569). Morgan Kaufmann, 1987.
9. D. Haussler, M. Kearns, N. Littlestone and M.K. Warmuth, Equivalence of models for polynomial learnability. *Information and Computation* **95** (1991), 129–161.
10. M. Kearns L. Pitt, A polynomial-time algorithm for learning k –variable pattern languages from examples. "Proc. Second Annual ACM Workshop on Computational Learning Theory" (pp. 57–71). Morgan Kaufmann, 1989.
11. S. Lange and R. Wiehagen, Polynomial-time inference of arbitrary pattern languages. *New Generation Computing* **8** (1991), 361–370.
12. S. Lange and T. Zeugmann, Set-driven and Rearrangement-independent Learning of Recursive Languages, *Mathematical Systems Theory* **29** (1996), 599–634.

13. S. Lange and T. Zeugmann, Incremental Learning from Positive Data, *Journal of Computer and System Sciences* **53**(1996), 88–103.
14. A. Mitchell, A. Sharma, T. Scheffer and F. Stephan, The VC-dimension of Subclasses of Pattern Languages, *in* "Proc. 10th International Conference on Algorithmic Learning Theory," (O. Watanabe and T. Yokomori, Eds.), Lecture Notes in Artificial Intelligence, Vol. 1720, pp. 93 - 105, Springer-Verlag, Berlin, 1999.
15. L. Pitt, Inductive Inference, DFAs and Computational Complexity, *in* "Proc. 2nd Int. Workshop on Analogical and Inductive Inference" (K.P. Jantke, Ed.), Lecture Notes in Artificial Intelligence, Vol. 397, pp. 18–44, Springer-Verlag, Berlin, 1989.
16. R. Reischuk and T. Zeugmann, *Learning One- Variable Pattern Languages in Linear Average Time, in* "Proc. 11th Annual Conference on Computational Learning Theory - COLT'98," July 24th - 26th, Madison, pp. 198–208, ACM Press 1998.
17. R. Reischuk and T. Zeugmann, A Complete and Tight Average-Case Analysis of Learning Monomials, *in* "Proc. 16th International Symposium on Theoretical Aspects of Computer Science," (C. Meinel and S. Tison, Eds.), Lecture Notes in Computer Science, Vol. 1563, pp. 414–423, Springer-Verlag , Berlin 1999.
18. R. Reischuk and T. Zeugmann, An Average-Case Optimal One-Variable Pattern Language Learner, *Journal of Computer and System Sciences* **60**, No. 2, 2000, 302–335.
19. P. Rossmanith and T. Zeugmann. Stochastic Finite Learning of the Pattern Languages, *Machine Learning* **44**, No. 1-2, 2001, 67–91.
20. L.G. Valiant, A Theory of the Learnable, *Communications of the ACM* **27** (1984), 1134–1142.
21. R. Wiehagen and T. Zeugmann, Ignoring Data may be the only Way to Learn Efficiently, *Journal of Experimental and Theoretical Artificial Intelligence* **6** (1994), 131–144.
22. T. Zeugmann, Lange and Wiehagen's Pattern Language Learning Algorithm: An Average-case Analysis with respect to its Total Learning Time, *Annals of Mathematics and Artificial Intelligence* **23**, No. 1-2, 1998, 117–145.

Sequential Sampling Algorithms: Unified Analysis and Lower Bounds

Ricard Gavaldà [1, ⋆] and Osamu Watanabe [2, ⋆⋆]

[1] Universitat Politècnica de Catalunya, Barcelona, Spain.
[2] Tokyo Institute of Technology, Tokyo, Japan.

Abstract. Sequential sampling algorithms have recently attracted interest as a way to design scalable algorithms for Data mining and KDD processes. In this paper, we identify an elementary sequential sampling task (estimation from examples), from which one can derive many other tasks appearing in practice. We present a generic algorithm to solve this task and an analysis of its correctness and running time that is simpler and more intuitive than those existing in the literature. For two specific tasks, frequency and advantage estimation, we derive lower bounds on running time in addition to the general upper bounds.

Keywords: Random sampling, sequential sampling, adaptive sampling, Chernoff bounds, Data mining.

1 Introduction

It is a universal phenomenon that data repositories in all domains are growing at an exponential rate. In many applications, the amount of available input data is already so large that not all data can be processed, maybe not even read, within reasonable time. *Random sampling* is one possible strategy that algorithm designers can use to deal with on massive amounts of data.

Typically, a sampling algorithm obtains a random sample from a large database, performs some computation on the sample, and hopes that the result is not too different from what would be obtained by computing over the whole database. Of course, for some computation tasks this approach fails completely; either the result is required to be exact, or there is no way of obtaining even an approximate answer without looking at all data. Often, however, an approximate answer is acceptable, and can be obtained with some confidence by looking at samples of reasonable size.

⋆ E-mail: gavalda@lsi.upc.es. Partially supported by EU EP27150 (Neurocolt II), by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT), by CIRIT 1997SGR-00366, TIC2000-1970-CE, and by the Spanish Government PB98-0937-C04-04 (project FRESCO).

⋆⋆ E-mail: watanabe@is.titech.ac.jp. Supported in part by Grant-in-Aids for Scientific Research on Priority Areas "Discovery Science" (1998–2000) and for Scientific Research (C) (2001–2002) from the Ministry of Education, Science, Sports and Culture of Japan.

K. Steinhöfel (Ed.): SAGA 2001, LNCS 2264, pp. 173–187, 2001.

To make the discussion concrete, we assume from now on that the task is that of estimating some quantity μ implicit in the database. As an easiest example, μ could be the fraction of database records having a certain property. Then our task is to find some approximation $\hat{\mu}$ of μ up to ε that is correct with probability at least $1-\delta$, for given parameters ε and δ. Depending on the application, we may require *absolute* approximation, that is, $\mu-\varepsilon \leq \hat{\mu} \leq \mu+\varepsilon$, or *relative* one, that is, $(1-\varepsilon)\mu \leq \hat{\mu} \leq (1+\varepsilon)\mu$. We call this type of approximation (especially, the relative one) a (ε,δ) -*approximation* of μ.

In these situations, large deviation bounds (such as Chernoff or Hoeffding type bounds, or those given by the Central Limit Theorem) may be used to determine a sample size that guarantees such approximations. These bounds have often been used, more or less rigorously, to support sampling strategies in different domains, such as machine learning and data mining [3,6,7,12,17,18]. In some scenarios, remarkable speedup has been reported, sometimes by orders of magnitude. In other works, however, researchers have considered sampling either impractical, or difficult to use. There are, in our opinion, two causes for the negative conclusions:

(1) Hoeffding (additive) type bounds are worst case, in the sense that they apply to all parameter settings. Generally this leads to prohibitive sample size. Furthermore, Hoeffding bounds can only be applied for absolute, not relative approximation.
(2) Chernoff (multiplicative) type bounds provide relative approximations and smaller bounds, but cannot be applied to our problem because they depend on the unknown quantity to be estimated. To make this point clear, suppose that μ is simply the frequency of some event in the database. Chernoff bound states that sample size $O((\varepsilon^2\mu)^{-1} \cdot \ln(\delta^{-1}))$ suffices for obtaining a relative (ε,δ) -approximation of μ. Since μ is the unknown, it is not clear how to apply this formula to determine sample size.

Sequential sampling algorithms try to overcome these difficulties in the following way. Instead of obtaining the whole sample from the start, they collect database records sequentially; from time to time (in the extreme case, at every step) they decide whether they have already seen enough records to reach a safe conclusion, and if so, they stop. Since, in practice, we are not often in the worst-case situation, these algorithms usually stop much before their worst-case bound. They have also been called *adaptive* [4,5] because their performance adapts to the complexity of the particular problem instance being solved.

Several researchers in the machine learning and the KDD communities have used heuristics based on stopping sampling when some conclusion seems very likely [12,17,18]. On the other hand, statisticians have worked extensively on sequential sampling since the 40s (Wald's work is classical [22]; see, e.g., [9] for recent works). But they focused mostly on minimizing the number of experiments for hypothesis testing.

Only very recently, some sequential sampling algorithms have been developed that (i) can be applied to current, general, KDD tasks and that (ii) have theoretical guarantees of correctness and performance [4,5,19,20,21]. In these works, the

algorithms have been experimentally tested against those using non-sequential sampling, or no sampling at all, and have systematically outperformed them. Typically, the performance of these algorithms depends on the quality of the data rather than the size of the database; so their advantage is expected to be more apparent as databases grow.

We thus advocate for trying to apply sequential sampling algorithms in more and more situations. This is difficulted by at least two factors. First, the analyses in the literature (e.g., [5,20,21]) are somewhat complicated and for the particular form of the task that is being solved, instead of reflecting clearly the intuition of why adaptive sampling works; thus, when a new task has to be solved most of the analysis has to be done again. Second, there are few theoretical tools to determine how good a given sequential sampling algorithm is for some task, that is, whether it is about as fast as possible or whether a large improvement is possible by more clever design; but see [2] for an exception.

This paper contributes to these issues as follows:

1. We isolate a basic sequential sampling task from which many others can be derived; we present a very generic sequential estimator and provide an analysis of the correctness and efficiency of this procedure much more intuitive and simple than those in the literature (e.g., [5,20,21]). The analysis is given in an almost "recipe" form for the use of algorithm designers.
2. We present a couple of case studies (already discussed in the literature) showing the conciseness of analyses obtained this way.
3. We show how this analysis technique may lead to lower bounds on the running time of some sequential sampling techniques. In particular, we prove lower bounds for the running time for (ε, δ)-approximation in so-called *frequency estimation* and *advantage estimation* problems; these lower bounds match (up to doubly log factors) the upper bounds given by our implementations of the generic estimator, which therefore are almost optimal.

2 Previous Work

We review in this section some existing work on sequential sampling applied to Knowledge Discovery in databases. Of course, other fields of computer science have used sampling extensively, too; see, e.g., [2].

Example 1. Lipton et al. [14,15] provide algorithms for estimating the size of queries to the database. Their core algorithm is a sequential sampling method for estimating the proportion p of records in a database that satisfy a certain property. That is, the task is essentially estimating a probability p that a certain event occurs in a sequence of i.i.d. Bernoulli trials, a task we call *frequency estimation*. The target μ is the probability p in this frequency estimation. Their algorithm achieves relative (ε, δ)-approximation in time $O(1/(\varepsilon^2 \mu) \ln(1/\delta))$.

Example 2. In [4,5] we presented an algorithm for a task appearing often in the context of supervised learning: *advantage estimation* for a Boolean classifier. A boolean classifier predicts a 0/1 value on any example. The advantage γ of a

Boolean classifier is its classification accuracy on a set of examples minus $1/2$, i.e., how much better it predicts than random guessing. The target μ is this advantage γ (of a given classifier) in advantage estimation. The algorithm in [5] achieves relative (ε, δ)-approximation in time $O(s \cdot (\ln \ln s + \ln(1/\delta)))$, where $s = 1/(\varepsilon^2 \mu^2)$.

