

Lecture Notes in Computer Science

Edited by G. Goos, J. Hartmanis and J. van Leeuwen

1363

J.-K. Hao E. Lutton E. Ronald
M. Schoenauer D. Snyers (Eds.)

Artificial Evolution

Third European Conference AE '97
Nîmes, France, October 22-24, 1997
Selected Papers



Springer

Volume Editors

Jin-Kao Hao

LGI2P, EMA-EERIE

Parc Scientifique Georges Besses, F-30000 Nîmes, France

E-mail: hao@eerie.fr

Evelyne Lutton

INRIA Rocquencourt, Projet FRACTALES

Domaine de Voluceau, B.P. 105, F-78154 Le Chesnay Cedex, France

E-mail: evelyne.lutton@inria.fr

Edmund Ronald

Marc Schoenauer

Centre de Mathématiques Appliquées - Ecole Polytechnique

F-91128 Palaiseau Cedex, France

E-mail: eronald@cmapx.polytechnique.fr

marc.schoenauer@polytechnique.fr

Dominique Snyers

Laboratoire I.A. et Sciences Cognitives, ENST de Bretagne

B.P. 832, F-29285 Brest Cedex, France

E-mail: dominique.snyers@enst-bretagne.fr

Cataloging-in-Publication data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Artificial evolution : third European conference ; selected papers / AE '97,
Nîmes, France, October 22 - 24, 1997. J.-K. Hao ... (ed.). - Berlin ;
Heidelberg ; New York ; Barcelona ; Budapest ; Hong Kong ; London ; Milan ;
Paris ; Santa Clara ; Singapore ; Tokyo : Springer, 1998
(Lecture notes in computer science ; Vol. 1363)
ISBN 3-540-64169-6

CR Subject Classification (1991): F1, F.2.2, I.2.6, I.5.1, G.1.6, J.3

ISSN 0302-9743

ISBN 3-540-64169-6 Springer-Verlag Berlin Heidelberg New York

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

© Springer-Verlag Berlin Heidelberg 1998

Printed in Germany

Typesetting: Camera-ready by author

SPIN 10631811 06/3142 - 5 4 3 2 1 0 Printed on acid-free paper

Preface

The Artificial Evolution conference was originally conceived as a forum for the French-speaking Evolutionary Computation community, but has of late been acquiring a very cosmopolitan audience, with the Asian research community represented by several papers in these proceedings. However, AE remains as intended a small and friendly gathering, which will continue to be held every two years, alternating with PPSN, which is the main meeting-place of the European evolutionary computation community.

Previous AE meets were held in Toulouse and Brest, and the organizing committee yet again maliciously deprived the attendees of the attractions of Parisian night-life, by siting the conference in the sunny city of Nimes, with EERIE graciously doing the hosting.

The invited talk on financial applications of Tabu Search was delivered orally by Stavros Zenios of Cyprus University, although the paper on the foundations of the method included here was authored by his colleague Fred Glover of the University of Colorado. As regards the main body of papers, exactly twice the number published were in fact submitted to the review procedure, and all papers for the main conference were directed to three referees. One referee became the designated “minder” for each paper, and helped us track the status of the revisions during the process leading to publication. In this way, 38 original submissions were winnowed down to 28 oral presentations at the conference, and these in turn, with the adjunction of some of the evolvable hardware and robotics material, gave rise to the 20 papers selected for inclusion in these proceedings.

Some conference attendees had the pleasure of assisting in a real-time version of the refereeing process, by judging the Khepera contest. These tiny robots had already been demonstrated at the 1995 conference by Francesco Mondada, and the manufacturer graciously donated one sample as a prize for the team with the robot evincing the most interesting behavior. Olivier Michel provided his simulator as a training test-bed, and organized the contest; entries were judged on the basis of both simulated and real-world behavior. The description of the winning entry — the Nightwatch — has been included in the proceedings, and the organizing committee wishes to thank the computer staff at EERIE for the use of their facilities, and the entrants for the entertainment they provided to all attendees.

An evolvable hardware workshop was organized at the last minute, or rather in the last month preceeding the conference. Professor Eduardo Sanchez of EPFL demonstrated the Firefly machine, and Olivier Michel showed the 3-dimensional version of his Khepera simulator and described his idea of “internet gateways” by means of which a simulated robot might jump into different simulated environments hosted by various computers on the net. Two full-length papers on the CAM-Brain project spearheaded by Hugo de Garis at ATR were also presented at the workshop and have been included in these proceedings.

The papers selected for this volume have been grouped into the following six sections which broadly reflect the organization of the oral presentations.

1. **Invited Paper** where the state of the art in Scatter Search is described by Fred Glover.
2. **GA Operators:** Devising new genetic operators, be they general-purpose or problem-specific, is a popular research area in evolutionary computation. Jens Gottlieb and Nico Voss analyze two data representation and special operators the context of the satisfiability problem. Cathy Escazut and Philippe Collard introduce the *dreamy GA*, which includes a diversity-preserving mechanism inspired by the biological reality of REM sleep. Peyral et al. introduce explicit memory in the form of two virtual individuals, the *winner* and the *loser*, and examine the effects of social interactions (attraction, avoidance) between these and the individuals of the population. Gusz Eiben introduces SAW, a constraint handling mechanism by means of adaptative penalties, and examines its use in graph coloring.
3. **Applications:** The growing acceptance of evolutionary computation is demonstrated by the steady flow of papers describing applications very far from “pure” optimization. In these proceedings, Cristina Cuenca and Jean-Claude Heudin present an evolutionary user preference learning agent for the Internet. A. Piccolboni and G. Mauri optimize an energy function in order to predict protein folding. Isabelle Servet et al. compare the effectiveness of three optimization methods, namely multiple restart hill-climbing, PBIL, and GAs, for the inverse problem of computing traffic streams in telephone nets. Christine Gaspin and Thomas Schiex applie GAs to the problem of genetic mapping in molecular biology, and test their algorithms on gene data for the *trichogramma brassicae* wasp. Two application studies involve finite automata: Leblanc et al. adress the inverse problem, in particular for automata with fixed points. And Julio Tanomaru generates Turing machines to solve simple arithmetical problems.
4. **Theoretical contributions:** Theoretical understanding seems to lag behind practice in the area of evolutionary computation; however, progress in the analysis of the behavior of the algorithms is now being made. Here Alexandru Agapie employs a Markovian model to derive minimal sufficient convergence conditions for the binary elitist genetic algorithm. Sangyeop Oh and Hyunsoo Yoon use the framework of computational ecosystems to examine punctuated equilibria where an SGA transits suddenly between metastable states. Bart Naudts and Alain Verschoren also examine the search dynamics of the SGA, specifically while solving an NP complete class of problems. Günter Rudolph investigates the influence of mutation distributions other than the conventional normal distribution.
5. **Methodologies:** This section of the proceedings has collected studies reflecting the general experience of the authors with evolutionary computation methods. Eric Dedieu et al. describe *opportunistic emergence* of robotic behaviors in the course of evolution. Ralf Salomon and Peter Eggenberger

define adaptation as the tracking of an optimum that moves with time, and compare the abilities of ES and GA in this respect. Christine Crisan and Heinz Muehlenbein conclude that a small modification of a solution followed by a restart helps search to escape local minima traps in the frequency assignment problem. Rochet et al. investigate the relevance of existing epistasis measures to GA-hardness. Finally, Marc Schoenauer and Leila Kallel attempt to estimate a priori the performance of crossover operators.

6. **Evolvable Hardware and Robotics:** The winning application in the Khepera Contest is presented here, with a robot doing a nightwatchman's rounds, detecting the state of light sources in its environment, and making a status report after each patrol. Two workshop papers from Hugo de Garis's Brain Building Project at ATR have been included here: Felix Gers et al. detail the cellular automaton model underlying the project's efforts, and describe some evolvability simulations, and the architecture of the hardware implementation in progress. A more speculative paper by de Garis et al. explores the issues raised by large scale neural evolvable structures with up to a billion neurons.

At this point, we would like to mention Catherine Dalverny, Isabelle Maurin, and Rachel Henaff at the EERIE in Nimes, and Geo Boléat of the CMAP in Paris, and thank them for their invaluable assistance with the nuts and bolts of organizing, and Aldjia Mazari who helped us escape from L^AT_EXhell. Finally, we would like to thank the EA97 program committee members for the service they rendered to the community by ensuring the high scientific content of the papers presented. The names of these very busy people, who still found time or made time to do the refereeing, are listed on the following page. The following additional referees, also donated their time: B.Ami, P. Brisset, M. Clergue, A. Gaspar, Leila Kallel, Tom Lenaerts, D. Memmi, Jean-Pierre Tillich, and Thomas Unger.

December 97

Jin-Kao Hao,
Evelyne Lutton,
Edmund Ronald,
Marc Schoenauer
and Dominique Snyers.

Artificial Evolution 97 - AE'97

October 22-24 1997

EMA-EERIE, Nîmes, France

EA'97 is the third conference on Evolutionary Computation organized in France. Following EA'94 in Toulouse and EA'95 in Brest, the Conference is sited in Nîmes, a 2000-year-old Roman City of Art and History, 45 km from the beautiful Mediterranean Sea.

EA'97 is hosted by LGI2P, common research laboratory in Computer Science and Production Engineering of the Ecole des Mines d'Alès (EMA) and Ecole pour les Etudes et la Recherche en Informatique et Electronique (EERIE).

Organizing Committee

Jin-Kao Hao (EMA-EERIE Nîmes) - Evelyne Lutton (INRIA Rocquencourt)
Edmund Ronald (CMAP Palaiseau) - Marc Schoenauer (CMAP Palaiseau)
Dominique Snyers (TELECOM Bretagne)

Program Committee

Jean-Marc Alliot (ENAC Toulouse) - Thomas Baeck (ICD Dortmund and Leiden University)
Pierre Bessiere (LIFIA Grenoble) - Paul Bourguin (CREA Palaiseau)
Bertrand Braunschweig (IFP Rueil) - Philippe Collard (I3S Nice)
Michel Cosnard (LIP Lyon) - Marco Dorigo (ULB Bruxelles)
Reinhardt Euler (UBO Brest) - David Fogel (Natural Selection Inc. La Jolla)
Hajime Kita (Tokyo Institute of Technology) - Jean Louchet (ENSTA Paris)
Bernard Manderick (VUB Bruxelles) - Zbigniew Michalewicz (UNCC Charlotte)
Olivier Michel (I3S Nice and EPFL Lausanne) - Francesco Mondada (EPFL Lausanne)
Nick Radcliffe (Quadstone Edinburgh) - Michèle Sebag (LMS Palaiseau)
Gilles Venturini (Université de Tours) - Spyros Xanthakis (Logicom Toulouse)

Highlights

An invited talk by F. Glover and S. Zenios

25 paper presentations

Khepera Contest, organized by Olivier Michel

Evolvable Hardware workshop, organized by Edmund Ronald

Contents

Invited Paper

Fred Glover

A Template for Scatter Search and Path Relinking 13

Genetic Operators

Jens Gottlieb and Nico Voss

Representations, Fitness Functions and Genetic Operators for the Satisfiability Problem 55

Cathy Escazut and Philippe Collard

Genetic Algorithms at the Edge of a Dream 69

**Mathieu Peyral, Antoine Ducoulombier, Caroline Ravisé,
Marc Schoenauer
and Michèle Sebag**

Mimetic Evolution 81

A.E. Eiben and J.K. van der Hauw

Adaptive Penalties for Evolutionary Graph Coloring 95

Applications

Cristina Cuenca and Jean-Claude Heudin

*An Agent System for Learning Profiles in Broadcasting
Applications on the Internet* 109

A. Piccolboni and G. Mauri

*Application of Evolutionary Algorithms to Protein Folding
Prediction* 123

Isabelle Servet, Louise Travé-Massuyès, and Daniel Stern

*Telephone Network Traffic Overloading Diagnosis and
Evolutionary Computation Techniques* 137

Christine Gaspin and Thomas Schiex

Genetic Algorithms for Genetic Mapping 145

B. Leblanc, E. Lutton and J.-P. Allouche <i>Inverse Problems for Finite Automata: a Solution Based on Genetic Algorithms</i>	157
Julio Tanomaru <i>Evolving Turing Machines from Examples</i>	167
Theory	
Alexandru Agapie <i>Genetic Algorithms: Minimal Conditions for Convergence</i>	183
Sangyeop Oh and Hyunsoo Yoon <i>An Analysis of Punctuated Equilibria in Simple Genetic Algorithms</i>	195
Bart Naudts and Alain Verschoren <i>SGA Search Dynamics on Second Order Functions</i>	207
Günter Rudolph <i>Asymptotical Convergence Rates of Simple Evolutionary Algorithms under Factorizing Mutation Distributions</i>	223
Methodologies	
Eric Dedieu, Olivier Lebeltel, and Pierre Bessière <i>Wings Were not Designed to Let Animals Fly</i>	237
Ralf Salomon and Peter Eggenberger <i>Adaptation on the Evolutionary Time Scale: A Working Hypothesis and Basic Experiments</i>	251
Christine Crisan and Heinz Muehlenbein <i>The Frequency Assignment Problem: A Look at the Performance of Evolutionary Search</i>	263
S. Rochet, G. Venturini, M. Slimane and E. M. El Kharoubi <i>A Critical and Empirical Study of Epistasis Measures for Predicting GA Performances: A Summary</i>	275
Leila Kallel and Marc Schoenauer <i>A Priori Comparison of Binary Crossover Operators: No Universal Statistical Measure, but a Set of Hints</i>	287

Evolvable Hardware and Robotics

A. Loeffler, J. Klahold and U. Rueckert

The Dynamical Nightwatch's Problem Solved by the Autonomous Micro-Robot Khepera

303

Felix Gers, Hugo de Garis and Michael Korkin

CoDi-1Bit: A Simplified Cellular Automata Based Neuron Model

315

Hugo De Garis, Lishan Kang, Qiming He, Zhengjun Pan, and Masahiro Ootani

Million Module Neural Systems Evolution — The Next Step in ATR's Billion Neuron Artificial Brain ("CAM-Brain") Project

335

Author Index

349

Invited Paper

A Template for Scatter Search and Path Relinking

Fred Glover

School of Business, CB 419
University of Colorado
Boulder, CO 80309-0419, USA

fred.glover@colorado.edu

Abstract. Scatter search and its generalized form called path relinking are evolutionary methods that have recently been shown to yield promising outcomes for solving combinatorial and nonlinear optimization problems. Based on formulations originally proposed in the 1960s for combining decision rules and problem constraints, these methods use strategies for combining solution vectors that have proved effective for scheduling, routing, financial product design, neural network training, optimizing simulation and a variety of other problem areas. These approaches can be implemented in multiple ways, and offer numerous alternatives for exploiting their basic ideas. We identify a template for scatter search and path relinking methods that provides a convenient and "user friendly" basis for their implementation. The overall design can be summarized by a small number of key steps, leading to versions of scatter search and path relinking that are fully specified upon providing a handful of subroutines. Illustrative forms of these subroutines are described that make it possible to create methods for a wide range of optimization problems.