Example 3. In [5], the algorithm in [4] is generalized to approximating any μ that can be obtained applying a smooth function to the average of bounded variables. The algorithm is applied to the *hypothesis selection problem:* selecting a hypothesis from a set H with utility μ that is at most within $(1+\varepsilon)$ factor of the best one. Its running time is $O(s \cdot (\ln \ln s + \ln(|H|/\delta))$, where s is as above.

Example 4. Scheffer and Wrobel [20,21] observed that similar techniques work for any μ for which one can prove deviation bounds that tend to 0 as sample size grows. They use it for the *n-best hypotheses selection* task, i.e., for given ε and δ, select n hypotheses from H such that none of the selected hypothesis is more than (absolute) ε worse than any discarded hypothesis. The running time of their algorithm is in $O(1/(\varepsilon^2) \ln(1/\delta))$, but note that it achieves absolute rather than relative approximation. Additionally, their algorithm includes Hoeffding races [17] to discard soon hypothesis that are unlikely to be among the n best.

3 Our Framework and a Generic Estimator Algorithm

We describe in this section a basic task that we want to solve by means of sequential sampling.

We want to estimate some unknown quantity μ, of which we know only that $0 < \mu < 1$. To do this, we have available an infinite sequence of *examples* $X_1, X_2, \ldots,$ drawn independently from example space X and some function $F : X^\star \mapsto R$ such that, for any t, $E[F(X_1, \ldots, X_t)] = \mu$.

An *estimator algorithm* A performs as follows: At the start, it reads input parameter δ. Then at each (time-)step $t \geq 1$, it reads the value of X_t and produces a pair of real numbers (μ_t, ε_t). Note that, strictly speaking, A is not an algorithm because it never stops.

We say that A is an *estimator for* μ if the following condition holds for all inputs δ,

(a) with probability at least $1 - \delta$, it holds for all t that

$$(1 - \varepsilon_t)\mu \;\leq\; \mu_t \;\leq\; (1 + \varepsilon_t)\mu, \quad \text{and}$$

(b) the sequence of ε_t tends to 0 as t grows.

For condition (a), we should be a bit careful on the order of quantification of δ and t. It requires that only with probability less than δ, there may be *some* step where approximation is not good.

We are interested in efficient estimator algorithms, for which the quantity ε_t tends to 0 as quickly as possible. Quite naturally, an estimator algorithm for μ

can be used to obtain an (ε, δ)-approximation to μ if both ε and δ are given: It is enough to run the estimator with parameter δ, monitor its output until ε_t goes below ε, and then produce μ_t as the estimation.

In this paper, we consider one generic estimator algorithm, which is stated in Figure 3. This algorithm is quite straightforward; it just computes μ_t by using the function F (that is given when specifying the problem) for observed set of examples $X_1, ..., X_t$. It is so natural that one may not be able to think of any other algorithm for solving the type of estimation problem that we are considering. Below we will show "a recipe" to prove the correctness and analyze the running time of this algorithm, depending on the convergence properties of the estimations for μ. We consider only relative approximation, but, if necessary, it is routine to develop another algorithm for absolute approximation. This would correspond to the algorithms in [20,21].

```
procedure GenericEstimator
    input δ;
    t := 0;  S_0 := ∅;
    while (true) do
        t := t + 1;
        S_t := S_{t-1} ∪ {X_t};
        compute μ_t := F(S_t);
        compute ε_t (by some formula obtained in the analysis);
        output (μ_t, ε_t);
    end-while
```

Fig. 1. The Generic Estimator Algorithm

3.1 Implementation and Analysis

Here we explain how to implement our generic algorithm for solving a given estimation problem and how to prove its correctness / efficiency for that problem.

Step 1. Review the literature on large deviation bounds and find a good bound B for which we can prove that, for all α and t,

$$\Pr[\,(1-\alpha)\mu \;\le\; \mu_t \;\le\; (1+\alpha)\mu\,] \;\ge\; 1 - B(\alpha, \mu, t, \dots).$$

Let us just write $B(\alpha, \mu, t)$ even though B may depend on other parameters (such as $\mathrm{Var}(\mu)$, or even S_t itself).

Step 2. Observe that for the sequence α_t defined by

$$B(\alpha_t, \mu, t) \;=\; \delta/t(t+1),$$

the total probability that there is an error at some t is below δ. This is because

$$\begin{aligned}&\Pr[\,\exists t:\ \neg[\,(1-\alpha)\mu \;\le\; \mu_t \;\le\; (1+\alpha)\mu\,]\,]\\ &\quad\le\; \sum_{t\ge 1} B(\alpha, \mu, t) \;\le\; \sum_{t\ge 1} \frac{\delta}{t(t+1)} \;=\; \delta.\end{aligned}$$

Unfortunately, this definition of α_t depends on unknown μ, so we cannot simply define $\varepsilon_t \stackrel{\text{def}}{=} \alpha_t$ in the algorithm. But we can use some kind of "bootstrapping" approach: if we assume that μ_t is quite near μ, plugging it in the equation instead of μ should give good enough results.

Step 3. Assume that statement (1) holds:

$$\forall t : \quad (1-\alpha_t)\mu \ \leq \ \mu_t \ \leq \ (1+\alpha_t)\mu. \tag{1}$$

Assuming that B is decreasing in μ, we have

$$B(\alpha_t, \mu_t/(1+\alpha_t), t) \ \geq \ (\ B(\alpha_t, \mu, t) \ = \ )\ \delta/t(t+1),$$

and, intuitively, this inequality should be quite tight. Assuming that B is also decreasing in α (which is most often the case), solve this inequation for α_t, and obtain an upper bound for α_t in terms of μ_t and t (and possibly other parameters), which we denote by $B'(\mu_t, t)$. Define ε_t to be that upper bound; that is,

$$\alpha_t \ \leq \ B'(\mu_t, t) \stackrel{\text{def}}{=} \varepsilon_t. \tag{2}$$

Step 4 (Conclusion). With probability at least $1-\delta$, the bound (1) holds for *any* $t \geq 1$. Then, so holds bound (2) (for any $t \geq 1$), and therefore

$$\forall t : \quad (1-\varepsilon_t)\mu \ \leq \ \mu_t \ \leq \ (1+\varepsilon_t)\mu. \tag{3}$$

Also, for reasonable bounds B, it should be immediate to check that the sequence ε_t tends to 0. Then we have proved that the algorithm, with this definition of ε_t, is an estimator for μ.

3.2 Efficiency

For given ε and δ, we can use our algorithm to obtain an (ε, δ)-approximation of μ. That is, we run the algorithm until ε_t becomes less than ε. The following procedure can be used to estimate the time used by the algorithm.

(1) We have a definition of ε_t using μ_t and t.
(2) We also have upper and lower bounds for μ_t in terms of μ, ε_t, and t, that hold with probability at least $1-\delta$.
(3) Solve for ε_t in these two equations. This gives bounds for ε_t in terms of μ and t that hold with probability $1-\delta$.
(4) Now demand that ε_t is at most ε, and solve for t. This gives a bound on the time t required/sufficient to reach $\varepsilon_t \leq \varepsilon$ with probability $1-\delta$.

An algorithmic trick to improve this bound on convergence time was introduced in [5]. The idea is to produce a new pair (μ_t, ε_t) not at each iteration, but

update the estimation only at time steps t of the form $t = \lfloor s^i \rfloor$, for some constant $s > 1$. Equivalently, we may view this as follows: the algorithm is not taking one example X_i at each time step, but rather a batch of $s^i - s^{i-1}$ examples at time i. This way, new errors can be introduced at exponentially increasing time steps only, and it is enough to define sequence α_t as $B(\alpha_t, \mu, t) = \frac{\delta}{i(i+1)}$. This results in a new definition of ε_t and in a smaller running time. Typically, this reduces a $\log t$ factor in the running time to some $\log\log t$ factor.

In [5], algorithms using this trick are referred to as *geometric* algorithms, by comparison with *arithmetic* algorithms that check some stopping condition at every step, or after each constant number of steps as in [12]. In [19] sampling schedules for machine learning problems are considered. Independently, the terms *arithmetic* and *geometric* schedules are proposed there to distinguish between essentially the same strategies. Also, geometric schedules are proved to be optimal in some precise sense.

4 Case Study (1): Frequency Estimation

In the following two sections, we present a couple of instances of the estimator and analysis above. The interested reader may want to compare this analysis with the lengthier ones in [4,5,20,21,23].

First we consider frequency estimation, i.e., Example 1 mentioned in Section 2. More specifically, we assume that we have access to a series of i.i.d. variables X_1, X_2, ..., such that each $X_i \in \{0, 1\}$ takes value 1 with probability p, and our goal is to estimate p. That is, our target μ is the probability (or frequency) p of X_i being 1. Clearly, we have $\mu = \mathrm{Exp}[(X_1 + \cdots + X_t)/t]$. Thus, the formula we would use for estimating μ_t is $\mu_t = (X_1 + \cdots + X_t)/t$. (In other words, the function F is defined by $F(X_1, ..., X_t) \stackrel{\text{def}}{=} (X_1 + \cdots + X_t)/t$ here.)

We show that, in this case, the generic estimator leads to a procedure for (ε, δ)-approximation that runs in time $\tilde{O}(\frac{1}{\varepsilon^2 \mu} \ln \frac{1}{\delta})$. (In this paper, we use $\tilde{O}(f)$ to denote $O(f \log f)$.) We also provide a similar lower bound. Recall that the log factor can be reduced to $\log\log$ by the trick discussed above, but we omit its use here for simplicity.

Both for applying our basic estimator algorithm and for deriving a lower bound, we use bounds provided by the Central Limit Theorem (see, e.g., [8]). Here we recall some basic facts about the Central Limit Theorem. Consider any sufficiently large t, and let X_1, X_2, ..., X_t be a series of i.i.d. random variables that take value either 1 or 0 with probability μ and $1-\mu$ respectively. Define $Y_t = \sum_{i=1}^{t} X_i$. Then the expectation and variance of Y_t are $\rho_t = t\mu$ and $\sigma_t = \sqrt{t\mu(1-\mu)}$ respectively. Now the Central Limit Theorem for sums of Bernoulli variables implies the following approximation (if t is large enough).

$$\Pr\left[\left|\frac{Y_t - \rho_t}{\sigma_t}\right| \leq x\right] \approx 1 - 2(1 - \Phi(x)), \tag{4}$$

where $\Phi(x) = \int_{-\infty}^{-x} exp(-z^2)\,dz$. In our analysis, we will assume that this approximation holds. (We remark that the Chernoff bound provides a very similar upper bound, which is in fact provable for all t .)