Table of Contents

1 Introduction

2 Foundations Of Scatter Search And Path Relinking

2.1 Scatter Search

2.2 Path Relinking

3 Outline Of The Scatter Search/Path Relinking Template

4 Diversification Generator

4.1 Diversification Generators for Zero-One Vectors

4.2 A Sequential Diversification Generator

4.3 Diversification Generator for Permutation Problems

4.4 Additional Role for the Diversification Generator

5 Maintaining And Updating The Reference Set

5.1 Notation and Initialization

6 Choosing Subsets Of The Reference Solutions

6.1 Generating the Subsets of Reference Solutions

6.2 Methods for a Dynamic RefSet

6.3 Arrays for the Subset Generation Method

6.4 Subset Generation Method

7 Improvement Method

7.1 Avoiding Duplications

7.2 Move Descriptions

7.3 Definitions of 1-moves and the Composition of M

7.4 Advanced Improvement Alternatives

8 Conclusions

APPENDIX 1 Construction-by-Objective: Mixed Integer and Nonlinear Optimization

APPENDIX 2: Checking for Duplicate Solutions

1 Introduction

Scatter search and path relinking have recently been investigated in a number of studies, disclosing the promise of these methods for solving difficult problems in discrete and nonlinear optimization. Recent applications of these methods (and of selected component strategies within these methods) include:¹

- Vehicle Routing – Rochat and Taillard (1995); Taillard (1996)
- Quadratic Assignment – Cung *et al.* (1996)
- Financial Product Design – Consiglio and Zenios (1996)
- Neural Network Training – Kelly, Rangaswamy and Xu (1996)
- Job Shop Scheduling – Yamada and Nakano (1996)
- Flow Shop Scheduling – Yamada and Reeves (1997)
- Graph Drawing – Laguna and Marti (1997)
- Linear Ordering – Laguna, Marti and Campos (1997)
- Unconstrained Continuous Optimization – Fleurent *et al.* (1996)
- Bit Representation – Rana and Whitley (1997)
- Optimizing Simulation – Glover, Kelly and Laguna (1996)
- Complex System Optimization – Laguna (1997)

We propose a template for generating a broad class of scatter search and path relinking methods, with the goal of creating versions of these approaches that are convenient to implement. Our design is straightforward, and can be readily adapted to optimization problems of diverse structures. We offer specific comments relating to multidimensional knapsack problems, graph partitioning problems, linear and nonlinear zero-one problems, mixed integer programming problems and permutation problems.

From the standpoint of classification, scatter search and path relinking may be viewed as evolutionary algorithms that construct solutions by combining others, and derive their foundations from strategies originally proposed for combining decision rules and constraints. The goal of these procedures is to enable a solution procedure based on the combined elements to yield better solutions than one based only on the original elements.

Historically, the antecedent strategies for combining decision rules were introduced in the context of scheduling methods, to obtain improved local decision rules for job shop scheduling problems. New rules were generated by creating numerically weighted combinations of existing rules, suitably restructured so that their evaluations embodied a common metric. The approach was motivated by the supposition that information about the relative desirability of alternative choices is captured in different forms by different rules, and that this information can be exploited more

¹ The References Section contains website listings where abstracts and/or copies can be obtained for a number of the references cited in this paper.

effectively when integrated by means of a *combination mechanism* than when treated by the standard strategy of selecting different rules one at a time, in isolation from each other. The decision rules created from such combination strategies produced better empirical outcomes than standard applications of local decision rules, and also proved superior to a “probabilistic learning approach” that selected different rules probabilistically at different junctures, but without the integration effect provided by generating combined rules (Crowston, et al., 1963; Fisher and Thompson, 1963).

The associated procedures for combining constraints likewise employed a mechanism of generating weighted combinations, in this case applied in the setting of integer and nonlinear programming, by introducing nonnegative weights to create new constraint inequalities, called *surrogate constraints* (Glover, 1965, 1968). The approach isolated subsets of constraints that were gauged to be most critical, relative to trial solutions based on the surrogate constraints, and produced new weights that reflected the degree to which the component constraints were satisfied or violated.

A principal function of surrogate constraints, in common with the approaches for combining decision rules, was to provide ways to evaluate choices that could be used to generate and modify trial solutions. From this foundation, a variety of heuristic processes evolved that made use of surrogate constraints and their evaluations. Accordingly, these processes led to the complementary strategy of combining solutions, as a *primal* counterpart to the *dual* strategy of combining constraints², which became manifest in scatter search and its path relinking generalization.

2 Foundations Of Scatter Search And Path Relinking

2.1 Scatter Search

The scatter search process, building on the principles that underlie the surrogate constraint design, is organized to (1) capture information not contained separately in the original vectors, (2) take advantage of auxiliary heuristic solution methods to evaluate the combinations produced and to generate new vectors.

The original form of scatter search (Glover, 1977) may be sketched as follows.

Scatter Search Procedure

1. Generate a starting set of solution vectors by heuristic processes designed for the problem considered, and designate a subset of the best vectors to be *reference solutions*. (Subsequent iterations of this step, transferring from Step 3 below, incorporate advanced starting solutions and best solutions from previous history as candidates for the reference solutions.)

² Surrogate constraint methods give rise to a mathematical duality theory associated with their role as relaxation methods for optimization (e.g., see Greenberg and Pierskalla 1970, 1973; Glover, 1975; Karwan and Rardin, 1976, 1979; Freville and Plateau, 1986, 1993).

2. Create new points consisting of linear combinations of subsets of the current reference solutions. The linear combinations are:
 - (a) chosen to produce points both inside and outside the convex regions spanned by the reference solutions.
 - (b) modified by generalized rounding processes to yield integer values for integer-constrained vector components.
3. Extract a collection of the best solutions generated in Step 2 to be used as starting points for a new application of the heuristic processes of Step 1. Repeat these steps until reaching a specified iteration limit.

Three particular features of scatter search deserve mention. First, the linear combinations are structured according to the goal of generating weighted centers of selected subregions, allowing for nonconvex combinations that project these centers into regions external to the original reference solutions. The dispersion pattern created by such centers and their external projections is particularly useful for mixed integer optimization. (See Appendix 1 for specific procedures in this context.) Second, the strategies for selecting particular subsets of solutions to combine in Step 2 are designed to make use of clustering, which allows different types of strategic variation by generating new solutions “within clusters” and “across clusters”. Third, the method is organized to use supporting heuristics that are able to start from infeasible solutions, and hence which remove the restriction that solutions selected as starting points for re-applying the heuristic processes must be feasible.³

In sum, scatter search is founded on the following premises.

- (P1) Useful information about the form (or location) of optimal solutions is typically contained in a suitably diverse collection of elite solutions.
- (P2) When solutions are combined as a strategy for exploiting such information, it is important to provide for combinations that can extrapolate beyond the regions spanned by the solutions considered, and further to incorporate heuristic processes to map combined solutions into new points. (This serves to provide both diversity and quality.)
- (P3) Taking account of multiple solutions simultaneously, as a foundation for creating combinations, enhances the opportunity to exploit information contained in the union of elite solutions.

The fact that the heuristic processes of scatter search are not restricted to a single uniform design, but represent a varied collection of procedures, affords additional strategic possibilities.

³ An incidental feature – that has more than incidental implications – is the incorporation of general (mixed integer) solution vectors, in contrast to a reliance on binary representations. By contrast, the focus on binary representations that held sway in genetic algorithm proposals until the mid to late 1980s can be shown to create “information gaps” for combining solutions (Glover, 1994a). The problems of *distortion* in binary-based GAs are therefore not surprising.

2.2 Path Relinking

From a spatial orientation, the process of generating linear combinations of a set of reference solutions may be characterized as generating paths between and beyond these solutions (where solutions on such paths also serve as sources for generating additional paths). This leads to a broader conception of the meaning of creating *combinations* of solutions. By natural extension, such combinations may be conceived to arise by generating paths between and beyond selected solutions in neighborhood space, rather than in Euclidean space (Glover 1989, 1994b).

This conception is reinforced by the fact that a path between solutions in a neighborhood space will generally yield new solutions that share a significant subset of attributes contained in the parent solutions, in varying "mixes" according to the path selected and the location on the path that determines the solution currently considered. The character of such paths is easily specified by reference to solution attributes that are added, dropped or otherwise modified by the moves executed in neighborhood space. Examples of such attributes include edges and nodes of a graph, sequence positions in a schedule, vectors contained in extreme point basic solutions, and values of variables and functions of variables. To generate the desired paths, it is only necessary to select moves that perform the following role: upon starting from an *initiating solution*, the moves must progressively introduce attributes contributed by a *guiding solution* (or reduce the distance between attributes of the initiating and guiding solutions). The process invites variation by interchanging the roles of the initiating and guiding solutions, and also by inducing each to move simultaneously toward the other as a way of generating combinations.⁴

Such an incorporation of attributes from elite parents in partially or fully constructed solutions was foreshadowed by another aspect of scatter search, embodied in an accompanying proposal to assign preferred values to subsets of *strongly determined* and *consistent* variables. The theme is to isolate assignments that frequently (and influentially) occur in high quality solutions, and then to introduce compatible subsets of these assignments into other solutions that are generated or amended by heuristic procedures. (Such a process implicitly relies on a simple form of frequency based memory to identify and exploit variables that qualify as consistent, and thereby provides a bridge to associated tabu search ideas discussed in later sections.)

Multiparent path generation possibilities emerge in path relinking by considering the combined attributes provided by a set of guiding solutions, where these attributes are weighted to determine which moves are given higher priority. The generation of such paths in neighborhood space characteristically "relinks" previous points in ways not achieved in the previous search history, hence giving the approach its name.

⁴ Variants of path relinking that use constructive and destructive neighborhoods, called *vocabulary building* approaches, produce strategic combinations of partial solutions (or "solution fragments") as well as of complete solutions. The organization of vocabulary building permits the goal for combining the solution components to be expressed as an optimization model in a number of contexts, with the added advantage of allowing exact methods to be used to generate the moves (see, e.g., Glover and Laguna, 1997).

Neighborhoods for these processes may differ from those used in other phases of search. For example, they may be chosen to *tunnel through* infeasible regions that may be avoided by other neighborhoods. Such possibilities arise because feasible guiding points can be coordinated to assure that the process will re-enter the feasible region, with out danger of becoming "lost." The ability of neighborhood structures to capture contextual features additionally provides a foundation for incorporating domain-specific knowledge about different classes of problems, thus enabling path relinking to exploit such knowledge directly.⁵

3 Outline Of The Scatter Search/Path Relinking Template

The goal of this paper is to provide a template for implementing both scatter search and path relinking that takes advantage of their common characteristics and principles. Such a template is based upon specific subroutines of the following types:

- (1) A Diversification Generator: to generate a collection of diverse trial solutions, using an arbitrary trial solution (or seed solution) as an input.
- (2) An Improvement Method: to transform a trial solution into one or more enhanced trial solutions. (Neither the input nor output solutions are required to be feasible, though the output solutions will more usually be expected to be so. If no improvement of the input trial solution results, the "enhanced" solution is considered to be the same as the input solution.)
- (3) A Reference Set Update Method: to build and maintain a Reference Set consisting of the b best solutions found (where the value of b is typically small, e.g., between 20 and 40), organized to provide efficient accessing by other parts of the method.
- (4) A Subset Generation Method: to operate on the Reference Set, to produce a subset of its solutions as a basis for creating combined solutions.
- (5) A Solution Combination Method: to transform a given subset of solutions produced by the Subset Generation Method into one or more combined solution vectors.

We provide illustrative designs for each of these subroutines except the last, which will typically vary according to the context. The processes for combining solutions in scatter search and path relinking, as embodied in (5), have been discussed at some length in the references previously cited, and especially in Glover and Laguna (1997). Consequently, we focus instead on other features, which deserve greater attention than they have been accorded in the past. Our purpose in particular is to identify forms of

⁵ This may be contrasted with the GA crossover concept, which lacks a unifying design to handle applications that are not well served by the original proposals for exchanging components of binary vectors. The crossover model, which has significantly influenced the popular view of the meaning of combining vectors, compels one of two outcomes: (a) domain-specific knowledge must be disregarded (a property of GAs once regarded to be a virtue); (b) amended versions of crossover must repeatedly be devised, without a guiding principle for structuring the effort to accommodate new problems. The second outcome has led to frequent reliance on ad hoc constructions, as noted by Reeves (1997) and Muhlenbein (1997).

the subroutines (1) to (4), preceding, that can be usefully adapted to a variety of settings, and that can be integrated to produce an effective overall method.