We use the following upper and lower approximations of $1-\Phi(x)$ [8].

$$\frac{1}{\sqrt{2\pi}}\frac{\exp(-x^2/2)}{x}\left(1-\frac{1}{x^2}\right) \;<\; 1-\Phi(x) \;<\; \frac{1}{\sqrt{2\pi}}\frac{\exp(-x^2/2)}{x}. \tag{5}$$

For $x \geq 1.1$, the $1-1/x^2$ factor can be replaced with 0.17 .

4.1 An Upper Bound

With the bound given by the Central Limit Theorem, we follow the steps in our "recipe" to instantiate the generic algorithm for frequency estimation.

Step 1. We use the bound derived (or more specifically the function $B(\alpha,\mu,t)$) from the righthand inequality of (5). For defining our $B(\alpha,\mu,t)$, we first obtain an appropriate x so that the following relation holds.

$$[\,(1-\alpha)\mu \;\leq\; \mu_t \;\leq\; (1+\alpha)\mu\,] \;\iff\; \left[\; \left|\frac{Y_t-\rho_t}{\sigma_t}\right| \;\leq\; x \;\right].$$

Recall the following definitions:

$$\begin{array}{lllll} \mu_t &=& (\sum_{i=1}^{t} X_i)/t, & Y_t &= \sum_{i=1}^{t} X_i \;\; \text{(hence, } Y_t = t\mu_t\text{)}, \\ \rho_t &=& t\mu, \quad \text{and} & \sigma_t &= \sqrt{t\mu(1-\mu)}. \end{array}$$

Thus, by using the fact that $(1-\alpha)\mu \;\leq\; \mu_t \;\leq\; (1+\alpha)\mu$ if and only if $|\mu_t-\mu| \leq \alpha\mu$, the above relation can be restated as

$$[\,|\mu_t-\mu| \;\leq\; \alpha\mu\,] \;\iff\; \left[\; |\mu_t-\mu| \;\leq\; x\sqrt{\frac{\mu(1-\mu)}{t}} \;\right].$$

Hence, this relation holds by letting $x = \alpha\sqrt{t\mu/(1-\mu)}$. Then, since we assume the approximation (4), we have

$$\Pr[\,(1-\alpha)\mu \;\leq\; \mu_t \;\leq\; (1+\alpha)\mu\,] \;\approx\; 1-2(1-\Phi(\alpha\sqrt{t\mu/(1-\mu)})). \tag{6}$$

Furthermore, using the bound in (5) and assuming that $x \geq 1$, we have

$$\begin{aligned} \Pr[\,(1-\alpha)\mu \;\leq\; \mu_t \;\leq\; (1+\alpha)\mu\,] &> 1-\frac{2}{\sqrt{2\pi}}\frac{\exp(-\alpha^2 t\mu/2(1-\mu))}{x} \\ &> 1-\frac{2}{\sqrt{2\pi}}\exp(-\alpha^2 t\mu/2). \end{aligned}$$

Thus, $B(\alpha,\mu,t) \stackrel{\text{def}}{=} \sqrt{2/\pi}\exp(-\alpha^2 t\mu/2)$ satisfies our purpose.

Step 2. Now define α_t so that the following relation holds.

$$B(\alpha_t, \mu, t) \leq \frac{\delta}{t(t+1)}.$$

For this, it is enough to define

$$\alpha_t \stackrel{\text{def}}{=} c\sqrt{\frac{1}{t\mu}\left(\ln\frac{t(t+1)}{\delta} + d\right)},$$

for some $c = 2$ and $d = \ln(\sqrt{\pi/2}) = 0.225\cdots$. (For simplifying our later discussion, we ignore d by using a slightly larger constant for c.)

Step 3. Assume here that $\mu_t \leq (1+\alpha_t)\mu$. Then we have

$$\alpha_t = c\sqrt{\frac{1}{t\mu}\ln\frac{t(t+1)}{\delta}} \leq c\sqrt{\frac{1}{t\frac{\mu_t}{(1+\alpha_t)}}\ln\frac{t(t+1)}{\delta}},$$

or

$$\frac{\alpha_t}{\sqrt{1+\alpha_t}} \leq c\sqrt{\frac{1}{t\mu_t}\ln\frac{t(t+1)}{\delta}}.$$

Since $\alpha_t < 1$ in all cases of interest, we have $\sqrt{1+\alpha_t} < 1.41\cdots$, and we can ignore the $\sqrt{1+\alpha_t}$ factor by taking a slightly larger constant for c. Then, define ε_t to be the right-hand side of this inequality and we have $\alpha_t \leq \varepsilon_t$.

4.2 A Lower Bound

Here again we assume the approximation (4) due to the Central Limit Theorem. Then we prove that if we use our generic algorithm for frequency estimation, it is impossible to bound ε_t by any function smaller than $c_1\sqrt{(1/t\mu_t)\ln(1/\delta)}$ for some constant c_1. That is, our definition of $\varepsilon_t = \mathcal{O}(\sqrt{(1/t\mu_t)\ln(t^2/\delta)})$ obtained above achieves almost optimal convergence speed (ignoring the difference in the log factor). (We note here that a more general result, including this one as a special case, was proved in [2] with a more complicated technique.)

Theorem 1. *For frequency estimation, suppose that we use the generic algorithm with the formula* $\mu_t = (\sum_{i=1}^{t} X_i)/t$. *Then there is some constant* c_1 *such that for any* $\delta \leq 0.27$, *any* μ, *and any* t, *it must hold that either* $\varepsilon_t > 3/4$ *or*

$$\varepsilon_t > c_1\sqrt{\frac{1}{t\mu_t}\ln\frac{1}{\delta}}$$

in order to guarantee that μ_t *is an* (ε_t, δ)*-approximation of* μ.

Remark. 1. This lower bound result is for our generic algorithm, and it does not deny all possibility of a better algorithm for frequency estimation. But, on the other hand, our generic algorithm is quite general, and in fact, this bound applies so long as the estimation μ_t is calculated as the observed frequency.
2. Note that we discuss the error probability at the t th step. Since it may be possible that errors occurs in earlier steps, the total error probability is larger than our estimation. We might be able to get a better lower bound if we also considered this point.
3. In the following, though very rough estimation, we prove this bound with $c_1 = 1/4$.

Proof. Consider any algorithm obtained from the generic one for frequency estimation. Suppose that, for a given $\delta \leq 0.27$, it outputs (ε_t, μ_t) at the t th step. We assume that $\varepsilon_t \leq 3/4$ (since otherwise, the theorem trivially holds). Furthermore, we assume that μ_t is an (ε_t, δ)-approximation of μ, or more precisely, the following holds.

$$\delta \ \geq \ \Pr[\, \neg[\, (1-\varepsilon_t)\mu \ \leq \ \mu_t \ \leq \ (1+\varepsilon_t)\mu \,]\,].$$

From the approximation (6), this implies that

$$\delta \ \geq \ 2(1-\Phi(\varepsilon_t\sqrt{t\mu/(1-\mu)})).$$

Let $x_t = \varepsilon_t\sqrt{t\mu/(1-\mu)}$.

First we note that $x_t \geq 1.1$. Because otherwise $2(1-\Phi(x_t)) \ > \ 2(1-\Phi(1.1)) \ = \ 0.27133\ldots \ > \ 0.27$, contradicting our assumption that $\delta \leq 0.27$. Now since $x_t \geq 1.1$, by using the left-side inequality of (6), we have

$$\delta \ \geq \ 2(1-\Phi(x_t)) \ \geq \ \frac{1}{\sqrt{2\pi}}\frac{\exp(-x_t^2/2)}{x_t}\left(1-\frac{1}{x_t^2}\right) \ \geq \ \frac{1}{0.17\sqrt{2\pi}}\frac{\exp(-x_t^2/2)}{x_t}.$$

Hence, for some constant d $(= 1/(0.17\sqrt{2\pi} = 2.34\ldots)$, we have

$$\frac{d}{\delta} \ \leq \ x_t \cdot \exp(x_t^2/2).$$

Since we may assume that $d/\delta > 8$, it is then easy to see that x_t must satisfy the following bound.

$$x_t \ > \ 2\sqrt{\ln\frac{d}{\delta} - 2\ln\ln\frac{d}{\delta}} \ > \ \frac{1}{2}\sqrt{\ln\frac{1}{\delta}}.$$

Hence,

$$\varepsilon_t \ = \ \frac{x_t}{\sqrt{t\mu(1-\mu)}} \ > \ \frac{1}{2}\sqrt{\frac{1}{t\mu(1-\mu)}\ln\frac{1}{\delta}} \ > \ \frac{1}{2}\sqrt{\frac{1}{t\mu}\ln\frac{1}{\delta}}$$

Here we may assume that $\mu_t \ \geq \ (1-\varepsilon_t)\mu$ holds. (Otherwise, μ_t clearly does not satisfy the approximation condition.) Then since $\mu \ \leq \ \mu_t/(1-\varepsilon_t)$, by using the assumption that $\varepsilon_t \leq 3/4$, we have

$$\varepsilon_t \ > \ \frac{1}{2}\sqrt{\frac{1}{t\mu}\ln\frac{1}{\delta}} \ > \ \frac{1}{2}\sqrt{\frac{1-\varepsilon_t}{t\mu_t}\ln\frac{1}{\delta}} \ > \ \frac{1}{4}\sqrt{\frac{1}{t\mu_t}\ln\frac{1}{\delta}}.$$

This completes the proof. □

4.3 Efficiency

For given ε and δ, let us estimate the time sufficient/necessary for obtaining an (ε, δ)-approximation of frequency μ by using our generic algorithm.

First consider the algorithm we instantiated in Section 4.1 and derive an upper bound. From the definition of ε_t obtained in Section 4.1 and under the assumption that $\mu_t \geq (1-\varepsilon_t)\mu$, it is easy to show that $\varepsilon_t \leq \varepsilon$ holds when

$$\frac{t}{\ln(t^2/\delta)} \approx \frac{t}{\ln(t(t+1)/\delta)} \geq \frac{c^2}{\varepsilon^2\mu}.$$

We observe that $x \geq 2z\ln(zy) \geq z\ln(z^2y)$ implies $x/\ln(x^2y) \gtrsim z$; using this approximation, we can roughly bound the time t sufficient for having $\varepsilon_t < \varepsilon$ as

$$t \leq \frac{2c^2}{\varepsilon^2\mu}\ln\frac{c^2}{\varepsilon^2\mu\delta} = \frac{2c^2}{\varepsilon^2\mu}\left(\ln\frac{1}{\delta} + \ln\frac{c^2}{\varepsilon^2\mu}\right).$$

That is, with probability $\geq 1-\delta$, we have $\varepsilon_t \leq \varepsilon$ by the time t reaches the bound above.

Here it should be noted that with an analysis quite specific to frequency estimation, Lipton and Naughton [15] gave an algorithm that runs in time $O(\frac{1}{\varepsilon^2\mu}\ln\frac{1}{\delta})$, i.e., a log factor faster than this algorithm. See [23] for some details and reasoning of this difference.

Next we show some lower bound from Theorem 1. Consider any instantiation of the generic algorithm, and let (ε_t, μ_t) be its output at the t th step. The theorem claims that, for any step t, it must hold that $\varepsilon_t \geq c_1(\sqrt{(1/t\mu_t)\ln(1/\delta)})$ in order to guarantee that μ_t is an (ε_t, δ)-approximation of μ. Then clearly, to obtain an (ε, δ)-approximation of μ, it must hold that

$$\varepsilon \geq \varepsilon_t \geq c_1\sqrt{\frac{1}{t\mu_t}\ln\frac{1}{\delta}} \geq c_1\sqrt{\frac{1}{t(1+\varepsilon)\mu}\ln\frac{1}{\delta}}.$$

From this and using the trivial bound $\varepsilon < 1$, we have the following lower bound.

$$t \geq \frac{c_1^2}{2\varepsilon^2\mu}\ln\frac{1}{\delta}.$$

5 Case Study (2): Advantage Estimation

In advantage estimation, i.e., Example 2 stated in Section 2, an algorithm has access to a series of i.i.d. variables X_1, X_2, ... such that each X_i takes only values 0 or 1 and $\mathrm{Exp}[X_i] = p$. Here we assume that $p > 1/2$. Our goal is to estimate the advantage $\mu = p - 1/2$. The formula we use in the generic algorithm is $\mu_t = (X_1 + \cdots + X_t)/t - 1/2$.

5.1 An Upper Bound

Step 1. Observe that we cannot apply Chernoff or (simple) Hoeffding bounds directly because μ is not the average of a sum of $0/1$ variables. On the other hand, defining $p_t = (\sum_{i=1}^{t} X_i)/t$, we have

$$(1-\alpha)\mu \le \mu_t \le (1+\alpha)\mu \iff p - \alpha\mu \le p_t \le p + \alpha\mu.$$

Thus, by applying Hoeffding bound now, we have

$$\Pr[\ (1-\alpha)\mu \le \mu_t \le (1+\alpha)\mu\ ] = \Pr[\ p-\alpha\mu \le p_t \le p+\alpha\mu\ ] \ge 1-2\exp(-2\alpha^2\mu^2 t).$$

Hence we define $B(\alpha,\mu,t) \stackrel{\text{def}}{=} 2\exp(-2\alpha^2\mu^2 t)$.

Notice here that the argument from now on works for any other μ for which a similar Hoeffding bounding can be applied. This is the case considered in [5].

Step 2. To have $B(\alpha_t,\mu,t)\ (=2\exp(-2\alpha_t^2\mu^2 t))\ =\ \delta/t(t+1)$, we define

$$\alpha_t \stackrel{\text{def}}{=} \sqrt{\frac{1}{2\mu^2 t}\ln\frac{t(t+1)}{\delta}}.$$

Step 3. Assuming that $\mu_t \le (1+\alpha_t)\mu$, we have

$$\alpha_t \ \le\ \sqrt{\frac{1}{2\left(\frac{\mu_t}{(1+\alpha_t)}\right)^2 t}\ln\frac{t(t+1)}{\delta}},$$

hence,

$$\frac{\alpha_t}{(1+\alpha_t)} \ \le\ \sqrt{\frac{1}{2\mu_t^2 t}\ln\frac{t(t+1)}{\delta}}.$$

Now define β_t to be the right-hand side of this inequality. Then by simple calculations, we have

$$\alpha_t/(1+\alpha_t) \le \beta_t \iff \alpha_t \le \beta_t/(1-\beta_t).$$

Thus, we define $\varepsilon_t \stackrel{\text{def}}{=} \beta_t/(1-\beta_t)$. This gives us an instance of the generic algorithm.

5.2 A Lower Bound

Theorem 2. *For advantage estimation, suppose that we use the generic algorithm with the formula* $\mu_t = (\sum_{i=1}^{t} X_i)/t - 1/2$. *Then there is a constant* c_2 *such that for any* $\delta \le 0.27$, *any* μ, *and any* t, *it must hold that either* $\varepsilon_t > 1/4$ *or*

$$\varepsilon_t \ >\ c_1\sqrt{\frac{1}{t\mu_t^2}\ln\frac{1}{\delta}}$$

in order to guarantee that μ_t *is an* (ε_t,δ)*-approximation of* μ.

Proof. We work by contradiction. Assume that the theorem is false, i.e., some instantiation of the generic algorithm for advantage estimation does better than stated in the theorem. Then we will show that some instantiation of the generic algorithm for frequency estimation contradicts the lower bound in Theorem 1.

In order to distinguish the pairs (μ_t, ε_t) produced by the two instances of the algorithm, we will use symbol μ to denote an advantage, μ_t and ε_t for the output of the advantage estimator, p for a frequency, and p_t and θ_t for the output of the frequency estimator.

Assume that the body of the advantage estimator computes μ_t and ε_t as:

$$\begin{array}{l}\text{compute } \mu_t := (\sum_{i=1}^{t} X_i) - 1/2\,; \\ \text{compute } \varepsilon_t := f(S_t)\,;\end{array}$$

for some function f; then instantiate the frequency estimator algorithm with the assignments:

$$\begin{array}{l}\text{compute } p_t := (\sum_{i=1}^{t} X_i)\,; \\ \text{compute } \theta_t := \text{if } f(S_t) > 1/4 \text{ then } 1 \text{ else } 4 \cdot (p_t - 1/2) \cdot f(S_t)\,;\end{array}$$

Observe that, for any fixed sequence of random examples, we have $p_t = \mu_t + 1/2$, and either $\theta = 1$ (if $\varepsilon_t > 1/4$), or $\theta_t = 4\mu_t\varepsilon_t$ (if $\varepsilon_t \le 1/4$).

We show first that, with probability $1-\delta$, p_t is a (θ_t, δ)-approximation of p. Indeed, by assumption μ_t is a (ε_t, δ)-approximation of μ, i.e., with probability $1-\delta$,

$$(1-\varepsilon_t)\mu \le \mu_t \le (1+\varepsilon_t)\mu$$

As noted in the start of Section 5.1, this is equivalent to

$$p - \varepsilon_t\mu \le p_t \le p + \varepsilon_t\mu.$$

Suppose first $\varepsilon_t > 1/4$. Then $\theta_t = 1$ and $(1-\theta_t)p \le p_t \le (1+\theta)p$ holds because we are assuming $p > 1/2$ for this proof. Otherwise, $\varepsilon_t \le 1/4$; then $\theta = 4\varepsilon_t\mu_t$ and also $\mu_t \ge (1-\varepsilon_t)\mu > \mu/2$. Therefore, applying the definition of θ_t and $p > 1/2$,

$$(1-\theta_t)p = p - \theta_t p = p - 4\mu_t\varepsilon_t \cdot p \le p - 4(\mu/2)\varepsilon_t \cdot (1/2) = p - \mu\varepsilon_t \le p_t$$

and, similarly, $(1+\theta_t)p \ge p_t$.

On the other hand, assume that for constant $c_2 = c_1/4$ and some $\delta \le 0.27$, μ, and t, we have both $\varepsilon_t \le 1/4$ and $\varepsilon_t \le c_2\sqrt{(1/t\mu_t^2)\ln(1/\delta)}$. Then, $\theta_t = 4\mu_t\varepsilon_t \le \mu_t < 3/4$. Also, $\theta_t = 4\mu_t\varepsilon_t \le 4c_2\sqrt{(1/t)\ln(1/\delta)} \le 4c_2\sqrt{(1/tp_t)\ln(1/\delta)}$. This contradicts Theorem 1. □

5.3 Efficiency

Proceeding as in Section 4.3, we can show that the time t sufficient/necessary to get an (ε, δ)-approximation of advantage μ by our generic algorithm is

$$\frac{c_2^2}{2\varepsilon^2\mu^2}\ln\frac{1}{\delta} \le t \le \frac{1}{2\varepsilon^2\mu^2}\ln\frac{1}{\varepsilon^2\mu^2\delta}.$$

This is about $1/\mu$ times bigger than the bound obtained for frequency estimation. Again, the log factor can be reduced to log log by the trick described in Section 3.2.

This upper bound was presented in [4], and is implied also by results in [5].

References

1. H. Chernoff, A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations, *Annals of Mathematical Statistics* 23, pp.493–509, 1952.
2. P. Dagum, R. Karp, M. Luby, and S. Ross, An optimal algorithm for monte carlo estimation, *SIAM J. Comput.* Vol. 29(5), pp.1484–1496, 2000.
3. C. Domingo and O. Watanabe, Scaling up a boosting-based learner via adaptive sampling, in *Proc. of Knowledge Discovery and Data Mining* (PAKDD'00), Lecture Notes in Artificial Intelligence 1805, Springer-Verlag, pp.317–328, 2000.
4. C. Domingo, R. Gavaldà, and O. Watanabe, Practical algorithms for on-line selection, in *Proc. of the First Intl. Conference on Discovery Science*, Lecture Notes in Artificial Intelligence 1532, Springer-Verlag, pp.150–161, 1998.
5. C. Domingo, R. Gavaldà, and O. Watanabe, Adaptive sampling methods for scaling up knowledge discovery algorithms, in *Proc. of the Second Intl. Conference on Discovery Science*, Lecture Notes in Artificial Intelligence, Springer-Verlag, pp.172–183, 1999. The final version will appear in *J. Knowledge Discovery and Data Mining* and is also available as research report C-136, Dept. of Math. and Computing Sciences, Tokyo Institute of Technology, from `www.is.titech.ac.jp/research/research-report/C/`.
6. P. Domingos and G. Hulten, Mining high-speed data streams, in *Proc. 6th Intl. Conference on Knowledge Discovery in Databases*, ACM Press, pp.71–80, 2000.
7. P. Domingos and G. Hulten, A general method for scaling up machine learning algorithms and its applications to clustering, in *Proc. 8th Intl. Conference on Machine Learning*, Morgan Kaufmann, pp.106–113, 2001.
8. W. Feller, *An Introduction to Probability Theory and its Applications* (Third Edition), John Wiley & Sons, 1968.
9. B.K. Ghosh, M. Mukhopadhyay, P.K. Sen, *Sequential Estimation*, Wiley, 1997.
10. P. Haas and A. Swami, Sequential sampling, procedures for query size estimation, *IBM Research Report*, RJ 9101 (80915), 1992.
11. W. Hoeffding, Probability inequalities for sums of bounded random variables, *Journal of the American Statistical Association* 58, pp.13–30, 1963.
12. G.H. John and P. Langley, Static versus dynamic sampling for data mining, in *Proc. of the Second Intl. Conference on Knowledge Discovery and Data Mining*, AAAI/MIT Press, pp.367–370, 1996.
13. J. Kivinen and H. Mannila, The power of sampling in knowledge discovery, in *Proc. of the 14th ACM SIGACT-SIGMOD-SIGACT Symposium on Principles of Database Systems* (PODS'94), ACM Press, pp.77–85, 1994.
14. R.J. Lipton, J.F. Naughton, D.A. Schneider, and S. Seshadri, Efficient sampling strategies for relational database operations, *Theoretical Computer Science* 116, pp.195–226, 1993.
15. R.J. Lipton and J.F. Naughton, Query size estimation by adaptive sampling, *Journal of Computer and System Science* 51, pp.18–25, 1995.
16. J.F. Lynch, Analysis and application of adaptive sampling, in *Proc. of the 19th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (PODS'99), ACM Press, pp.260–267, 1999.