We specify the general template in outline form as follows. This template reflects the type of design often used in scatter search and path relinking, but is not intended to represent the only way to organize and implement such methods. (E.g., see Glover (1994a, 1995), for other alternatives.)

SS/PR Template

Initial Phase

1. (*Seed Solution Step.*) Create one or more seed solutions, which are arbitrary trial solutions used to initiate the remainder of the method.
2. (*Diversification Generator.*) Use the Diversification Generator to generate diverse trial solutions from the seed solution(s).
3. (*Improvement and Reference Set Update Methods.*) For each trial solution produced in Step 2, use the Improvement Method to create one or more enhanced trial solutions. During successive applications of this step, maintain and update a Reference Set consisting of the b best solutions found.
4. (*Repeat.*) Execute Steps 2 and 3 until producing some designated total number of enhanced trial solutions as a source of candidates for the Reference Set.

Scatter Search/Path Relinking Phase

5. (*Subset Generation Method.*) Generate subsets of the Reference Set as a basis for creating combined solutions.
6. (*Solution Combination Method.*) For each subset X produced in Step 5, use the Solution Combination Method to produce a set $C(X)$ that consists of one or more combined solutions. Treat each member of $C(X)$ as a trial solution for the following step.
7. (*Improvement and Reference Set Update Methods.*) For each trial solution produced in Step 6, use the Improvement Method to create one or more enhanced trial solutions, while continuing to maintain and update the Reference Set.
8. (*Repeat.*) Execute Steps 5-7 in repeated sequence, until reaching a specified cutoff limit on the total number of iterations.

Table 1, following, summarizes the relationships between sources of input and output solutions in the preceding template.

<u>Source of Input Solutions</u>	<u>Source of Output Solutions</u>
Arbitrary seed solutions	Diversification Generator
Diversification Generator	Improvement Method
Improvement Method	Reference Set Update Method
Reference Set Update Method	Subset Generation Method
Subset Generation Method	Solution Combination Method
Solution Combination Method	Improvement Method

Table 1: Input/Output Links

Scatter search and path relinking are often implemented in connection with tabu search (TS), and their underlying ideas share a significant intersection with the TS perspective. A principal element of this perspective is its emphasis on establishing a strategic interplay between intensification and diversification. In the original scatter search design (Glover, 1977), which carries over to the present template, intensification is achieved by:

- (a) the repeated use of the Improvement Method as a basis for refining the solutions created (from combinations of others);
- (b) maintaining the Reference Set to consist of the highest quality solutions found; and
- (c) choosing subsets of the Reference Set and uniting their members by strategies that reinforce the goal of generating good solutions (as opposed to relying on mating and “crossover” schemes that are heavily based on randomization).

In the mid to late 1980s, a number of the elements proposed earlier in scatter search began to be introduced in hybrid variants of GA procedures. Consequently, some of the current descendants of these hybrid approaches appear to have a structure similar to the outline of the SS/PR Template. Nevertheless, significant differences remain, due to perspectives underlying scatter search and path relinking that have not become incorporated into the GA hybrids. These are particularly reflected in the concepts underlying intensification and diversification, which will be elaborated in subsequent discussions.

The remaining sections are devoted to providing illustrative forms of the subroutines that support the foregoing template. We first focus on the diversification generator, followed by the method for updating the reference set and then the method for choosing subsets of the reference solutions. Finally we examine the issue of specifying an improvement method, and identify an approach that likewise embodies ideas that have not been adequately considered in the literature.

4 Diversification Generator

We indicate two simple types of diversification generators, one for problems that can be formulated in a natural manner as optimizing a function of zero-one variables, and the other for problems that can more appropriately be formulated as optimizing a permutation of elements. In each of these instances we disregard the possible existence of complicating constraints, in order to provide a simple representation of the basic ideas. However, these methods can also be used in the presence of such constraints by using the solutions generated as targets for creating solutions that satisfy the additional requirements of feasibility. This can be done by applying neighborhood procedures (including those that use constructive or destructive neighborhoods) to insure the preservation or attainment of feasibility, while utilizing evaluations that give preference to moves which approach the targets. In addition, Appendix 1 shows how zero-one solution generators can be embedded in a method to create diversified collections of feasible points for mixed integer programming and nonlinear optimization problems.

These approaches embody the tabu search precept that diversification is not the same as randomization. In this respect, they differ from the randomized approaches for creating variation that are typically proposed in other types of evolutionary approaches. The goal of diversification is to produce solutions that differ from each other in significant ways, and that yield productive (or “interesting”) alternatives in the context of the problem considered. By contrast, the goal of randomization is to produce solutions that may differ from each other in any way (or to any degree) at all, as long as the differences are entirely “unsystematic”. From the tabu search viewpoint, a reliance on variation that is strategically generated can offer advantages over a reliance on variation that is distinguished only by its unpredictability.

4.1 Diversification Generators for Zero-One Vectors

We let x denote an n -vector each of whose components x_j receives the value 0 or 1. The first type of diversification generator we consider takes such a vector x as its seed solution, and generates a collection of solutions associated with an integer $h = 1, 2, \dots, h^*$, where $h^* \leq n - 1$. (Recommended is $h^* \leq n/5$.)

We generate two types of solutions, x' and x'' , for each value of h , by the following rule:

Type 1 Solution: Let the first component x'_1 of x' be $1 - x_1$, and let $x'_{1+kh} = 1 - x_{1+kh}$ for $k = 1, 2, 3, \dots, k^*$, where k^* is the largest integer satisfying $k^* \leq n/h$. Remaining components of x' equal 0.

To illustrate for $x = (0,0,\dots,0)$: The values $h = 1, 2$ and 3 respectively yield $x' = (1,1,\dots,1)$, $x' = (1,0,1,0,1 \dots)$ and $x' = (1,0,0,1,0,0,1,0,0,1,\dots)$. This progression

suggests the reason for preferring $h^* \leq n/5$. As h becomes larger, the solutions x' for two adjacent values of h differ from each other proportionately less than when h is smaller. An option to exploit this is to allow h to increase by an increasing increment for larger values of h .

Type 2 Solution: Let x'' be the complement of x' .

Again to illustrate for $x = (0,0,...,0)$: the values $h = 1, 2$ and 3 respectively yield $x' = (0,0,...,0)$, $x'' = (0,1,0,1,...)$ and $x' = (0,1,1,0,1,1,0,...)$. Since x'' duplicates x for $h = 1$, the value $h = 1$ can be skipped when generating x'' .

We extend the preceding design to generate additional solutions as follows. For values of $h \geq 3$ the solution vector is shifted so that the index 1 is instead represented as a variable index q , which can take the values $1, 2, 3, ..., h$. Continuing the illustration for $x = (0,0,...,0)$, suppose $h = 3$. Then, in addition to $x' = (1,0,0,1,0,0,1,...)$, the method also generates the solutions given by $x' = (0,1,0,0,1,0,0,1,...)$ and $x' = (0,0,1,0,0,1,0,0,1,...)$, as q takes the values 2 and 3 .

The following pseudo-code indicates how the resulting diversification generator can be structured, where the parameter MaxSolutions indicates the maximum number of solutions desired to be generated. Comments within the code appear in *italics*, enclosed within parentheses.

First Diversification Generator for Zero-One Solutions.

NumSolutions = 0

For $h = 1$ to h^*

Let $q^* = 1$ if $h < 3$, and otherwise let $q^* = h$

(q^* denotes the value such that q will range from 1 to q^* . We set $q^* = 1$ instead of $q^* = h$ for $h < 3$ because otherwise the solutions produced for the special case of $h < 3$ will duplicate other solutions or their complements.)

For $q = 1$ to q^*

let $k^* = (n-q)/h$ <rounded down>

For $k = 1$ to k^*

$$x'_{q+kh} = 1 - x_{q+kh}$$

End k

If $h > 1$, generate x'' as the complement of x'

(x' and x'' are the current output solutions.)

NumSolutions = NumSolutions + 2 (or + 1 if $h = 1$)

If NumSolutions $\geq$ MaxSolutions, then stop generating solutions.

End q

End h

The number of solutions x' and x'' produced by the preceding generator is approximately $q^*(q^*+1)$. Thus if $n = 50$ and $h^* = n/5 = 10$, the method will generate

about 110 different output solutions, while if $n = 100$ and $h^* = n/5 = 20$, the method will generate about 420 different output solutions.

Since the number of output solutions grows fairly rapidly as n increases, this number can be limited, while creating a relatively diverse subset of solutions, by allowing q to skip over various values between 1 and q^* . The greater the number of values skipped, the less “similar” the successive solutions (for a given h) will be. Also, as previously noted, h itself can be incremented by a value that differs from 1.

For added variation:

If further variety is sought, the preceding approach can be augmented as follows. Let $h = 3, 4, \dots, h^*$, for $h \leq n - 2$ (preferably $h^* \leq n/3$). Then for each value of h , generate the following solutions.

Type 1A Solution: Let $x'_1 = 1 - x_1$ and $x'_2 = 1 - x_2$. Thereafter, let $x'_{1+kh} = 1 - x_{1+kh}$ and let $x'_{2+kh} = 1 - x_{2+kh}$, for $k = 1, 2, \dots, k^*$, where k^* is the largest integer such that $2 + kp \leq n$. All other components of x' are the same as in x .

Type 2A Solution: Create x'' as the complement of x' , as before.

Related variants are evident. The index 1 can also be shifted (using a parameter q) in a manner similar to that indicated for solutions of type 1 and 2.

4.2 A Sequential Diversification Generator

The concept of diversification invites a distinction between solutions that differ from a given solution (e.g., a seed solution) and those that differ from each other.⁶ Our preceding comments refer chiefly to the second type of diversification, by their concern with creating a collection of solutions whose members exhibit certain contrasting features.

Diversification of the first type can be emphasized in the foregoing design by restricting attention to the complemented solutions denoted by x'' when h becomes larger than 2. In general, diversification of the second type is supported by complementing larger numbers of variables in the seed solution. We stress that this type of diversification by itself is incomplete, and the relevance of diversification of the first type is important to heed in many situations.

Approaches that combine characteristics of both types of diversification can be founded on the ideas of *sequential diversification* (Glover and Laguna, 1997). A “diverse sequence” can be composed by a rule that selects each successive vector to

⁶ These distinctions have often been overlooked by the genetic algorithm community, in spite of the emergence of GA hybrids that take advantage of tabu search to obtain improved outcomes. The interplay between intensification and diversification is sometimes further obscured by confusing it with the control theory notion of “exploitation versus exploration,” which GAs have traditionally adopted. The nature and consequences of these differing ideas are discussed, for example, in Glover and Laguna (1997).

maximize the minimum distance from all vectors previously considered, according to a chosen distance metric. (There can sometimes be many vectors that qualify to be chosen as the next member of a partially constructed sequence by such a criterion. Ties can be broken by a perturbed distance function where distances to vectors that appear later in the partial sequence are considered smaller than distances to vectors that appear earlier in the sequence.) Sequential diversification makes it possible to generate solutions that differ from a *set* of solutions (such as a set of solutions examined during a search process), rather than those that differ from a single solution, by considering the members of the set to be first members of the sequence.

A sequential diversification generator for 0-1 vectors that follows the prescription to maximize the minimum distance from preceding vectors is embodied in the following procedure. We say that a solution y complements x over an index set J if $y_j = 1 - x_j$ for $j \in J$ and $y_j = x_j$ for $j \notin J$.

Sequential (Max/Min) Diversification Generator

1. Designate the seed solution x and its complement to be the first two solutions generated.
2. Partition the index set $N = \{1, \dots, n\}$ for x into two sets N' and N'' that, as nearly as possible, contain equal numbers of indexes. Create the two solutions x' and x'' so that x' complements x over N' and x'' complements x over N'' .
3. Define each subset of N that is created by the most recent partition of N to be a *key subset*. If no key subset of N contains more than 1 element, stop. Otherwise, partition each key subset S of N into two sets S' and S'' that contain, as nearly as possible, equal numbers of elements. Balance the overall collection of partitions so that the number with $|S'| > |S''|$ also equals the number with $|S'| < |S''|$ as nearly as possible. (This includes the case where S may contain only 1 element, and hence where S' or S'' is empty.)
4. Let N' be the union of all subsets S' and let N'' be the union of all subsets S'' . Create the complementary solutions x' and x'' relative to N' and N'' as in Step 2, and then return to Step 3.

The foregoing process generates approximately $2(1 + \log n)$ solutions. If n is a power of 2, every solution produced maximizes the minimum Hamming distance from all previous solutions generated. (This maxmin distance, measured as the number of elements by which the solutions differ, is $n/2$ for every iteration after the first two solutions are generated in Step 1. Such a maxmin value is also approximately achieved when n is not a power of 2.)

Useful variations result by partitioning N in different ways, but different partitions do not always produce different results. For example, if the initial rule is to partition each set so that the first member of the partition only contains indexes that are smaller than the minimum index contained in the second, then a corresponding rule that first

reindexes the last fourth of the elements of N to become instead the second fourth of these elements will produce exactly the same set of solutions as the initial rule.

The foregoing 0-1 Diversification Generator and the one first described can additionally be applied to generate solutions that differ from a set of solutions by using a TS frequency-based approach. An example occurs by weighting the vectors of the set according to their importance, and creating a combined vector of weighted frequencies by summing the weighted vectors. The resulting vector can then be transformed into a seed solution for the Diversification Generator by mapping each of its components into 0 or 1, according to whether the value of the component lies below or above a chosen threshold.

The next section offers another type of organization to assure that new solutions differ from the seed solution as well as from each other.