17. O. Maron and A. Moore, Hoeffding races: accelerating model selection search for classification and function approximation, in *Advances in Neural Information Processing Systems*, Morgan Kaufmann, pp.59–66, 1994.
18. A.W. Moore and M.S. Lee, Efficient algorithms for minimizing cross validation error, in *Proc. of the 11th Intl. Conference on Machine Learning*, Morgan Kauffman, pp.190–198, 1994.
19. F.J. Provost, D. Jensen, and T. Oates, Efficient Progressive Sampling, in *Proc. of the 5th ACM SIGKDD Intl. Conference on Knowledge Discovery and Data Mining*, ACM Press, pp.23–32, 1999.
20. T. Scheffer and S. Wrobel, A sequential sampling algorithm for a general class of utility criteria, in *Proc. of the 6th ACM SIGKDD Intl. Conference on Knowledge Discovery and Data Mining*, ACM Press, 2000, to appear.
21. T. Scheffer and S. Wrobel, Finding the most interesting patterns in a database quickly by using sequential sampling, Technical Report, University of Magdeburg, School of Computer Science, january 2001.
22. A. Wald, *Sequential Analysis*, John Wiley & Sons, 1947.
23. O. Watanabe, Simple sampling techniques for discovery science, *IEICE Trans. Info. & Systems*, E83-D (1), pp.19–26, 2000.

Approximate Location of Relevant Variables under the Crossover Distribution

Peter Damaschke

Chalmers University, Mathematical and Computing Sciences,
41296 Göteborg, Sweden

Abstract. Searching for genes involved in traits (e.g. diseases), based on genetic data, is considered from a computational learning perspective. This leads to the problem of learning relevant variables of functions from data sampled from a certain class of distributions generalizing the uniform distribution. The Fourier transform of Boolean functions is applied to translate the problem into searching for local extrema of certain functions of observables. We work out the combinatorial structure of this approach and illustrate its potential use.

Keywords: Learning from samples, relevance, Boolean functions, Fourier transform, crossover distribution, genetics, local extrema.

1 Introduction

Gene hunting for complex traits. A hot topic in genetics is searching for genes that are involved in certain traits, such as genetic diseases. We may consider the candidate genes as variables and their alleles (different versions of the same gene) as their values. Whether or not an individual has a specific trait depends on the genotype, that is, on the combination of alleles of the relevant genes for this trait. Hence we may describe a binary trait by a 0,1-valued function of the genotype. In general, traits are not fully determined by the genes, but also influenced by environment, nutrition, etc., or unknown factors. Therefore we consider probabilistic 0,1-valued functions. Our model assumption is that the genotype determines the probability of the considered trait, called its penetrance.

Data on genotypes and corresponding traits in pedigrees have been analyzed for a long time by statistical methods, in order to search for relevant genes. Analysis becomes particularly difficult if a trait is caused by several genes (so-called complex traits, explicitly mentioned in [9] as a computational challenge). The penetrance may be an arbitrarily complicated function, due to interactions between various genes and proteins. We do not restrict attention to any specific modes of inheritance, such as single-gene dominant or recessive traits, etc.

Relevant variables. As in the above problem domain, observable effects or events are often caused by certain combinations of values of r attributes among n candidate attributes (with r typically much smaller than n), and one wishes

K. Steinhöfel (Ed.): SAGA 2001, LNCS 2264, pp. 189–202, 2001.

to conclude from given examples which attributes might be the relevant ones. This type of problem is well-known in the learning theory literature.

The situation can be modeled by an unknown function f of n variables, which properly depends on only r of them. The goal is to learn f, first of all the set of relevant variables, from values $f(x)$ for several x. Depending on the application, numerous versions of this problem arise: The learner may be able to submit freely chosen queries (assignments) x to an oracle that knows f, or the queries are sampled according to some known or unknown distribution.

Here we consider learning of relevant variables of probabilistic Boolean functions from given samples, where a sample is a multiset of arguments $x \in \{0,1\}^n$. That means, for each x, $f(x)$ is a probability distribution on the image set. For 0,1-valued functions f, it is equivalent to say that $f(x) = 1$ with some probability $p(x)$ if assignment x is presented. We call $p(x)$ the penetrance of f for x. We assume that, in a sample being a multiset of assignments x, all $f(x)$ are independent. (It will become apparent in Section 5 that this independence assumption is justified in the intended genetics application.)

Contributions and organization of the paper. We exploit ideas from algorithmic learning theory to derive a structural framework for gene hunting based on parents-children data. We use the Fourier transform of penetrances p to construct guide functions from observable parameters that "point to" the relevant variables in a certain sense. Since the Fourier transform represents arbitrary functions as linear combinations of very simple ones, we finally obtain rather simple guide functions. So this approach is, in principle, applicable to arbitrarily complex traits. One should be aware that we mainly study the combinatorial nature of the problem and leave out the statistical aspects. In order to decouple the structural issues from the statistical ones, we always implicitly assume that our samples are large enough to get sufficiently accurate estimates of observable values. (However we address the sample size in case of the uniform distribution, cf. Theorem 1.)

In Section 2 we recall the well-known relationship between Fourier transform and relevant variables of Boolean functions and give a canonical algorithm for learning the relevant variables under the uniform distribution. Sections 3–5 contain the new contributions, where Section 3 is the core of the paper. It addresses our learning problem under the more general class of crossover distributions. Fourier transform along with some nice properties of the crossover distribution allows us to build efficient search strategies. The combinatorial Theorem 5 is the key result. Using this theorem we can reduce our learning problem to a conceptually simpler one: searching for local extrema of the mentioned guide functions by function value queries (which is interesting for its own). The idea is illustrated in Section 4 by a concrete construction which is then fully treated in case $r = 2$. In Section 5 we propose several ways of applying our approach to gene hunting, which was the original motivation. Full exploration of the approach must be left for future research. Especially the properties of guide functions deserve further study.

Literature. Relevance is a common topic in machine learning and inference, see [1,8,12,17] for some recent contributions and further pointers. Learning relevant variables by queries has been intensively studied under several learning models, e.g. exact learning by adaptive and nonadaptive membership queries [4,5]. (Again, much more references can be found there.) In [11], the model we supposed here is called p-concepts, however the goal there is different from ours.

The use of Fourier transform for learning Boolean functions is well-established, and the interplay between Fourier spectrum and relevance of variables has also been observed several times [2,3,7,14,15], so it is impossible to summarize even the main results here. However most work is focused on the uniform sample distribution.

The crossover distribution is well-known in genetics [10]. For a survey of gene hunting in pedigree data and a problem discussion we refer to [18]. A special aspect of modelling traits is addressed in [13] where the cases with two involved genes have been classified.

2 Relevance and Fourier Transform

The learning problem. IR denotes the set of real numbers. A function $p : \{0,1\}^n \longrightarrow IR$ is called pseudo-Boolean. If $0 \leqslant p(x) \leqslant 1$ for all $x \in \{0,1\}^n$, we may interpret the $p(x)$ as probabilities. Assume that an oracle for p behaves as follows: If x is submitted as a query then the oracle responds 1 and 0 with probability $p(x)$ and $1-p(x)$, respectively. Answers to several queries are completely independent (including the case of repeated submission of the same x). The $p(x)$ are called penetrances. Queries are independently sampled according to some distribution on $\{0,1\}^n$. In the present section we consider the uniform distribution: every x has probability 2^{-n}.

A variable v is called relevant if there exists an assignment x on the other variables such that $p(x,0) \neq p(x,1)$. (Here the last component is the value of v, and x is the assignment on the other $n-1$ variables. To simplify our formalism, we will often use this loose notation if the meaning is clear from the context.) R denotes the set of relevant variables (relevant set, for short) of p.

Imagine that a learner wants to identify R (subject to some error probability) by observing many pairs of queries x and oracle answers $f(x)$. The naive approach (estimate all $p(x)$ by frequencies of response 1 to queries x and compare them) would require a huge sample with $> 2^n$ queries. However, if the learner knows in advance (or has good reasons to believe) that $|R| \leqslant r$ for some $r \ll n$, he can learn a more efficient representation of p form a sample of reasonable size, as we outline now.

Learning via Fourier transform. Let T be any subset of variables. Parity function Q_T is defined by $Q_T(x) = (-1)^{x \cdot T}$, where $x \cdot T$ means the number of 1s that x has in T. It is well-known that the Q_T form an orthonormal basis in the linear space of functions $p : \{0,1\}^n \longrightarrow IR$. The coefficients in $p = \sum_T h_T Q_T$, given by $h_T = 2^{-n} \sum_x Q_T(x)p(x)$, are the Fourier coefficients of p. We omit set symbols in subscripts and write h for $h_\emptyset$, h_v for $h_{\{v\}}$, etc.

The projection $p[T]$ of p to T is defined as follows: For any assignment z on T, $p[T](z)$ is the average $p(x)$ over all x that induce z on T. If the x are drawn from the uniform distribution, $p[T](z)$ is the probability of answer 1 under the condition that the assignment on T is z. We may consider $p[T]$ either as a function of the variables in T, or as a function of all variables with relevant variables in T only; it will not cause confusion if we jump between these views. If T contains only one variable v, we simply write $p[v]$. Note that $p[T] = (\sum_S h_S Q_S)[T] = \sum_S h_S Q_S[T] = \sum_{S \subseteq T} h_S Q_S$.

This equation is used to prove that $R = \bigcup_{h_S \neq 0} S$: Implication "$\subseteq$" is evident. To see "$\supseteq$", observe $p = p[R] = \sum_{S \subseteq R} h_S Q_S$. On the other hand, $p = \sum_S h_S Q_S$. Since the Fourier representation is unique, we get $h_S = 0$ for all $S \not\subseteq R$.

Hence one can learn R under the assumption $|R| \leqslant r$ as follows: Learn the h_T for all sets T, with $|T| = 1, 2, 3, \ldots, r$. Whenever $h_T \neq 0$, add T to the relevant set. Abort when r relevant variables have been found. Hence we have to examine $O(n^r)$ coefficients in the worst case.