4.3 Diversification Generator for Permutation Problems

Although permutation problems can be formulated as 0-1 problems, they constitute a special class that preferably should be treated somewhat differently. Assume that a given trial permutation P used as a seed is represented by indexing its elements so they appear in consecutive order, to yield $P = (1, 2, \dots, n)$. Define the subsequence $P(h:s)$, where s is a positive integer between 1 and h , to be given by $P(h:s) = (s, s+h, s+2h, \dots, s+rh)$, where r is the largest nonnegative integer such that $s+rh \leq n$. Then define the permutation $P(h)$, for $h \leq n$, to be $P(h) = (P(h:h), P(h:h-1), \dots, P(h:1))$.

Illustration:

Suppose P is given by

$$P = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18)$$

If we choose $h = 5$, then $P(5:5) = (5, 10, 15)$, $P(5:4) = (4, 9, 14)$, $P(5:3) = (3, 8, 13, 18)$, $P(5:2) = (2, 7, 12, 17)$, $P(5:1) = (1, 6, 11, 16)$, to give:

$$P(5) = (5, 10, 15, 4, 9, 14, 3, 8, 13, 18, 2, 7, 12, 17, 1, 6, 11, 16)$$

Similarly, if we choose $h = 4$ then $P(4:4) = (4, 8, 12, 16)$, $P(4:3) = (3, 7, 11, 15)$, $P(4:2) = (2, 6, 10, 14, 18)$, $P(4:1) = (1, 5, 9, 13, 17)$ to give:

$$P(4) = (4, 8, 12, 16, 3, 7, 11, 15, 2, 6, 10, 14, 18, 1, 5, 9, 13, 17)$$

In this illustration we have allowed h to take the two values closest to the square root of n . These values are interesting based on the fact that, when h equals the square root of n , the minimum relative separation of each element from each other element in the new permutation is maximum, compared to the relative separation of exactly 1 in the permutation P . In addition, other useful types of separation result, and become more pronounced for larger values of n .

In general, for the goal of generating a diverse set of permutations, preferable values for h range from 1 to $n/2$. We also generate the reverse of the preceding permutations, denoted by $P^*(h)$, which we consider to be more interesting than $P(h)$. The preference of $P^*(h)$ to $P(h)$ is greater for smaller values of h . For example, when $h = 1$, $P(h) = P$ and $P^*(h)$ is the reverse of P . (Also, $P(n) = P^*(1)$.) In sum, we

propose a Diversification Generator for permutation problems to be one that generates a subset of the collection $P(h)$ and $P^*(h)$, for $h = 1$ to $n/2$ (excluding $P(1) = P$).

4.4 Additional Role for the Diversification Generator

A diversification generator that takes one of the forms indicated above can be used to modify the balance between intensification and diversification in the SS/PR Template. In particular, a stronger diversification emphasis can be effected by means of optional additions to Steps 6 and 7, which we denote by 6A and 7A, as follows.

6A. (*Pre-Improvement Diversification.*) Use selected trial solutions produced in Step 6 as seeds to produce additional trial solutions by the Diversification Generator.

7A. (*Post-Improvement Diversification.*) Similarly, use selected trial solutions produced in Step 7 to produce additional trial solutions by the Diversification Generator, where each resulting trial solution in this case is immediately subjected again to Step 7 to produce an enhanced solution.

The provision to apply Step 7 again in the Post-Improvement Diversification is not required in the Pre-Improvement Diversification because Step 6 (and hence 6A) is directly followed by Step 7 in the SS/PR Template. Typically the optional steps 6A and 7A will be used to generate a single additional trial solution for each trial solution selected to be a seed solution. The Diversification Generator of the next section can easily be restricted to generate such a solution that is “far from” the seed solution by reference to the observations previously indicated. The potential value of these optional steps for a given application can of course be conveniently determined by implementing a version of the SS/PR Template that excludes them, and then comparing the outcomes with those produced by incorporating Step 6A and/or Step 7A.

5 Maintaining And Updating The Reference Set

The Reference Set Update method accompanies each application of the Improvement Method, and we examine this updating function first because of its linking role, which also introduces structures that provide a foundation for the Subset Generation Method. The update operation consists of maintaining a record of the b best solutions found, where value of b is treated as a chosen constant, but may readily be allowed to vary. The underlying issues are conceptually straightforward, but their ramifications within the SS/PR Template are sufficiently extensive to motivate a concrete design for such a procedure.⁷

Hereafter, we will refer to the Reference Set by the abbreviation “RefSet”. We provide a pseudo-code description of the procedure for maintaining RefSet, which we

⁷ This section and the next are devoted explicitly to concerns of computer implementation. We include details that serve primarily for convenience as well as those that serve a more substantial function.

call the *RefSet Update Routine*, that is organized to handle vectors of 0-1 variables. Related forms to handle permutation vectors follow the same general format.

5.1 Notation and Initialization

Let $bNow$ denote the current number of solutions in the Reference Set. $bNow$ begins at 0 in an Initialization Step, and is increased each time a new solution is added to the Reference Set, until reaching the value $bMax$ (as where $bMax$ may take a value from 10 to 40, for example).

At each step, RefSet stores the best solutions in an array $x[i]$, $i = 1$ to $bNow$. The solution vector $x[i]$ may be viewed as the “ i th row” of the array. An associated location array $loc(i)$, $i = 1$ to $bNow$, indicates the ranking of the solutions; that is, $x[loc(1)]$ (the x vector stored in location “ $loc(1)$ ”) is the best solution, $x[loc(2)]$ is the next best solution, and so forth.

A solution $x = x'$ is not permitted to be recorded if it duplicates another already in RefSet.⁸ We speed the check to see if one solution duplicates another by keeping some simple auxiliary information about each solution. Specifically, in addition to keeping a value $E(loc(i))$ which identifies the evaluation (such as an objective function value) for $x[loc(i)]$, we may also keep a hash function value for this solution which we denote by $Hash(loc(i))$. Such a value can be determined by a single pass of the components of $x[loc(i)]$.

To test for adding x' to RefSet, let $E0$ and $Hash0$ be defined for x' in the same way the corresponding quantities $E(loc(i))$ and $Hash(loc(i))$ are defined for $x[loc(i)]$. In addition to storing x' properly, if it qualifies to be added to RefSet, the RefSet Update Subroutine will keep track of a value $NewRank$, which will be the rank of x' (as 1st best, 2nd best, etc.) if it is added to RefSet, and otherwise receives the value 0. (Thus, checking whether $NewRank$ is 0 or positive upon completion of the subroutine will automatically disclose whether x' succeeded or failed to qualify for inclusion.)

Finally, we let $RefSetCall$ count the number of times the RefSet Update subroutine is called (which thus tells how many solutions are generated and examined as potential additions to RefSet), and let $RefSetAdd$ count the number of times a solution is actually added. These values can be useful in the situation where it is desired to control the number of steps of executing the overall algorithm by limiting the maximum value of $RefSetCall$ or $RefSetAdd$, for example. The values may appropriately be segregated and re-initialized to give separate statistics for the Initial Phase and for the Scatter Search/Path Relinking Phase of the SS/PR Template. Auxiliary information that may be useful is recorded by three counters, $DupCheck$,

⁸ Unlike the development of scatter search and path relinking, where duplications have no relevance, the notion of avoiding duplicate population members was not embraced in GA methods for many years, and some GA researchers continue instead to argue in favor of duplications (see, e.g., Goldberg, 1989). However, the work of Whitley and Kauth (1988) and Whitley (1989) introduces a “steady state” design that avoids duplications, and the usefulness of this policy for such amended GAs has also been demonstrated by Davis (1991).

FullDupCheck and FullDupFound, which respectively count the number of partial duplication checks, full duplication checks, and the number of occurrences when duplications were found.

Pseudo-Code for the RefSet Update

Initialization Step

```

bNow = 0
RefSetCall = 0
RefSetAdd = 0
DupCheck = 0
FullDupCheck = 0
FullDupFound = 0

```

RefSet Update Subroutine (To Add x' to RefSet if it qualifies):

(This subroutine calls the Add Subroutine, specified below, which carries out the details of adding the solution x' to Refset.)

Begin Subroutine:

```

RefSetCall = RefSetCall + 1
NewRank = 0

```

If bNow = 0 then:

```

    NewRank = 1

```

(this indicates that x' will be recorded as the solution with the first rank, i.e., the best, since no other solution is yet recorded.)

```

    Compute Hash0

```

```

    Call the Add Subroutine

```

```

    End the RefSet Update Subroutine

```

Elseif bNow > 0 then:

(First check x' against the worst of the bNow best solutions, as follows:)

```

    If  $E0 \leq E(\text{loc}(\text{bNow}))$  and if bNow = bMax then:

```

(x' is not better than the worst, and the list is full, so don't add x' to RefSet)

```

        End the RefSet Update Subroutine.

```

```

    Else

```

```

        Compute Hash0

```

```

    Endif

```

```

    If  $E0 > E(\text{loc}(1))$  then:

```

(x' is the new best solution of all, so record the rank of x' as NewRank)

```

        NewRank = 1.

```

```

    Else

```

(go through the solutions $x[\text{loc}(1)]$ to $x[\text{loc}(\text{bNow})]$ in reverse order, to test if x' duplicates a previous solution or is new)

```

    For i = bNow to 1 (decreasing index order)

```

```

        If  $E(\text{loc}(i)) = E0$  then:

```

```

            DupCheck = DupCheck + 1

```

(this counts the duplication checks that are not trivially eliminated by the objective function value)

```

    If Hash(loc(i)) = Hash0 then:
        FullDupCheck = FullDupCheck + 1
        (check  $x'$  against  $x[loc(i)]$ )
        If  $x' = x[loc(i)]$  then:
            FullDupFound = FullDupFound + 1
            End the RefSet Update Subroutine
            ( $x'$  is not added to RefSet)
        Endif
    Endif
Elseif  $E(loc(i)) > E0$  then:
    (by the sequence of the loop, the current  $i$  is the largest index  $i$  that satisfies  $E(loc(i)) > E0$  – i.e.,  $x[loc(i)]$  is the worst solution that is still better than  $x'$ , so  $x'$  will be ranked as the  $(i+1)$ -st best solution)
    NewRank =  $i+1$ 
    Call the Add Subroutine
    End the RefSet Update Subroutine
Endif
End i
(If the method reaches here, it has gone through the loop above without finding a solution better than  $x'$  and without finding a duplicate for  $x'$ . So  $x'$  qualifies as a new best solution, though its evaluation value must implicitly tie for best.)
NewRank = 1
Endif
Call the Add Subroutine
Endif
End RefSet Update Subroutine

```

Now we indicate the subroutine called by the RefSet Update Subroutine. In addition to the arrays already noted, we include a LastTime(loc0) array (updated at the end of the following Add Subroutine) where loc0 ranges over the locations loc(i), $i = 1$ to bMax. The array LastTime(loc0), which records the last (“most recent”) time that the solution stored in location loc0 changes its identity, is important for linking with the routines of Section 4, which are designed to assure no duplicate subsets X of RefSet are ever generated).

Add Subroutine (to add x' to RefSet, given that it qualifies):

Begin Subroutine

```
RefSetAdd = RefSetAdd + 1
```

(*x' will be recorded in the location occupied by the current worst solution. In case $bNow < bMax$, imagine the “worst” to be the empty solution in the location $loc(bNow + 1)$, which will become $loc(bNow)$ after incrementing $bNow$.)*

```
If  $bNow < bMax$  then:
```

```
     $bNow = bNow + 1$ 
```

loc(bNow) = bNow

Endif

(Next, the location pointers, loc(i), must be updated. First save the pointer to the solution that is currently worst, because this is the location where x' will be stored.)

loc0 = loc(bNow)

(Now update the location pointers that change, as a result of making x' the solution that acquires the rank of NewRank. We only need to change pointers from NewRank + 1 to bNow, because loc(NewRank) will be updated for x' , below. To avoid destroying proper values, the change must be made in reverse order.)

If NewRank < bNow then:

For i = bNow to NewRank + 1 (in decreasing index order)

loc(i) = loc(i-1)

End i

Endif

x[loc0] = x'

(thus x' is stored in the current location loc0)

loc(NewRank) = loc0

(x' will now be accessed as the solution whose rank is NewRank, via the loc array)

Hash(loc0) = Hash0.

(Finally, record the "time", given by NowTime, when the solution in location loc0 last changed its identity. NowTime is updated elsewhere, as shown later.)

LastChange(loc0) = NowTime

End of Add Subroutine

A variation on ideas embodied in the preceding routines can be used to provide a method that checks for and eliminates duplications among solutions that are passed to the Improvement Method in Steps 2 and 7 of the SS/PR Template. By the philosophy of the approach developed here, such a weeding out of duplicates in the part of the overall approach can also be particularly useful, and a pseudo-code for the corresponding method is provided in Appendix 2.⁹

6 Choosing Subsets Of The Reference Solutions

We now introduce a special approach for creating different subsets X of RefSet, as a basis for implementing Step 5 of the SS/PR Template. It is important to note the

⁹ A simple refinement can also be introduced in the current routine by identifying the first and last indexes of nonzero components of x' during the pass of the components of x' that identifies Hash0. In case the hash value check does not disclose that x' differs from x[loc(i)], the correspondence of these first and last indexes with those previously saved for x[loc(i)] can also be checked. This allows a full check of x' to x[loc(i)] to be restricted to components within the range of these indexes.

SS/PR Template prescribes that the set $C(X)$ of combined solutions (i.e., the set of all combined solutions that we intend to generate) is produced in its entirety at the point where X is created. Therefore, once a given subset X is created, there is no merit in creating it again. This creates a situation that differs noticeably from those considered in the context of genetic algorithms.