To get estimators for the h_T, rewrite the above formula as

$$h_T = \sum_{x|Q_T(x)=+1} p(x)/2^n - \sum_{x|Q_T(x)=-1} p(x)/2^n.$$

The sums are twice the conditional probabilities of answer 1 if $Q_T(x) = +1$ and -1, respectively. Finally replace these two conditional probabilities with the observed frequencies. We may perform a statistical test with null hypothesis $h_T = 0$, that is, we fix a threshold D depending on sample size N, and consider h_T to be 0 if the estimated $|h_T|$ is below D. By routine calculation using the Chernoff bound, the probability that some $h_T = 0$ is misclassified as $h_T \neq 0$ is bounded by an arbitrarily small constant if $N = \Theta(r \log n / D^2)$. The probability that some relevant variable v with $|p(x,0) - p(x,1)| > \delta$ for some x remains undetected can also be made arbitrarily small, choosing $D = \Theta(\delta / 2^r)$. Thus $N = \Theta(r 2^{2r} \log n / \delta^2)$ suffices to exclude both types of error with high probability. It follows:

Theorem 1. *Given a probabilistic Boolean function p with $|R| \leqslant r$, those relevant variables whose change can affect p by more than δ can be learned by a sample of size $\Theta(r 2^{2r} \log n / \delta^2)$ under the uniform distribution, examining $O(n^r)$ Fourier coefficients of subsets of size at most r.*

We remark that the complexity of learning deterministic Boolean functions by learner-chosen assignments is roughly $O(r 2^r \log n)$ [4]. The exponential term in r is not off-putting, as r is small in the intended applications.

Efficiency. If p has particular structural properties, the above algorithm needs much fewer than $O(n^r)$ Fourier coefficients to learn R. We give an example: A function p is called locally monotone if, for every variable v, either all assignments x on the variables $\neq v$ satisfy $p(x,0) \leqslant p(x,1)$, or all x satisfy $p(x,0) \geqslant p(x,1)$.

Theorem 2. *If p is locally monotone then $h_v \neq 0$ for all $v \in R$. Consequently, if the learner knows this property in advance, he can learn R from the $O(n)$ coefficents of singleton sets.*

Proof. Let $v \in R$ and assume for contradiction that $|T| > 1$ for all $T \ni v$ with $h_T \neq 0$. Note that $Q_\emptyset[v] = 1$ and $Q_T[v] = 0$ for all $T \not\subseteq \{v\}$. Hence $p[v] = \sum_T h_T Q_T[v] = h$. That is, $p[v]$ is constant: $p[v](0) = p[v](1)$. Let w.l.o.g. be $p(x,0) \leqslant p(x,1)$ for all assignments x on the variables $\neq v$. Some of these inequalities is strict, due to $v \in R$. But $p[v](i)$ is the average of $p(x,i)$, for $i = 0, 1$ – contradiction. □

One may ask more generally: Given an integer constant t, for which class of functions p can we learn R by examining $O(n^t)$ Fourier coefficients (although it might be $t < r$)? It is natural to consider only function classes being closed under permutation of variables.

Consider the following algorithm ION ("increment only if necessary"):

Let S be the set of relevant variables found so far (initially $S = \emptyset$). Starting with $t := 1$ and $s := 0$, repeat the following until $|S| = r$ or $t > r$: Estimate h_T for all T with $|T \cap S| = s$ and $|T \setminus S| = t$. If $h_T \neq 0$ then let $S := S \cup T$ and reset $t := 1$, $s := 0$. If S did not grow, let $s := s + 1$. If already $s = |S|$, let $t := t + 1$ and reset $s := 0$.

The next theorem says that ION has optimal efficiency, in a certain sense.

Theorem 3. *Let r be fixed and $t \leqslant r$. Let $\mathcal{A}$ be any algorithm that learns R provided that $|R| \leqslant r$, and $\mathcal{C}$ be the class of functions where $\mathcal{A}$ learns R from $O(n^t)$ coefficients h_T. Then ION learns R of every $p \in \mathcal{C}$ after examination of $O(n^t)$ coefficients, too.*

Proof. Consider some p, and let u be the maximum value of t during execution of ION if p is given. It is not hard to see that ION examines less than $2^r n^u$ coefficients, which is $O(n^u)$ for fixed r. Hence, if ION fails to learn R for some p in $O(n^t)$ steps, there must be some T with $h_T \neq 0$ such that $|T \setminus S| > t$, where S denotes the final outcome of ION. Moreover, any such set T has more than t elements outside S. Now assume that $\mathcal{A}$ learns the relevant set of p in $O(n^t)$ steps. Then we can foil $\mathcal{A}$, due to a symmetry argument: Since we may apply any permutation to the variables, particularly to the variables outside S, there are functions in $\mathcal{C}$ where $\mathcal{A}$ needs $\Omega(n^{t+1})$ steps to guess any of these sets. □

3 Relevance and Crossover Distribution

Linkage and crossover distribution. Given linearly ordered variables: $v_1 < \ldots < v_n$, and positive numbers $\theta_i \leqslant \frac{1}{2}$, $i = 1, \ldots, n$, the crossover distribution is defined as follows: $v_1 = 0$ or 1 with probability $\frac{1}{2}$, and $v_{i+1} \neq v_i$ (this is called a switch) with probability θ_i, and all switches are independent. We call θ_i the switch probability of interval $[v_i, v_{i+1}]$. It can be generalized to arbitrary

intervals: Let $i < j < k$. If $[v_i, v_j]$ and $[v_j, v_k]$ have switch probabilities λ and μ, respectively, then $[v_i, v_k]$ has switch probability $(1-\lambda)\mu + \lambda(1-\mu) = \lambda + \mu - 2\lambda\mu$.

The linkage L of an interval $[v_i, v_k]$ with switch probability θ is defined to be $L := 1 - 2\theta$. Variables v_i, v_k are called linked if $L > 0$, and unlinked if $L = 0$. If all variables are unlinked, we have the uniform distribution. Note that $\theta = \frac{1}{2}(1-L)$ and $1 - \theta = \frac{1}{2}(1+L)$. Linkage is multiplicative in the following sense: Consider $i < j < k$. If $[v_i, v_j]$ and $[v_j, v_k]$ have linkage L and M, respectively, then $[v_i, v_k]$ has linkage $1-2(\lambda+\mu-2\lambda\mu) = (1-2\lambda)(1-2\mu) = LM$. Hence $d = -\ln L$ is additive and can be interpreted as a distance, called the linkage distance. Also note that $L = e^{-d}$. (These concepts were developed in the early days of genetics, known under the keyword Haldane map function.) The set of variables is uniquely partitioned into segments of pairwise linked variables, called linkage groups. We can place the variables of a linkage group on a finite segment of the real coordinate axis, with distances according to their linkage distances. Therefore we sometimes refer to variables as points.

Generalizing the Fourier learning algorithm. We want to generalize the algorithm for learning the relevant set R to the case that assignments x come from a crossover distribution. In the rest of this section we derive the combinatorics that will be used, in Section 4, to get fast search strategies for the approximate location of the relevant variables.

Again, the learner observes p restricted to small subsets T of variables and then puts together these partial information to infer R. Note that we do not assume that the switch probabilities are a priori known.

Let $p_T(z)$ be the probability of oracle response 1, under the condition that the assignment on T is z, and let $q_T(z)$ be the probability that the assignment on T is z. (Under the uniform distribution we had $p_T = p[T]$ and $q_T = 2^{-|T|}$.) The $p_T(z)$ and $q_T(z)$ are observable in the sense that they can be estimated well as long as T is small enough, compared to $\log_2$ of the sample size.

Next we study how these observables depend on the function p. Then we can use the result for the inverse problem of learning R and p from the observables. In the sequel, y and z denote assignments on R and T, respectively. We write $q_T(y,z)$ instead of $q_{R\cup T}(y,z)$, as R is fixed. W.l.o.g. let be $R \cap T = \emptyset$, otherwise we may add dummy variables for each element in $R \cap T$, linked to the original variables with linkage 1.

First of all, note that $p_T(z)q_T(z) = \sum_y p(y)q_T(y,z)$, and recall $p = \sum_S h_S Q_S$, $S \subseteq R$. This yields $p_T(z)q_T(z) = \sum_S h_S \sum_y Q_S(y)q_T(y,z)$, where for every S, y refers merely to the assignment induced on S rather than on R.

Let $c(S,T,z) := \sum_y Q_S(y)q_T(y,z)$. We get $p_T(z)q_T(z) = \sum_S h_S c(S,T,z)$, hence $p_T(z)q_T(z)$ is a linear combination of terms coming from the Fourier components of p. Intuitively, $c(S,T,z)$ says how much of Q_S is present in $p_T(z)q_T(z)$. It remains to study the $c(S,T,z)$ for any S,T,z. Let us first look at the $q_T(y,z)$ appearing in $c(S,T,z)$.

Calculating the observables. Let $s = |S|$, $t = |T|$, and $k = s + t - 1$. Consider $S \cup T$ and let $L_1, \ldots, L_k$ be the linkages of the k intervals between neighbored variables in $S \cup T$. We loosely say that *linkage* L_i *is incident to variable* $v \in S \cup T$ if v is one of the endpoints of the i th interval.

If o denotes the all-0 assignment, we get right from the definition of crossover distribution: $2^{k+1} q_T(o,o) = \prod_i (1 + L_i) = \sum_K \prod_{i \in K} L_i$, where K runs over all subsets of $\{1, \ldots, k\}$. We call the $L_K := \prod_{i \in K} L_i$ *linkage products*. This also gives $q_T(y,z)$ for arbitrary y, z. Only some "$+$" in the above formula turn to "$-$": If a variable v changes from 0 to 1 then the (one or two) linkages incident to v change their signs. Hence L_K changes its sign iff K contains exactly one linkage incident to v.

We can also write the $c(S,T,z)$ as linear combinations of linkage products: Let us start with $z = o$ again. Remember that $c(S,T,o) = \sum_y Q_S(y) q_T(y,o)$, and $q_T(o,o) = 2^{-s-t} \sum_K L_K$, where the K are all subsets of $\{1, \ldots, s+t-1\}$. If we change any $v \in S$ then L_K changes its sign in $q_T(y,o)$ iff K contains exactly one linkage incident to v. But the parity of y changes, too, thus:

Lemma 1. *If some* $v \in S$ *is changed then* L_K *changes its sign in* $c(S,T,o)$ *iff* K *contains no or two linkages incident to* v.

All assignments y on S are obtained from $y = o$ by changing variables from 0 to 1. So what is the coefficient of L_K in $c(S,T,o)$? If K contains, for each $v \in S$, exactly one incident linkage then, by Lemma 1, the sign of L_K never changes, thus its coefficient in $c(S,T,z)$ is 2^{-t}. All other L_K have coefficient 0. Namely, there exists $v \in S$ such that K contains no or two linkages incident to v. Consider the pairing of all y where mates differ in the value of v only. These mates have opposite parities, hence the sum of their contributions is 0. We get:

Lemma 2. $c(S,T,o) = 2^{-t} \sum_K L_K$, *where* K *runs over those subsets of linkages which contain, for every* $v \in S$, *exactly one linkage incident to* v.