In some scatter search proposals, for example, the set $C(X)$ associated with X consists of a single (weighted) center of gravity of the elements of X . Once the center of gravity is created from X , it is preferable to avoid recreating the same subset X in the future. Other proposals similarly specify a particular set of combined points to be created from a given subset X . These points may be variable in number, as from a deterministic algorithm applied to X that terminates when the quality of points generated falls below a threshold. However, the total number of such points that are retained after an initial screening is usually small. Some path relinking proposals have a corresponding character (see, e.g., Glover, 1994b).

In such situations, we seek a procedure that generates subsets X of RefSet that have useful properties, while avoiding the duplication of subsets previously generated. Our approach for doing this is organized to generate four different collections of subsets of RefSet, which we refer to as SubsetType = 1, 2, 3 and 4. The principles we apply to generate these subsets can be applied to create additional of subsets of a related character. We can also adapt the basic ideas to handle a policy that, by contrast, allows selected subsets of solutions to be generated a second time — as where it may be desired to create more than one “brood” of offspring from a given collection of parents, under conditions where the history of the method suggests that such a collection should be singled out for generating additional offspring.

A central consideration is that RefSet itself will not be static, but will be changing as new solutions are added to replace old ones (when these new solutions qualify to be among the current b best solutions found). The dynamic nature of RefSet requires a method that is more subtle than one that itemizes various subsets of an unchanging RefSet. In addition, we wish to restrict attention to a relatively small number of subsets with useful features, since there are massive numbers of subsets that may be generated in general. The types of subsets we consider are as follows.

- SubsetType = 1: all 2-element subsets.
- SubsetType = 2: 3-element subsets derived from the 2-element subsets by augmenting each 2-element subset to include the best solution not in this subset.
- SubsetType = 3: 4-element subsets derived from the 3-element subsets by augmenting each 3-element subset to include the best solutions not in this subset.
- SubsetType = 4: the subsets consisting of the best i elements, for $i = 5$ to $bNow$.

The total number of subsets that satisfy the preceding stipulations is usually quite manageable. For example, if $bMax = 10$ there are 45 different 2-elements subsets for SubsetType = 1, and the collections for SubsetType = 2 and 3 each contain a bit less

than 45 additional subsets. All together, SubsetType = 1 to 4 would generate approximately 130 distinct subsets. If $bMax = 20$, the total number of different subsets generated is a little less than 600. Depending on the number of solutions contained in $C(X)$, and on the amount of time required to generate a given combined solution and to enhance it by the Improvement Method, the value of $bMax$ can be increased or decreased, or the types of subsets produced can similarly be changed (to produce variants or subsets of the four types generated by the process subsequently indicated).

Since the method will continue to add new solutions to RefSet — until no new solutions can be found that are better than the $bMax$ best — the number of subsets generated will typically be larger than the preceding figures. Consequently, a limit is placed on the number of solutions generated in total, in case the best solutions keep changing. (A limit may also be placed on the number of iterations that elapse after one of the top 2 or 3 solutions has changed.) An appropriate cutoff can be selected by initial testing that gives the cutoff a large value, and by saving statistics to determine what smaller value is sufficient to generate good solutions.

Rationale

The reason for choosing the four indicated types of subsets of RefSet is as follows. First, 2-element subsets are the foundation of the first “provably optimal” procedures for generating constraint vector combinations in the surrogate constraint setting, whose ideas are the precursors of the ideas that became embodied in scatter search (see, e.g., Glover, 1965; Greenberg and Pierskalla, 1970). Also, conspicuously, 2-element combinations have for many years dominated the genetic algorithm literature (in “2-parent” combinations for crossover).

The generation of 2-element subsets also automatically generates (b-2)-element subsets, as complements of the 2-element subsets. We find the collection of (b-2)-element subsets less interesting, in the sense that the relative composition of such subsets tends to be much less varied than that of the 2-element subsets. (An exception occurs where the excluded elements may significantly affect the nature of the set $C(X)$.)

Our extension of the 2-element subsets to 3-element subsets is motivated by the fact that the values 2 and 3 are (proportionally) somewhat different from each other, and hence we anticipate the 3-element subsets will have an influence that likewise is somewhat different than that of the 2-element subsets. However, since the 3-element subsets are much more numerous than the 2-element subsets, we restrict consideration to those that always contains the best current solution in each such subset, which therefore creates approximately the same number of subsets as the 2-element subsets. A variation would be to replace this best solution with one of the top several best solutions, chosen pseudo randomly. Likewise, we extend the 3-element subsets to 4-element subsets for the same reason, and similarly restrict attention to a subcollection of these that always includes the two best solutions in each such subset.

As the subsets become larger, the proportional difference in their successive sizes becomes smaller. The computational effort of handling such subsets also tends to

grow. Hence we have chosen to limit the numbers of elements in these subsets (in this case to 4).

Nevertheless, to obtain a limited sampling of subsets that contain larger numbers of solutions we include the special subsets designated as `SubsetType = 4`, which include the b best solutions as b ranges from 5 to $bNow$. (Recall that $bNow$ increases as each new solution is added, until reaching $bMax$.) Since such subsets increasingly resemble each other for adjacent values of b as b grows, an option is to increment b by some fraction of $bMax$, e.g., $bMax/5$, instead of by 1.

6.1 Generating the Subsets of Reference Solutions

Each of the following algorithms embodies within it a step that consists of generating the set of combined solutions $C(X)$ and executing the Improvement Method (e.g., as proposed in Section 4). Regardless of the form of the Improvement Method used, we will understand that the Reference Set Update method of the previous section is automatically applied with it.

To introduce the general approach to create the four types of subsets, we first briefly sketch a set of four corresponding simple algorithms that could be used in the situation where `RefSet` is entirely static (i.e., where the set of $bMax$ best solutions never changes). These algorithms have the deficiency of potentially generating massive numbers of duplications if applied in the dynamic setting (where they must be re-initiated when `RefSet` becomes modified). However, their simple nature gives a basis for understanding the issues addressed by the more advanced approach.

In the following, when we specify that a particular solution is to become the second solution in X , we understand that the current first solution in X is unchanged, and similarly when we specify that a solution is to become the third (fourth) solution in X , we understand that the previous first and second (and third) solutions are unchanged.

Simple Algorithm for Subset Type 1

```

For i = 1 to bNow - 1
    Let x[loc(i)] be the first solution in X
    For j = i+1 to bNow
        Let x[loc(j)] be the second solution in X
        Create C(X) and execute the Improvement Method
    End j
End i

```

Simple Algorithm for Subset Type 2

```

Let x[loc(1)] be the first solution in X
For i = 2 to bNow - 1
    Let x[loc(i)] be the second solution in X
    For j = i+1 to bNow
        Let x[loc(j)] be the third solution in X
        Create C(X) and execute the Improvement Method
    End j
End i

```

Simple Algorithm For Subset Type 3

```

Let x[loc(1)] and x[loc(2)] be the first two solutions in X
For i = 3 to bNow - 1
    Let x[loc(i)] be the third solution in X
    For j = i+1 to bNow
        Let x[loc(j)] be the fourth solution in X
        Create C(X) and execute the Improvement Method
    End j
End i

```

Simple Algorithm for Subset Type 4

```

For i = 1 to bNow
    Let x[loc(i)] be the ith solution in X
    If i ≥ 5:
        Create C(X) and execute the Improvement Method
End i

```

6.2 Methods for a Dynamic RefSet

We must create somewhat more elaborate processes than the preceding to handle a dynamically changing reference set. We first indicate initializations that will be used to facilitate these processes.

Initialization Step:

```

For SubsetType = 1 to 4
    LastRunTime(SubsetType) = 4
End i
NowTime = 0
StopCondition = 0
SubsetType = 0

```

LastRunTime (SubsetType) is set to 4 above, since the approach cycles through 4 component methods. This initialization is a device to allow the approach to execute

all 4 methods on the first pass. Afterward, the procedure will terminate based on updated values of `LastRunTime` compared to `LastChange`, as subsequently explained.

In the iterative application of the steps of the SS/PR Template, we start from `SubsetType = 0`, as in the final statement of the Initialization Step, and repeatedly increase this index in a circular pattern that returns to `SubsetType = 1` after reaching `SubsetType = 4`. However, the design that follows can be readily adapted to cycle through the subsets in any other order.

6.3 Arrays for the Subset Generation Method

There are two key arrays for the subset generation method, `LastChange(loc(i))` and `LastRunTime(SubsetType)`, that form the basis for avoiding duplications efficiently. We describe the function of these arrays, and the associated components that govern their updates, as follows.

(1) `LastChange(loc(i))` – identifies the last (most recent) time that the solution stored in location `loc(i)` changed its identity (i.e., the last time a new solution was written over the old one in this location). More precisely, `LastChange(loc(i))` is assigned the value `NowTime` when this change occurs. `NowTime` is increased by 1 each time one of the four algorithms prepares to generate the subsets of the type it deals with. As a result, `NowTime` is always 1 more than the value that could have been assigned to `LastChange(loc(i))` on the previous execution of any (other) algorithm. Thus, the condition `LastChange(loc(i)) = NowTime` can only hold if a solution was changed in location `loc(i)` during the execution of the current algorithm for selecting elements to combine.

(2) `LastRunTime(SubsetType)` – identifies the last time (the value of `NowTime` on the last time) that the Algorithm `SubsetType` ($= 1, 2, 3, \text{ or } 4$) was executed. Thus, for $iLoc = loc(i)$, the condition `LastChange(iLoc) < LastRunTime(SubsetType)` means that the last time the solution `x[iLoc]` changed, occurred before the last time the Algorithm `SubsetType` was run. Hence `x[iLoc]` will now be the same as when the Algorithm looked at it previously. On the other hand, if `LastChange(iLoc) ≥ LastRunTime(SubsetType)`, then the solution was changed either by Algorithm `SubsetType` itself, or by another algorithm executed more recently, and so `x[iLoc]` is not the same as when Algorithm `SubsetType` looked at it before.

6.4 Subset Generation Method

A basic part of the Subset Generation Method is the Subset Control Subroutine, which oversees the method and calls other subroutines to execute each Algorithm `SubsetType` (for `SubsetType = 1` to 4). We indicate the form of this subroutine first. (The parameter `StopCondition` that governs the outer loop, which immediately follows, is initialized to 0 in the Initialization Step. When the cumulative number of executions of the Improvement Method, as it is applied within the various Algorithm Subroutines, exceeds a chosen limit, then `StopCondition` is set to 1 and the overall method thereby stops.)

Subset Control Subroutine

While StopCondition = 0 do

 SubsetType = SubsetType + 1

 If SubsetType > 4 then SubsetType = 1

 NowTime = NowTime + 1

 iNew = 0

 iOld = 0

(The next loop isolates all new (changed) solutions by storing their locations in LocNew(i), i = 1 to iNew, and all old (unchanged) solutions by storing their locations in LocOld(i), i = 1 to iOld. If iNew winds up 0, nothing has changed. When all algorithms are performed one after another, as here, regardless of sequence, the condition iNew = 0 means nothing has changed for any of them, and the method can stop.)

 For i = 1 to bNow

 iLoc = loc(i)

 if LastChange(iLoc) ≥ LastRunTime(SubsetType) then

 iNew = iNew + 1

 LocNew(iNew) = iLoc

 else

 iOld = iOld + 1

 LocOld(iOld) = iLoc

 Endif

 End i

 If iNew = 0 then end the Subset Control Subroutine

(iNew = 0 here implies all combinations of the four types of subsets have been examined for their current composition without generating any new solutions, and so the SS/PR Template can terminate as a result of exhaustively considering all relevant subsets of RefSet in its final composition.)

 If SubsetType = 1 Call Algorithm 1 Subroutine

 If SubsetType = 2 Call Algorithm 2 Subroutine

 If SubsetType = 3 Call Algorithm 3 Subroutine

 If SubsetType = 4 Call Algorithm 4 Subroutine

 (if StopCondition > 0 stop)

(Having identified the sets of old and new solutions and generated new combinations from them, we can update LastRunTime(SubsetType) to be the current NowTime value, so that the next time the algorithm is applied, LastRunTime(SubsetType) will in fact be the last (most recent) time the algorithm was run.)

 LastRunTime(SubsetType) = NowTime

End do

End Subset Control Subroutine

Next we identify the Algorithm Subroutines. Each Algorithm Subroutine works on the following principle. A subset X of RefSet can be new if and only if at least one

element of X has not been contained in any previous X for the same SubsetType. We exploit this by sorting the solutions into old and new components, and executing a loop that first generates all combinations of new with new, and then a loop that generates all combinations of new with old. Meanwhile, any solution that is changed on the present application of the algorithm is excluded from being accessed once it has changed, because all subsets that include this solution will be generated on a later pass. To access a solution after it changes its rank, but before the loop is completed, would create duplications (unless the solution changes again), and in any case may generate more solutions than necessary. The method generates the least number of solutions “currently known” to be necessary.

Algorithm 1 Subroutine

Begin Subroutine

(Currently $iNew > 0$. If $iNew > 1$, then we look at all combinations of new with new)

If $iNew > 1$ then

For $i = 1$ to $iNew - 1$

$iLoc = LocNew(i)$

If $LastChange(iLoc) < NowTime$ then

(the solution in $iLoc$ is still unchanged, so we can use it, otherwise the method skips it)

Let $x[iLoc]$ be the first element of X

For $j = i + 1$ to $iNew$

$jLoc = LocNew(j)$

If $LastChange(jLoc) < NowTime$ then

Let $x[jLoc]$ be the second element of X

Create the set $C(X)$ and execute the Improvement Method

(Optional check: if $LastChange(iLoc) = NowTime$ then jump to the end of the "I loop" to pick up the next I , and generate fewer solutions.)