By a couple of definitions, we can formulate this in a more concise way.

Definition 1. *A* chain *in the ordered set* $S \cup T$ *is an inclusion-maximal subsequence of variables from* S *which is not interrupted by variables from* T. *An* inner chain *is between two variables of* T, *otherwise we speak of an* outer chain. *If* $S \cup T$ *starts or ends with a member of* T, *we also say that it starts or ends with have an* empty outer chain. *An* empty inner chain *is the space between two neighbored members of* T *in the sequence. A linkage product* L_K *is a* chain product *if* K *consists only of linkages incident to the variables of some chain, and satisfies the following:* K *contains every second linkage, and, in case of a nonempty outer chain,* K *contains the outermost linkage. A* big chain product *is a product of chain products, one from every chain.*

Note that we have exactly $t + 1$ chains, that every inner/outer chain has two/one chain product(s), with the understanding that an empty set of linkages has product 1, and that the number of big chain products is thus exactly 2^{t-1}.

The definition is easier than it sounds. An example illustrates the notion: If the elements of T and S are ordered like this: SSSTTSSSST, then we have four chains of size 3,0,4,0, and the chain products are L_1L_3 ; 1, L_4 ; $L_5L_7L_9$, L_6L_8 ; 1.

Now Lemma 2 implies:

Theorem 4. *$c(S,T,o)$ is 2^{-t} times the sum of all big chain products.*

Similarly, we can extend the theorem to arbitrary $c(S,T,z)$, starting from $c(S,T,o)$: If some $v \in T$ changes then the signs of linkages incident to v change, and this holds for every fixed y . Therefore L_K changes its sign in $c(S,T,z)$ iff K contains exactly one linkage incident to v . This proves:

Theorem 5. *$c(S,T,z)$ is 2^{-t} times the sum of all big chain products, with sign $+$ or $-$. The sign of a big chain product is $+/-$ iff it contains, for an even/odd number of $v \in T$ with value 1 , exactly one linkage incident to v . In particular, $c(\emptyset,T,z) = 1/2$.*

Together with $p_T(z)q_T(z) = \sum_S h_S c(S,T,z)$, this allows us to compute all the $p_T(z)q_T(z)$, for arbitrary constellations of T and R . Next we demonstrate the application of Theorem 5 to our learning problem.

4 Design and Use of Guide Functions

Let one variable move. We now outline the idea of using Theorem 5 to construct functions that support efficient search for R . We concentrate on the case that all variables are linked. Extension to several linkage groups is straightforward.

The learner has to observe the $p_T(z)$ and $q_T(z)$ on several carefully chosen sets T . In the following we describe a way of choosing test sets for approaching the relevant variables quickly.

We fix all variables of T but one, and let this variable v "move" on the linkage distance axis, i.e. our sets T differ in the position of v only. Let x be the coordinate of v . We define some real-valued function g with argument x , where g is a linear combination of terms $p_T(z)q_T(z)$ for several fixed assignments z on T .

Remember that the $p_T(z)q_T(z)$ are in turn linear combinations of linkage products, and every linkage is some e^x or e^{-x} , with constant factors. It follows that a function g constructed in this way is piecewise a linear combination of $e^x, e^{-x}, 1$, where the pieces are the intervals between variables from R . Our aim is to assemble g in such a way that its local extrema are positions of variables from R . These local extrema can be found quickly by a binary-search-like technique. Ideally this leads us to all relevant variables in a linkage group, by examining a logarithmic number of observables, rather than $O(n^r)$ under the uniform distribution.

Example of guide functions from small T. It follows a concrete construction of a guide function. We use T with three elements and place two of them at the

leftmost point (w.l.o.g. 0) and the rightmost point of the linkage group, whereas the middle element (point x) is moving.

To calculate the observables in terms of x and p, we first consider a component Q_S of p, for a fixed $S \subseteq R$ with $h_S \neq 0$. The elements of S form two inner chains, thus we have four big chain products. We denote them $b_x(0,0)$, $b_x(0,1)$, $b_x(1,0)$, $b_x(1,1)$, respectively, where the first component says whether the big chain product starts with the first linkage (1) or not (0), and the second component says whether exactly one linkage is incident to x (1) or not (0).

Example: If the ordering is TSSSTSSSST then $b_x(0,0) = L_2L_4L_5L_7L_9$, $b_x(0,1) = L_2L_4L_6L_8$, $b_x(1,0) = L_1L_3L_6L_8$, $b_x(1,1) = L_1L_3L_5L_7L_9$.

Theorem 5 yields $8c(S,T,(0,0,0)) = b_x(1,0) + b_x(1,1) + b_x(0,0) + b_x(0,1)$, and similarly, $8c(S,T,(0,1,0)) = b_x(1,0) - b_x(1,1) + b_x(0,0) - b_x(0,1)$. Defining $g_S(x) := 4c(S,T,(0,0,0)) - 4c(S,T,(0,1,0))$ we get $g_S(x) = b_x(1,1) + b_x(0,1)$. Finally we define $g := \sum_S h_S g_S$. Note that this g is in fact computable from the observables: Due to Section 3 we have:
$g = 4p_T(0,0,0)q_T(0,0,0) - 4p_T(0,1,0)q_T(0,1,0)$.

We continue studying $g_S(x) = b_x(1,1) + b_x(0,1)$ for a fixed S. Let $x_1 < \ldots < x_s$ be the coordinates of the variables in S, and $x_i < x < x_{i+1}$. We easily see

$$g_S(x) = e^{-x_1+x_2-x_3+\ldots+x_{i-1}-x_i+x-x_{i+1}+\cdots} + e^{x_1-x_2+x_3-x_4+\ldots+x_i-x+x_{i+1}-x_{i+2}+\ldots}$$

$$g_S(x) = e^{-x_1+x_2-x_3+\ldots+x_i-x+x_{i+1}-x_{i+2}+\cdots} + e^{x_1-x_2+x_3-x_4+\ldots+x_{i-1}-x_i+x-x_{i+1}+\ldots}$$

for odd and even i, respectively. If we remove the outer elements from T (or put them an infinite linkage distance away) then $g_S(x)$ simplifies to $b_x(0,1)$. Since now $|T| = 1$ and $q_T(0) = q_T(1) = \frac{1}{2}$, we also have $g_S = 2p_T(0) - 2p_T(1) = 4p_T(0) - 2$. Moreover, if s is even then $b_x(0,1) = 0$ for all x. Thus we get the surprising

Proposition 1. *The components $h_S Q_S$ of p with even $|S|$ have no influence at all on the p_T for $|T| = 1$.*

In particular, if p has nonzero Fourier coefficients for even $|S|$ only, we need sets T with $|T| \geqslant 2$ in order to learn any relevant variable. On the other hand, we have:

Proposition 2. *For any S within a linkage group, there exists a guide function g_S with $|T| = 2$ having its local maxima at positions of variables from S.*

Namely, if we remove only the rightmost element from our original T then we obviously get $g_S(x) = b_x(1,1)$ if s is even, and $g_S(x) = b_x(0,1)$ if s is odd, and in both cases g_S has the desired property. (Remember that, in big contrast to this, a nonzero h_S cannot be recognized at all from sets T smaller than S under the uniform distribution.) There remains the problem that p has in general several Fourier components whose guide functions g_S might sum up to a constant, such that we get no local extremum at all. Intuitively this seems to be a sporadic case, but it remains to investigate how to avoid such bad cases

safely, by clever combinations of observables. Below we illustrate this matter in the simplest non-trivial case $r \leqslant 2$.

Detecting two linked relevant variables. We show how the above construction works for $r \leqslant 2$, but it becomes apparent that the method can be extended, with some care, to arbitrary r. We write Q_1 instead of Q_{x_1}, etc.

First let be $r = 1$, $R = \{x_1\}$, and $p = h + h_1 Q_1$. We use $|T| = 1$, so coefficient h doesn't matter, and we simply get $g = h_1 e^{-|x-x_1|}$. Our g has a (sharp) local extremum at x_1. That means, to learn R we only have to learn the local extremum of g by querying $g(x)$ at several points x that can be selected by the learner. This result could be easily obtained without the general theory from Section 3, but already case $r = 2$ would be cumbersome without Fourier decomposition and linkage products, such that their benefits become obvious now.

Let $R = \{x_1, x_2\}$ with $x_1 < x_2$, and $p = h + h_1 Q_1 + h_2 Q_2 + h_{12} Q_{12}$. We start with $|T| = 1$. Due to the preceding discussion we get $g(x) = h_1 e^{-|x-x_1|} + h_2 e^{-|x-x_2|}$. If one of h_1, h_2 is 0 then we are in the situation of case $r = 1$ and can find the other relevant variable as above. Let be $h_1, h_2 \neq 0$. Note that g tends to 0 if x goes to $\pm\infty$. If h_1, h_2 have equal sign then the pieces of g outside interval $[x_1, x_2]$ are strictly monotone. If h_1, h_2 have different signs then g inside $[x_1, x_2]$ is strictly monotone. In both cases we easily conclude that at least one of x_1, x_2 is a local extremum.

It remains case $h_1 = h_2 = 0$, so $p = h + h_{12} Q_{12}$, $h > 0$, $h \geqslant |h_{12}|$. By Proposition 1 we have to trouble a guide function g defined with $|T| = 2$. Our $g(x)$ defined above is:

$$g(x) = \begin{cases} he^{-x} + h_{12} e^{-x+x_1-x_2} & \text{if} \quad x < x_1 \\ he^{-x} + h_{12} e^{-x_1+x-x_2} & \text{if} \quad x_1 < x < x_2 \\ he^{-x} + h_{12} e^{-x_1+x_2-x} & \text{if} \quad x_2 < x \end{cases}$$

Since an estimator for x can be computed from the $q_T(z)$, we may multiply g by e^x to get a somehow nicer guide function in this case:

$$g(x)e^x = \begin{cases} h + h_{12} e^{x_1-x_2} & \text{if} \quad x < x_1 \\ h + h_{12} e^{-x_1+2x-x_2} & \text{if} \quad x_1 < x < x_2 \\ h + h_{12} e^{-x_1+x_2} & \text{if} \quad x_2 < x \end{cases}$$

This function is constant outside $[x_1, x_2]$, such that x_1, x_2 can be obtained quickly by binary search with help of $O(\log n)$ function values.