Endif

End j

Endif

End i

Endif

If $jOld > 0$ then

For $i = 1$ to $iNew$

$iLoc = LocNew(i)$

If $LastChange(iLoc) < NowTime$ then

Let $x[iLoc]$ be the first element of X

For $j = 1$ to $jOld$

$jLoc = LocOld(j)$

If $LastChange(jLoc) < NowTime$ then

Let $x[jLoc]$ be the second element of X

Create the set $C(X)$ and execute the Improvement Method

(Optional check: if LastChange(iLoc) = NowTime then jump to the end of the “i loop” to pick up the next i, and generate fewer solutions.)

Endif

End j

Endif

End i

Endif

End Subroutine

Algorithm 2 Subroutine

Begin Subroutine

loc1 = loc(1)

Let x[loc1] be the first element of X

If LastChange(loc1) $\geq$ LastRunTime(SubsetType) then

(The solution in location loc1 is new, since last time.)

For i = 2 to bNow - 1

iLoc = loc(i)

If LastChange(iLoc) < NowTime then

(The solution in iLoc is still unchanged, so we can use it, otherwise the method skips it)

x[iLoc] is the second element of X

For j = i + 1 to bNow

jLoc = loc(j)

If LastChange(jLoc) < NowTime then

x[jLoc] is the third element of X

Create C(X) and execute the Improvement Method

(Optional check: if LastChange(iLoc) = NowTime then jump to the end of the “i loop” to pick up the next i, and generate fewer solutions.)

Endif

End j

Endif

End i

End Algorithm 2 Subroutine (if reach here)

Else

(The solution in location loc1 is not new, since last time.)

(If iNew > 1, then we look at all combinations of new with new.)

If iNew > 1 then

For i = 1 to iNew - 1

iLoc = LocNew(i)

If LastChange(iLoc) < NowTime Then

x[iLoc] is the second element of X

For j = i + 1 to iNew

jLoc = LocNew(j)

If LastChange(jLoc) < NowTime then

```

        x[jLoc] is the third element of X
        Create C(X) and execute the Improvement Method
        (Optional check: if LastChange(iLoc) = NowTime
        then jump to the end of the "i loop" to pick up the next
        i, and generate fewer solutions.)
    Endif
End j
Endif
End i
Endif
If jOld > 1 then
    For i = 1 to iNew
        iLoc = LocNew(i)
        If LastChange(iLoc) < NowTime then
            Let x[iLoc] be the second element of X
            For j = 2 to jOld
                (loc1 is actually also LocOld(1))
                jLoc = LocOld(j)
                If LastChange(jLoc) < NowTime then
                    Let x[jLoc] be the third element of X
                    Create C(X) and execute the Improvement Method
                    (Optional check: if LastChange(iLoc) = NowTime
                    then jump to the end of the "i loop" to pick up the next
                    i, and generate fewer solutions.)
                Endif
            End j
        Endif
    End i
Endif
End Subroutine

```

The optional checks in Algorithms 1 and 2 are based on the fact that the condition $\text{LastChange}(i\text{Loc}) = \text{NowTime}$ implies that $x[i\text{Loc}]$ has been changed by finding a new solution with the Improvement Method. No duplications are created if the algorithm continues its course, using the old version of $x[i\text{Loc}]$. But it is also legitimate to jump to the end of the "i loop", as indicated, to generate fewer solutions. Algorithm 2 can also include a more influential check (in the same locations) which asks if $\text{LastChange}(\text{loc1}) = \text{NowTime}$, and terminates the current execution of the algorithm if so. In this case, a variation on the general organization could allow Algorithm 2 to be re-initiated immediately, since all its subsets will not incorporate a new "best overall" solution. Similar comments apply to introducing optional checks within the Algorithm 3 Subroutine, where $\text{LastChange}(\text{loc2})$ can also be checked. We do not bother to include further mention of such options.

Algorithm 3 Subroutine**Begin Subroutine**

loc1 = loc(1)

loc2 = loc(2)

Let $x[loc1]$ and $x[loc2]$ be the first two elements of X

If $LastChange(loc1) \geq LastRunTime(SubsetType)$ or $LastChange(loc2) \geq$

$LastRunTime(SubsetType)$ then

(The solution in location loc 1 or in loc2 is new, since last time.)

For $i = 3$ to $bNow - 1$

$iLoc = loc(i)$

 If $LastChange(iLoc) < NowTime$ Then

(The solution in iLoc is still unchanged, so we can use it, otherwise the method skips it.)

 Let $x[iLoc]$ be the third solution in X

 For $j = i + 1$ to $bNow$

$jLoc = loc(j)$

 If $LastTime(jLoc) < NowTime$ then

 Let $x[jLoc]$ be the fourth solution in X

 Create $C(X)$ and execute the Improvement Method

 Endif

 End j

 Endif

End i

End Algorithm 3 Subroutine (if reach here)

Else

(Solutions in locations loc1 and loc2 are not new, since last time)

(If iNew > 1, then we look at all combinations of new with new.)

If $iNew > 1$ then

 For $i = 1$ to $iNew - 1$

$iLoc = LocNew(i)$

 If $LastChange(iLoc) < NowTime$ Then

 Let $x[iLoc]$ be the third solution in X

 For $j = i + 1$ to $iNew$

$jLoc = LocNew(j)$

 If $LastTime(jLoc) < NowTime$ then

 Let $x[jLoc]$ be the fourth solution in X

 Create $C(X)$ and execute the Improvement Method

 Endif

 End j

 Endif

 End i

Endif

If $jOld > 2$ then

 For $i = 1$ to $iNew$

$iLoc = LocNew(i)$

 If $LastChange(iLoc) < NowTime$ then

```

        Let x[iLoc] be the third solution in X
        For j = 3 to jOld
            jLoc = LocOld(j)
            If LastChange(jLoc) < NowTime then
                Let x[jLoc] be the fourth solution in X
                Create C(X) and Execute Improvement Method
            Endif
        End j
    Endif
End i
Endif
End Subroutine

```

Algorithm 4 Subroutine

Begin Subroutine

```

    new = 0
    For i = 1 to 4
        iLoc = loc(i)
        Let x[iLoc] be the ith solution in X
        If LastChange(iLoc) ≥ LastRunTime(SubsetType) then new = 1
    End i
    For i = 5 to bNow
        If LastChange(iLoc) ≥ LastRunTime(SubsetType) then new = 1
        If LastChange(iLoc) < NowTime then
            Let x[iLoc] be the ith solution in X
            If new = 1 then
                Create C(X) and execute the Improvement Method
            Endif
        Endif
    End i
End Subroutine

```

The preceding subroutines complete the collection for generating subsets of the Reference Set, without duplication. The comments within the subroutines should be sufficient to make their rationale visible, and to provide a basis for variations of the forms previously discussed.

7 Improvement Method

There often exist alternative neighborhoods of moves available to construct improvement methods for various kinds of optimization problems. Experience from numerous applications suggests that there is merit in using more than one such neighborhood. For example, a common theme of strategic oscillation, as applied in

tabu search, is to cycle among alternative neighborhoods according to various patterns. Strategic oscillation also commonly operates by cycling through various regions, or “levels” of a given neighborhood.

The approach of cycling through different levels of a neighborhood is manifest in two types of candidate list strategies, the Filtration Strategy and the Sequential Fan Strategy, proposed with tabu search (see, e.g., Glover and Laguna, 1997). The goal of these strategies is to identify attractive moves with an economical degree of effort. In addition, however, the Filtration and Sequential Fan strategies offer a useful basis for converting a simple Improvement Method into a more advanced one. We propose a way to marry these two candidate list strategies to create a *Filter and Fan Method* which provides a convenient form of an Improvement Method for the SS/PR Template.

We emphasize that other options exist for creating improvement methods that can be incorporated in the SS/PR Template. In settings where an effective improvement method already exists and has demonstrated its utility, such a procedure can be relied on directly and the primary concern will be to exploit it by the routines described in the preceding sections. Alternatively, components of such a procedure – as, for example, the moves it relies on and the evaluations it uses to choose among these moves – can be embedded within the general approach we introduce here.

Component Moves

The moves to serve as building blocks for the proposed method will characteristically be simple types of moves as illustrated by adjacent integer changes for integer variables (e.g., “flip” moves for 0-1 variables) and by elementary insert or swap moves for permutation problems. We call the chosen component moves level 1 moves, or *1-moves*.

An associated Level 1 Improvement Method can be defined relative to the 1-moves, which operates by segregating a collection of 1-moves by a preliminary candidate list approach, such as an Aspiration Plus strategy (Glover and Laguna, 1997). A random selection of such a collection is possible, in the interest of simplification, but at an appreciable risk of reducing overall effectiveness. (When randomization is used, the initial list should typically be larger than otherwise required.)

A useful goal for the initial candidate list strategy is to assure that a number of these 1-moves are among the highest evaluation moves currently available, so that if none of them is improving, the method is likely to be at a local optimum relative to these moves. The Level 1 Method then terminates when no moves from its candidate list are improving moves, thus presumably stopping at a local optimum or a “near” local optimum (relative to a larger collection of moves that encompasses those of the candidate list). The candidate list construction for Level 1 can be dynamic to allow the size of the list to grow when no improving move is found. (The Aspiration Plus strategy has this character, for example.) Such an optimum makes it possible to assure that termination will occur at a local optimum, if desired. The Filter and Fan Method then goes beyond this stopping point to create higher level moves. For this purpose,

we extract a subset M of some number of best moves from those examined by the Level 1 method when it terminates, where for example $|M| = 20$ or 40.

General Design.

The general design of the Filter and Fan Method is to isolate a subset $M(L)$ of the best moves at a given level L , to be used as a basis for generating more advanced moves at level $L+1$ when level L fails to yield an improving move. In case $L=1$, we choose $M(1)$ to be a subset of M .

Suppose that m is a given L -move from $M(L)$, and let $A(m)$, be a related set of 1-moves (derived from M) so that the result of applying any 1-move m' in $A(m)$, after applying m will create an $(L+1)$ -move which we denote by $m@m'$. By restricting $M(L)$ to consist of a relatively small number of the moves examined at Level L (e.g., choosing $|M(L)| = 10$ or 20), and likewise restricting $A(m)$ to consist of a relatively small number of 1-moves, the total number of $L+1$ moves $m@m'$ can be maintained at a modest size. For example, a steady state choice that always picks $|M(L)| = 16$ and $|A(m)| = 8$ (for each m in $M(L)$) will generate only 128 $(L+1)$ -moves to be examined at each level $L+1$. If none are improving, in this example the 16 best are selected to compose $M(L+1)$, and the process repeats.

The utility of this design is to avoid the combinatorial explosion of possibilities that results by generating the set of all possible $(L+1)$ -moves at each step. Instead the approach filters a subset $M(L)$ of best moves at level L , and for each of these moves likewise filters a set $A(m)$ of best 1-moves from M . The “fan” that generates $|M(L)||A(m)|$ potential $(L+1)$ -moves as candidates to examine at Level $L+1$ is therefore maintained to be of reasonable size.¹⁰

We say that $A(m)$ is *derived from* M , not only because $A(m)$ may be smaller than M , but also because some of the moves of M may not be legitimate once the L -move m in $M(L)$ is executed. The 1-moves available after applying move m may not precisely correspond to moves of the original set M . For example, if the 1-moves correspond to flipping the values of 0-1 variables, then a move m may have flipped values for several variables in M , and the corresponding 1-moves in M will no longer be accessible. However, it is generally easy to keep a record for each move m in $M(L)$ that identifies the moves of M that should be excluded from $A(m)$, allowing $A(m)$ to be composed of the best $|A(m)|$ remaining members of M . Similar comments apply to moves such as swap moves and insert moves.

A simple steady state version of the Filter and Fan method can be summarized as follows. Let n_0 be the chosen size of the initial M , n_1 be the size of each set $M(L)$ and n_2 be the size of each set $A(m)$ (where n_1 and n_2 do not exceed n_0). In many applications, n_0 will be at most 40 and n_1 and n_2 will be at most 20 (and smaller

¹⁰ The emphasis on controlling computational effort while producing good candidate moves can be facilitated in some settings by an accelerated (shortcut) evaluation process. This can occur by selecting members of an initial candidate list that provides the source of M , as illustrated by the use of surrogate constraint evaluations in place of a lengthier evaluation that identifies the full consequences of a move relative to all constraints of a problem. Accelerated evaluations can also be applied to isolating M from the initial candidate list, while reserving more extensive types of evaluations to isolating $M(1)$ from M , and to deriving $A(m)$ from M for the moves m generated at various levels.

values may be preferable). We call this version a *strict* improvement method because it does not allow steps that are nonimproving, and therefore terminates at a local optimum relative to the multilevel moves it employs.

Filter and Fan Strict Improvement Method

1. *Generate a candidate list of 1-moves for the current solution x .*
 - (a) If any of the 1-moves are improving: Choose the best member from the list and execute it to create a new current solution x . (The “best member” may be the only member if the list terminates with the first improving move encountered.) Then return to the start of step 1.
 - (b) If none of the 1-moves are improving: Identify the set M of the n_0 best 1-moves examined. Let $M(1)$ be a subset of the n_1 best moves from M , and let $X(1)$ be the set of solutions produced by these moves. Set $L = 1$ and proceed to step 2.
2. *For each L -move m in $M(L)$:* Identify the associated set $A(m)$ of the n_2 best compatible moves m' derived from M , and evaluate each resulting $(L+1)$ -move $m@m'$. (Equivalently, evaluate each solution that results by applying move m' to the corresponding member of $X(L)$.) When fewer than n_2 moves of M are compatible with move m , restrict consideration to this smaller set of moves in composing $A(m)$.
 - (a) If an improving move is found during the foregoing process: Select the best such move generated (by the point where the process is elected to terminate), and execute the move to create a new current solution x . Then return to step 1.
 - (b) If no improving move is found by the time all moves in $M(L)$ are examined: Stop if L has reached a chosen upper limit $\text{Max}L$. Otherwise, identify the set $M(L+1)$ of the n_1 best $(L+1)$ -moves evaluated (and/or identify the associated set $X(L+1)$). (If fewer than n_1 distinct $(L+1)$ -moves are available to be evaluated, then include all distinct $(L+1)$ -moves in $M(L+1)$.) Then set $L = L + 1$ and return to the start of Step 2.

The identification of $M(L+1)$ (and/or $X(L+1)$) in Step 2(b) can of course be undertaken as part of the process of looking for an improving move, rather than waiting until no improving move is found. The appropriate organization depends on the setting. Also, an evident option for executing the preceding method is to allow a variable-state version where n_1 and/or n_2 decreases as L increases, thereby reducing the number of candidates for successively higher levels of moves. Another option is to allow L to change its value by a different pattern. We now examine relevant considerations for implementing this method.

7.1 Avoiding Duplications

A slight change in the preceding description can improve the approach by avoiding the generation of a number of duplicate outcomes. Such duplications are especially

likely to arise when generating 2-moves, in the setting where solutions are generated by flipping 0-1 variables. To illustrate, suppose $n_0 = n_1 = n_2 = 20$. Thus, initially M consists of the 20 best 0-1 flips, and we consider the case where none are improving. Then at Step 2, having chosen $M(1) = M$ (since $n_1 = n_0$), the method as described would select each move m from $M(1)$ and extend it by applying a compatible 1-move m' taken from $A(m)$, where in this case $A(m)$ consists of all of M excluding the single flip that produced m . Thus, each move m in $M(1)$ would be matched with the 19 other 1-moves that constitute flips other than the one embodied in m . Over the 20 elements of $M(1)$ this yields $20 \times 19 = 380$ possibilities to evaluate. But this is double the number that is relevant, since each flip of two variables x_i and x_j will be generated twice — once when the x_i flip is in $M(1)$ and the x_j flip is in $A(m)$, and once when the x_j flip is in $M(1)$ and the x_i flip is in $A(m)$. Such duplication can easily be removed by restricting a flip of two variables x_i and x_j so that $j > i$, where x_i belongs to $M(1)$ and x_j belongs to $A(m)$. This indexing restriction may alternately be applied after the flips are sorted in order of their attractiveness. In either case, the result may be viewed as restricting the definition of $A(m)$.

Potential duplications for other types of moves can similarly be easily avoided at the level $L = 1$, where 2-moves are being generated in Step 2. In the case of swap and insert moves, a more balanced set of options can be created by restricting the number of moves recorded in M that involve any given element (or position). For example, if 5 of the 20 best swap moves involve swapping a given element i , then it may be preferable to record only the 2 or 3 best of these in M , and therefore complete the remainder of M with moves that may not strictly be among the 20 best.

As L grows larger, the chance for duplications drops significantly, provided they have been eliminated at the first execution of Step 2. Consequently, special restrictions for larger values of L can be disregarded. Instead, it is easier to screen for duplications at the point where a move becomes a candidate to include in $M(L+1)$ (or equivalently, the associated solution becomes a candidate to include in $X(L+1)$). The method for updating the Reference Set, given in the Section 3, can be used as a design to conveniently identify and eliminate such duplications in the present context as well.¹¹

7.2 Move Descriptions

The influence of move descriptions, where the same move can be characterized in alternative ways, can affect the nature of moves available independently of the neighborhood used. This phenomenon is worth noting, because standard analyses tend to conceive the neighborhood structure as the sole determinant of relevant

¹¹ This can be made simpler by recording “incremental solutions” — or more precisely the representations of solutions in $X(L)$ that result when the current solution x is represented as the zero vector — since these will generally have few nonzero components, and may be stored and accessed more quickly than complete solution vectors.

outcomes. The phenomenon arises in the Filter and Fan method because the move description implicitly transmits restrictions on deeper level moves in a manner similar to the imposition of *tabu* restrictions in tabu search. Thus, an implicit memory operates by means of the attributes of the moves considered, and these attributes depend on the move description.

To illustrate, a description that characterizes an insert move to consist of inserting element i in position p (and shifting other elements appropriately) can yield a different outcome, once intervening moves are made, than a description that characterizes the same move as inserting element i immediately before an element v . (Note that a more restrictive description, such as specifying that the move consists of inserting element i between elements u and v , may render the move impossible once either u or v changes its position.)

The phenomenon can be usefully demonstrated for swap moves by the situation where two moves in M are respectively characterized as swapping elements i and j and swapping elements i and k . After performing the first swap, the second will receive a changed evaluation, since i is no longer in the same position. If instead the same moves are characterized as swapping the elements in positions p and q , and in positions p and r (where elements i , j and k are currently in these positions), then the result of the first swap gives a different outcome for the second swap than the one illustrated previously; that is, the second swap now corresponds to swapping elements j and k rather than i and k . (Still another outcome results if a swap is characterized as a double insert move, e.g., as inserting an element i immediately after the current predecessor of j and inserting j immediately after the current predecessor of i .)

A preferable description of course depends in part on the nature of the problem. An interesting possibility is to allow two (or more) move characterizations, and then to choose the one in a given situation that yields the best result. This is analogous to allowing different solution attributes to define potential restrictions by the attribute-based memory of tabu search.

By the same token, it may be seen that greater flexibility can be obtained simply by relaxing the definition of a move. For example, in the setting of 0-1 problems, instead of characterizing M as a set of value-specific flips (as illustrated by stipulating that x_j should change from 0 to 1), we can allow M to be value-independent (as illustrated by stipulating that x_j should change to $1 - x_j$). The value-independent characterization allows greater latitude for generating moves from M , and is relevant as L grows beyond the value of 1. Such a characterization should be accompanied by a more explicit (and structured) use of tabu search memory, however, to control the possibility of cycling. The value-specific characterization is sufficiently limiting to avoid this need in the present illustration.

Another degree of latitude exists in deriving $A(m)$ from M . Suppose the moves of M are denoted $m(1)$, $m(2)$, ..., $m(u)$, where the moves with smaller indexes have higher evaluations. If we stipulate that $A(m)$ should consist of the r best of these moves, restricted to those that are compatible with m , we may choose to avoid some computation by ordering M in advance, as indicated, and then simply selecting the first r compatible members to compose $A(m)$. However, since the relative

attractiveness of the moves in M may change once the move m is made, an alternative strategy is instead to examine some larger number of compatible members m' of M to improve the likelihood of including the “true r best” options for the compound moves $m@m'$. This of course does not change the form of the method, since it amounts to another possibility for choosing the size of $|A(m)|$. However, this observation discloses that it may be preferable to choose the size of $|A(m)|$ to be larger relative to the size of $|M(L)|$ than intuition may at first suggest.

7.3 Definitions of 1-moves and the Composition of M

It is entirely possible in the application of the Filter and Fan Method that a 1-move may be defined in such a way to produce an infeasible solution, but a coordinated succession of 1-moves will restore feasibility. A simple example is provided by the graph partitioning problem, where feasible solutions consist of partitioning a set of $2n$ nodes into two sets that contain n nodes each. A 1-move that consists of swapping two nodes that lie in different sets will maintain feasibility, but a 1-move that consists of moving a node from one set to the other will not. Nevertheless, since the second (simpler) 1-moves are many fewer in number, a candidate list strategy that identifies attractive moves of this type, differentiated according to the set in which they originate, also provides a useful basis for composing compound moves by the Filter and Fan Method. In this situation, the successive 1-moves must alternately be chosen from different sets, and feasible solutions only occur for even values of L . Such implementations are easily accommodated within the general framework, and we do not bother to introduce more complicated notation to represent them.

Similarly, a method that applies 0-1 flips to an integer programming problem may allow M to include flips that create infeasibility (as in a multidimensional knapsack problem where any flip from 0 to 1 will drive a “boundary solution” infeasible). Rather than avoiding moves that produce infeasibility, the process may make provision for an infeasibility generated at a level L to be countered by focusing on moves to recover feasibility at level $L + 1$. (The focus may be extended to additional levels as necessary.)

A related but more advanced situation arises in ejection chain strategies where 1-moves may be defined so that no sequence of them produces a feasible solution. In this type of construction a reference structure guides the moves selected to assure that a feasible solution can always be generated by one or more associated trial solutions (see, e.g., Glover, 1992; Rego 1996; Rego and Roucairol, 1996). In these instances the Filter and Fan Method can accordingly be modified to rely on the trial solutions as a basis for evaluating the L -moves generated.

The composition of M can be affected by an interdependency among subcollections of moves. In certain pivot strategies for network flow optimization, for example, once the best pivot move associated with a given node of the network is executed, then the quality of all other moves associated with the node deteriorates. Thus, instead of choosing M to consist of the n_0 best moves overall, which could possibly include several moves associated with the same node, it can be preferable to allow M to

include only one move for any given node. More generally, M may be restricted by limiting the numbers of moves of different categories that compose it.

The potential to modify M , $M(L)$ and $A(m)$ as the method progresses can be expanded by allowing these sets to be “refreshed” by a more extensive examination of alternatives than those earmarked for consideration when $L = 1$. For example, in some settings the execution of a move m may lead to identifying a somewhat restricted subset of further moves that comprise the only alternatives from which a compound improving move can be generated at the next level. In such a case, M and $A(m)$ should draw from such alternatives even though they may not be encompassed by the original M . Such an expansion of M and its derivative sets must be carefully controlled to avoid losing the benefit of reducing overall computation that is provided by a more restricted determination of these sets.

Allowing for such an expansion can increase the appropriate value of $MaxL$. Or inversely, the reliance on smaller sizes of M , $M(L)$ and $A(m)$ can decrease the appropriate value of $MaxL$. A reasonable alternative is to determine $MaxL$ indirectly by a decision to terminate when the quality of the best current L -moves (or of the 1-moves that are used to generate these L -moves) drops below a chosen threshold. These considerations are relevant to the “partial refreshing” process of the Multi-Stream variant, described in the next section.

7.4 Advanced Improvement Alternatives

The preceding Strict Improvement version of the Filter and Fan method has a conspicuous limitation that can impair the quality of the solutions it produces. In the absence of applying a refreshing option, which can entail a considerable increase in computation or complexity, the merit of the set M as the source of the sets $M(L)$ and $A(m)$ diminishes as L grows. This is due to the fact that M is composed of moves that received a high evaluation relative to the current solution before the compounding effects of successive 1-moves is considered, and as these moves are drawn from M for progressively larger values of L , the relevance of M as the source of such moves deteriorates. Under such circumstances the value of $MaxL$ will normally not be usefully chosen to be very large (and a value as small as 4 or 5 may not be atypical).

This limitation of the strict improvement method can be countered by one of three relatively simple schemes:

- (1) A Shift-and-Update variant.
- (2) A Diversification variant.
- (3) A Multi-Stream variant.

We examine these three variants as follows.

Shift-and-Update Variant

The Shift-and-Update method operates by choosing a nonimproving move when the method would otherwise terminate, but shifting the outcome away from the lowest levels to insure that the choice will yield a new solution that lies some minimum number of 1-moves “away from” the current solution x . The new solution is then

updated to become the current solution, and the method repeats. The method can be described by the following rule.

Shift-and-Update Rule

Identify a value MinL such that $1 < \text{MinL} \leq \text{MaxL}$. When no improvement is found upon reaching $L = \text{MaxL}$, select the best solution found over the levels $L \geq \text{MinL}$. Then specify this solution to be the current solution x and return to the beginning of Step 1.

Simple recency-based tabu search memory can be used with this rule to keep from returning to a preceding solution and generally to induce successively generated solutions to continue to move away from regions previously visited. This variant creates a degree of diversification which may be increased by restricting attention to value-specific move descriptions, so that each set $M(L)$ will be progressively farther removed from $M(1)$.

Diversification Variant

A more ambitious form of diversification, which can be applied independently or introduced as a periodic alternative to the Shift-and-Update approach, operates by temporarily replacing the steps of the Filter and Fan Method with a series of steps specifically designed to move away from regions previously visited. We express this approach by the following rule.

Diversification Rule

Introduce an explicit Diversification Stage when the approach fails to find an improvement (and terminates with $L = \text{MaxL}$). In this stage change the evaluator to favor new solutions whose attributes differ from those of solutions previously encountered, and execute a chosen number MinL of successive 1-moves guided by the changed evaluator. Immediately following this stage, the next pass of the Filter and Fan Method (starting again from Step 1 with a new x) uses the normal evaluator, without a diversification influence.

This diversification approach can use tabu search frequency memory to encourage moves that introduce attributes that were rarely or never found in solutions previously generated, and similarly to encourage moves that eliminate attributes from the current solution that were often found in previous solutions. An alternative is to apply the Diversification Stage directly within the Filter and Fan structure, where M and its derivative sets are generated by keeping the value of n_0 , and especially the values n_1 and n_2 , small. The evaluator in this case may alternatively be a composite of the normal evaluator and a diversification evaluator, with the goal of producing solutions that are reasonably good as well as somewhat different from previous solutions. Then the values of n_0 , n_1 and n_2 may be closer to those during a non-diversification stage. For a simpler implementation, instead of using frequency memory in the

Diversification Stage, the approach can be applied as noted in Section 1 by introducing a Post-Improvement step that uses the Diversification Generator.

Multi-Stream Variant

The Sequential Fan candidate list strategy embodies an additional element, not yet considered, that consists of generating several solution streams simultaneously. The inclusion of this element in the Filter and Fan Method produces a method that increases the likelihood of finding improved solutions. The ideas underlying this variant may be sketched in overview as follows.

Starting from the current solution x in Step 1 of the Filter and Fan Method, the approach selects a small number s of the best moves from the set M . Each selected move is allowed to initiate a different stream by creating an associated set of additional moves to launch the rest of the Filter and Fan Method. The outcome effectively shifts the Filter and Fan Method by one step, so that the solutions produced by these s moves take the place of solution x to produce additional levels of moves.