A property of these guide functions. In the following we review the case $|T| = 1$. Our $g = \sum_S h_S g_S$ defined above satisfies $\lim_{x\to\infty} = \lim_{x\to-\infty} = 0$, and g is piecewise a linear combination of e^x, e^{-x}. In particular $g'' = g$ for all x where g is differentiable, and these are all x except positions of variables in R. Hence g is, between points of R, convex if $g > 0$ and concave if $g < 0$. Hence every local maximum above 0 and every local minimum below 0 indicates a position from R. Moreover, at least one such extremum exists unless the entire

g is constant. If p, and all restrictions of p where the Boolean values of some relevant variables are fixed, have nonzero Fourier components on sets of odd size, then this observation can be used to detect the relevant variables successively. The given p satisifies the mentioned condition e.g. if $h_v \neq 0$ for all single variables v, as it is true for locally monotone p (see Theorem 2), but also in other cases: It is easy to see that the condition is equivalent to the following: The incidence matrix of variables and sets S with $h_S \neq 0$ has full rank over $GF(2)$.

Learning local extrema of a real-valued function g on an ordered domain of n points by function value queries is used to find the position of a relevant variable, once we have constructed a suitable guide function. Basically, we may use Golden section search (see e.g. [6,16]):

Let us call a triple of points (x_1, x_2, x_3) with $x_1 < x_2 < x_3$ *non-monotone* if $g(x_1) \leqslant g(x_2)$ and $g(x_2) \geqslant g(x_3)$, with at least one of these inequalities being strict. If $|g(x)|$ is monotone increasing/decreasing at the left/right end (as it is true for our g above), we immediately get a non-monotone triple to start with: Take the two outermost points and a further point close to the leftmost or rightmost one, with larger $|g(x)|$. Then e.g. the following is well-known:

Theorem 6. *Once we have a non-monotone triple, we can find a local maximum by* $\log_\varphi n$ *queries to function values, with* $\varphi = \frac{1}{2}(\sqrt{5}+1)$*, and this query number is worst-case optimal.*

In our application, the $g(x)$ are noisy, as they come from estimated probabilities. Hence also the comparison results become more and more random when we approach a local extremum. The accuracy depends on the sample size. But at least we can greatly restrict the intervals where relevant variables can be found (by direct inspection) and thus quickly exclude the vast majority of candidates.

5 Application to Genetics

Trivial application. First consider a fixed pair of "parents" with many "children". (Since large sample sizes are required, this first approach is not suitable in human genetics, but e.g. for plants.) Chromosomes come in pairs with homologous genes at the same loci, that is, every individual has two versions of each chromosome. In the parental genotypes, name the chromosomes in each pair arbitrarily by 0 and 1. For each gene, a "child" inherits the allele from chromosome 0 or 1 from each "parent". This immediately defines a randomized Boolean function p for the offspring of this pair, with two variables for each gene. Note that the random genotypes of "children" are independent. If a gene is not involved in the trait then both variables are irrelevant. (However, not all genes being relevant for the trait at all must show up in a particular "family".) Linkage groups correspond to chromosomes, hence, if the candidate genes for the trait are on mutually different chromosomes then the variables are unlinked, such that assignments x are uniformly distributed in the "sibship".

Affected siblings. Consider families with two (or more) children and define another function $p^{(2)}$ as follows. As above, introduce a Boolean variable for each gene and parent, but now assign value 0/1 to a variable if the siblings inherited equal/different alleles. (In case 0 the siblings' alleles are called "identical by descent" in the genetics literature.) Notre that assignments of several disjoint pairs of siblings are independent. Define $p^{(2)}$ to be the probability that both siblings have the trait.

We still restrict attention to the case that the candidate genes are unlinked. Then the assignments defined above follow the uniform distribution, too. Let $R^{(2)}$ be the relevant set of $p^{(2)}$. The theorem below ensures that we do not miss relevant variables when using $p^{(2)}$ instead of p:

Theorem 7. $R^{(2)} = R$.

Proof. Inclusion "$\subseteq$" is evident. Let $v \in R$.

Let $\oplus$ denote the componentwise XOR of two vectors. We have: $p^{(2)}(z) = 2^{-n} \sum_{x \oplus y = z} p(x)p(y) = 2^{-n} \sum_x p(x)p(x \oplus z)$. In the following, the last component of arguments refers to v, whereas the first component is the assignment on all other variables. Then we can write the above formula for the all-0 assignment as $p^{(2)}(o,0) = 2^{-n} \sum_x (p(x,0)^2 + p(x,1)^2)$ and $p^{(2)}(o,1) = 2^{-n} \sum_x 2p(x,0)p(x,1)$. For every x we have $p(x,0)^2 + p(x,1)^2 \geqslant 2p(x,0)p(x,1)$, with strict inequality iff $p(x,0) \neq p(x,1)$. Hence $p^{(2)}(o,0) > p^{(2)}(o,1)$, which proves $v \in R^{(2)}$. □

The last inequality in the proof carries over to any mixture (positive linear combination) of several functions $p^{(2)}$. Hence any relevant gene leads to some relevant variable, even if our sample is from many families with two children, as in human genetics. (Function p from above does not share this property.)

In the case of rare traits it is suitbale to restrict a sample to families with affected first child ("censoring"). Define assignments z by shared alleles as above, and let $p^{(1)}$ be the probability that the second child also has the trait ("siblings relative risk"). Since $p^{(1)}(z) = \frac{\sum_{x \oplus y = z} p(x)p(y)}{\sum_{x \oplus y = z} p(x)} = \frac{\sum_x p(x)p(x \oplus z)}{\sum_x p(x)} = \frac{p^{(2)}(z)}{2^{-n} \sum_x p(x)}$, this function has the same nice property as $p^{(2)}$.

Putting things together: Crossover distribution. Finally we come to the general case that several candidate genes may reside on the same chromosome as well. We briefly lecture some basic facts about inheritance. In the process of meiosis, a cell with only one set of chromosomes (gamete), is produced from a normal cell with double set of chromosomes, and then both parents transmit these gametes which fuse into a complete cell of a new individual. Each chromosome created during meiosis consists alternatingly of segments of both chromosomes of the parent. Every switch from one chromosome to the other one is called a crossover, and crossovers at different points are independent, such that the assignments x defined above follow a crossover distribution, and they are independent for different individuals.

The assignments z from the definition of $p^{(2)}$ also follow the crossover distribution. Furthermore $p_T^{(2)}(o)q_T(o) = \sum_z (p_T(z)q_T(z))^2$. This yields another possibility of constructing guide functions with similar properties as in Section 4, which are even applicable to affected siblings from many families. For example, if we take $|T| = 1$ then, due to Theorem 5, $p_T^{(2)}(o)q_T(o)$ has the form $(c + u(x))^2 + (c - u(x))^2$ where c and $u(x)$ is the term from the even and odd Fourier components, respectively. This equals $2(u(x)^2 + c^2)$, and we have $\lim_{x\to\pm\infty} u(x) = 0$. Moreover $u'' = u$ yields $(u^2)'' = (2uu')' = 2(u'^2 + u^2) > 0$. Hence our function is convex between points of R, such that local maxima indicate relevant variables.

Acknowledgements

Thanks to Olle Nerman (Mathematical Statistics, Chalmers) for drawing my interest to genetic linkage analysis, and to Ed Reingold (Univ. of Illinois) for the pointers to his papers.

References

1. D.A. Bell, H. Wang: A formalism for relevance and its application in feature subset selection, *Machine Learning* 41 (2000), 175-195
2. A. Bernasconi: Mathematical techniques for the analysis of Boolean functions, PhD thesis, Univ. Pisa 1998
3. N. Bshouty, J.C. Jackson, C. Tamon: More efficient PAC-learning of DNF with membership queries under the uniform distribution, *ACM Symp. on Computational Learning Theory COLT'99*, 286-293
4. P. Damaschke: Adaptive versus nonadaptive attribute-efficient learning, *Machine Learning* 41 (2000), 197-215
5. P. Damaschke: Parallel attribute-efficient learning of monotone Boolean functions, *7th Scand. Workshop on Algorithm Theory SWAT'2000*, *LNCS* 1851, 504-512, journal version accepted for *J. of Computer and System Sciences*
6. A.S. Goldstein, E.M. Reingold: A Fibonacci version of Kraft's inequality with an application to discrete unimodal search, *SIAM J. Computing* 22 (1993), 751-777
7. J.C. Jackson: An efficient membership-query algorithm for learning DNF with respect to the uniform distribution, *J. of Comp. and Sys. Sci.* 55 (1997), 414-440
8. G.H. John, R. Kohavi, K. Pfleger: Irrelevant features and the subset selection problem, *11th Int. Conf. on Machine Learning 1994*, Morgan Kaufmann, 121-129
9. D.S. Johnson (ed.): Challenges for Theoretical Computer Science (draft), available at `http://www.research.att.com/~dsj/nflist.html#Biology`
10. S. Karlin, U. Liberman: Classifications and comparisons of multilocus recombination distribution, *Proc. Nat. Acad. Sci. USA* 75 (1979), 6332-6336
11. M.J. Kearns, R.E. Schapire: Efficient distribution-free learning of probabilistic concepts, in: *Computational Learning Theory and Natural Learning Systems*, MIT Press 1994, 289-329 (preliminary version in *FOCS'90*)
12. R. Kohavi: Feature subset selection as search with probabilistic estimates, in: R. Greiner, D. Subramanian (eds.): Relevance, Proc. 1994 AAAI Fall Symposium, 122-126

13. W. Li, J. Reich: A complete enumeration and classification of two-locus disease models, *Human Hereditary* (1999)
14. N. Linial, Y. Mansour, N. Nisan: Constant depth circuits, Fourier transform, and learnability, *J. of ACM* 40 (1993), 607-620
15. Y. Mansour: Learning Boolean functions via the Fourier transform, in: *Theoretical Advances in Neural Computing and Learning*, Kluwer 1994
16. A. Mathur, E.M. Reingold: Generalized Kraft's inequality and discrete k-modal search, *SIAM J. Computing* 25 (1996), 420-447
17. J.C. Schlimmer: Efficiently inducing determinations: a complete and systematic search algorithm that uses optimal pruning, *10th Int. Conf. on Machine Learning 1993*, Morgan Kaufmann, 284-290
18. J.D. Terwilliger, H.H.H. Göring: Gene mapping in the 20th and 21st centuries: statistical methods, data analysis, and experimental design, *Human Biology* 72 (2000), 63-132

Author Index

Behnke, Henning 145

Codognet, Philippe 73

Damaschke, Peter 189
Diaz, Daniel 73

Fitzpatrick, Stephen 49
Fleischhauer, Thomas 107

Gavaldà, Ricard 173
Gottschling, Peter 131

Hromkovič, Juraj 1

Knödler, Kosmas 107
Kolonko, Michael 145

Mahnig, Thilo 33

Meertens, Lambert 49
Meloni, Carlo 91
Mertins, Ulrich 145
Mitterer, Alexander 107
Mühlenbein, Heinz 33

Naumann, Uwe 131

Poland, Jan 107

Sauerhoff, Martin 65
Schnitter, Stefan 145
Sorkin, Gregory B. 117

Watanabe, Osamu 173

Zell, Andreas 107
Zeugmann, Thomas 155
Zuber-Goos, Frank 107