Specifically, let $x[i]$, $i \in S = \{1, \dots, s\}$ denote the solutions produced by the s best moves derived from x . The evaluations for each $x[i]$ are updated to generate a refreshed candidate list for each. Let $M(0:i)$ denote the set of n_0 best 1-moves from the candidate list created for $x[i]$ (in the same way that M represents the set of n_0 best 1-moves from the candidate list created for x). Similarly, let $M(L:i)$ and $X(L:i)$, $i \in S$, denote the sets of moves and corresponding sets of solutions for an arbitrary level $L \geq 1$.

The Multi-Stream variant then operates exactly as the single stream variant, by allowing each $x[i]$ for $i \in S$ to take the role of x , except that the computation is restricted in a special way. The purpose of this restriction is to allow the Multi-Stream variant to require only slightly more effort than the single stream variant, except for the initial step that identifies a fresh candidate list for each $x[i]$. (To achieve greater speed, an option is to forego this identification and instead rely on choosing each $M(0:i)$ as a subset of an enlarged set M generated for x . The relevance of this alternative is determined by considerations discussed in Section 3.)

The restricted treatment of the sets $M(L:i)$ at each stage of the Multi-Stream variant produces the following modifications of Steps 1 and 2 of the Filter and Fan Method.

Multi-Stream Implementation

1A. (Modification of step 1.)

- (a) As in the original step 1(a), return to the start of step 1 with a new solution x if an improving move is identified while examining the candidate list for x . Similarly, if no such improving move is found, but an improving move is identified in the process of examining the candidate list for one of the solutions $x[i]$, $i \in S$, then choose the best such move (up to the point where the process is elected to discontinue), and return to the start of step 1.
- (b) If no improving moves are found in step 1A (a), then amend the approach of the original step 1 (b) as follows. Instead of creating each set $M(1:i)$ as the n_1 best moves from $M(0:i)$, coordinate the streams so that the entire

collection $M(1:i)$, $i \in S$, retains only the $n1$ best moves from the collection $M(0:i)$, $i \in S$. (Thus, on average, each $M(1:i)$ contains only $n1/s$ moves. Some of these sets may be empty.) Then set $L = 1$ and proceed to step 2A.

2A. (*Modification of step 2*). Consider each $i \in S$ such that $M(L:i)$ is not empty. For each L-move in $M(L:i)$ identify the associated set $A(m)$ of the $n2$ best compatible 1-moves taken from $M(0:i)$, to produce candidate $(L+1)$ -moves of the form $m@m'$ for $m' \in A(m)$.

- (a) If an improving $(L+1)$ -move is found, select such a move as in the original step 2(a) and return to the start of step 1 with a new x .
- (b) If no improving move is found while examining the set of moves in the collection $M(L:i)$, $i \in S$: Stop if L has reached $MaxL$. Otherwise, identify the collection $M(L+1:i)$, $i \in S$, to consist of the best $n1$ moves generated from the entire collection $M(L:i)$, $i \in S$. Then set $L = L + 1$ and return to the start of step 2A.

The Multi-Stream variant requires some monitoring to assure that the sets $M(L:i)$ do not contain duplicate L-moves (i.e., the sets $X(L:i)$ do not contain duplicate solutions), or to assure that such duplications are removed when they occur. Simple forms of TS memory can be used for this purpose, or again the type of screening used to update the set of Reference Solutions for the SS/PR Template (as described in Section 3) can be employed with the method. Special restrictions for level $L = 1$, as previously noted for the single-stream case discussed in the first part of Section 5, can be readily extended to the Multi-Stream case.

The Multi-Stream approach can be joined with the Shift-and-Update variant or the Diversification variant, and is particularly susceptible to being exploited by parallel processing.

8 Conclusions

This paper is intended to support the development of scatter search and path relinking methods, by offering highly specific procedures for executing component routines. We have also undertaken to indicate a type of Improvement Method that exploits the marriage of two candidate list strategies – the Filtration and Sequential Fan strategies – that have been proposed in connection with tabu search, and which invite further examination in their own right. We emphasize that there are additional ways to implement scatter search and path relinking, and our purpose is not to be exhaustive in considering the best alternatives. Nevertheless, the SS/PR Template and its subroutines offer a potential to facilitate the creation of initial methods and to reduce the effort involved in creating additional refinements.

REFERENCES

1. Consiglio, A. and S.A. Zenios (1996). "Designing Portfolios of Financial Products via Integrated Simulation and Optimization Models," Report 96-05, Department of Public and Business Administration, University of Cyprus, Nicosia, CYPRUS, to appear in *Operations Research*. [<http://zeus.cc.ucy.ac.cy/ucy/pba/zenios/public.html>]
2. Consiglio, A. and S.A. Zenios (1997). "a Model for Designing Callable Bonds and its Solution Using Tabu Search." *Journal of Economic Dynamics and Control* 21, 1445-1470. [<http://zeus.cc.ucy.ac.cy/ucy/pba/zenios/public.html>]
3. Crowston, W.B., F. Glover, G.L. Thompson and J.D. Trawick (1963). "Probabilistic and Parametric Learning Combinations of Local Job Shop Scheduling Rules," ONR Research Memorandum No. 117, GSIA, Carnegie Mellon University, Pittsburgh, PA
4. Cung, V-D., T. Mautor, P. Michelon, A. Tavares (1996). "Scatter Search for the Quadratic Assignment Problem", Laboratoire PRISM-CNRS URA 1525. [http://www.prism.uvsq.fr/public/vdc/CONFS/ieee_icec97.ps.Z]
5. Davis, L., ed. (1991). *Handbook of Genetic Algorithms*, Van Nostrand Reinhold.
6. Fisher, H. and G.L. Thompson (1963). "Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules," *Industrial Scheduling*, J.F. Muth and G.L. Thompson, eds., Prentice-Hall. 225-251.
7. Fleurent, C., F. Glover, P. Michelon and Z. Valli (1996). "A Scatter Search Approach for Unconstrained Continuous Optimization," *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*. 643-648.
8. Freville, A. and G. Plateau (1986). "Heuristics and Reduction Methods for Multiple Constraint 0-1 Linear Programming Problems," *European Journal of Operational Research*, 24, 206-215.
9. Freville, A. and G. Plateau (1993). "An Exact Search for the Solution of the Surrogate Dual of the 0-1 Bidimensional Knapsack Problem," *European Journal of Operational Research*, 68, 413-421.
10. Glover, F. (1963). "Parametric Combinations of Local Job Shop Rules," Chapter IV, ONR Research Memorandum no. 117, GSIA, Carnegie Mellon University, Pittsburgh, PA.
11. Glover, F. (1965). "A Multiphase Dual Algorithm for the Zero-One Integer Programming Problem," *Operations Research*, Vol 13, No 6, 879.
12. Glover, F. (1968). "Surrogate Constraints," *Operations Research*, 16, 741-749.
13. Glover, F. (1975). "Surrogate Constraint Duality in Mathematical Programming," *Operations Research*, 23, 434-451.
14. Glover, F. (1977). "Heuristics for Integer Programming Using Surrogate Constraints," *Decision Sciences*, Vol 8, No 1, 156-166.
15. Glover, F. (1992). "Ejection Chains, Reference Structures and Alternating Path Methods for Traveling Salesman Problems," University of Colorado. Shortened version published in *Discrete Applied Mathematics*, 1996, 65, 223-253. [<http://spot.colorado.edu/~glover> (under Publications)]
16. Glover, F. (1994a). "Genetic Algorithms and Scatter Search: Unsuspected Potentials," *Statistics and Computing*, 4, 131-140. [<http://spot.colorado.edu/~glover> (under Publications)]
17. Glover, F. (1994b). "Tabu Search for Nonlinear and Parametric Optimization (with Links to Genetic Algorithms)," *Discrete Applied Mathematics*, 49, 231-255. [<http://spot.colorado.edu/~glover> (under Publications)]
18. Glover, F. (1995). "Scatter Search and Star-Paths: Beyond the Genetic Metaphor," *OR Spectrum*, 17, 125-137. [<http://spot.colorado.edu/~glover> (under Publications)]

19. Glover, F., J. P. Kelly and M. Laguna (1996). "New Advances and Applications of Combining Simulation and Optimization," *Proceedings of the 1996 Winter Simulation Conference*, J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain (Eds.), 144-152. [<http://spot.colorado.edu/~glover> (under OptQuest heading)]
20. Glover, F. and M. Laguna (1997). *Tabu Search*, Kluwer Academic Publishers. [<http://spot.colorado.edu/~glover> (under Tabu Search heading)]
21. Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*, Reading, Massachusetts: Addison-Wesley.
22. Greenberg, H. J. and Pierskalla, W.P. (1970). "Surrogate Mathematical Programs," *Operations Research*, 18, 924-939.
23. Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI.
24. Karwan, M.H. and R.L. Rardin (1976). "Surrogate Dual Multiplier Search Procedures in Integer Programming," School of Industrial Systems Engineering, Report Series No. J-77-13, Georgia Institute of Technology.
25. Karwan, M.H. and R.L. Rardin (1979). "Some Relationships Between Lagrangean and Surrogate Duality in Integer Programming," *Mathematical Programming*, 17, 230-334.
26. Kelly, J., B. Rangaswamy and J. Xu (1996). "A Scatter Search-Based Learning Algorithm for Neural Network Training," *Journal of Heuristics*, Vol. 2, pp. 129-146.
27. Laguna, M. and R. Martí (1997). "GRASP and Path Relinking for 2-Layer Straight Line Crossing Minimization," Research Report, University of Colorado [<http://www.bus.colorado.edu/Faculty/Laguna/Papers/crossmin.html>]
28. Laguna, M. (1997). "Optimizing Complex Systems with OptQuest," Research Report, University of Colorado, [<http://www.bus.colorado.edu/Faculty/Laguna/Papers>]
29. Laguna, M., R. Martí and V. Campos (1997). "Tabu Search with Path Relinking for the Linear Ordering Problem," Research Report, University of Colorado. [<http://www.bus.colorado.edu/Faculty/Laguna/Papers/lop.html>]
30. Muhlenbein, H. (1997). "The Equation for the Response to Selection and its Use for Prediction," to appear in *Evolutionary Computation*. [<http://set.gmd.de/AS/ga/ga.html>]
31. Rana, S. and D. Whitley (1997). "Bit Representations with a Twist," Proc. 7th International Conference on Genetic Algorithms, T. Baeck ed. pp: 188-196, Morgan Kaufman. [<http://www.cs.colostate.edu/~whitley/Pubs.html>]
32. Rego, C. (1996). "Relaxed Tours and Path Ejections for the Traveling Salesman Problems," to appear in the *European Journal of Operational Research*. [<http://www.uportu.pt/~crego>]
33. Rego, C. and C. Roucairol (1996). "A Parallel Tabu Search Algorithm Using Ejection Chains for the Vehicle Routing Problem," in *Meta-Heuristics: Theory & Applications*, 661-675, I.H. Osman and J.P. Kelly, (eds.), Kluwer Academic Publishers. [<http://www.uportu.pt/~crego>]
34. Reeves, C.R. (1997). "Genetic Algorithms for the Operations Researcher," to appear in *INFORMS Journal on Computing* (with commentaries and rejoinder).
35. Rochat, Y. and É. D. Taillard (1995). "Probabilistic diversification and intensification in local search for vehicle routing". *Journal of Heuristics* 1, 147-167. [<http://www.idsia.ch/~eric>]
36. Taillard, É. D. (1996). "A heuristic column generation method for the heterogeneous VRP", Publication CRT-96-03, Centre de recherche sur les transports, Université de Montréal. To appear in *RAIRO-OR*. [<http://www.idsia.ch/~eric>]
37. Whitley, D. and J. Kauth. (1988). "GENITOR: A Different Genetic Algorithm," Proceedings of the 1988 Rocky Mountain Conference on Artificial Intelligence.

38. Whitley, D. (1989). The GENITOR Algorithm and Selective Pressure: Why Rank Based Allocation of Reproductive Trials is Best, Morgan Kaufmann, J. D. Schaffer, ed., pp. 116-121.
39. Yamada, T. and C. Reeves (1997). "Permutation Flowshop Scheduling by Genetic Local Search," 2nd IEE/IEEE Int. Conf. on Genetic Algorithms in Engineering Systems (GALESIA '97), pp. 232-238, Glasglow, UK.
40. Yamada, T. and R. Nakano (1996). "Scheduling by Genetic Local Search with Multi-Step Crossover," 4th International Conference on Parallel Problem Solving from Nature, 960-969.

APPENDIX 1 Construction-by-Objective: Mixed Integer and Nonlinear Optimization

The generality of the zero-one diversification generators of Section 3 can be significantly enhanced by a *construction-by-objective* approach, which makes it possible to generate points that satisfy constraints defining a polyhedral region. This approach is particularly relevant for mixed integer programming (MIP) problems and nonlinear optimization problems that include linear constraints. The resulting solutions can also be processed by standard supporting software to produce associated points that satisfy integer feasibility conditions that accompany MIP problems.

The approach stems from the observation that a specified set of vectors produced by a zero-one diversification generator can instead be generated by introducing appropriately defined objective functions which are optimized over the space of 0-1 solutions. These same objective functions can then be introduced and optimized, either exactly or heuristically, over other solution spaces, to create a more general process for generating trial solutions.

Denote the set of x vectors that satisfy a specified set of constraining conditions by XC , and denote the set of zero-one vectors by $X(0,1)$. We then identify a *construction function* $f(x)$ relative to a complementary pair of 0-1 solutions x' and x'' so that

$$x' = \operatorname{argmax}(f(x) : x \in X(0,1))$$

$$x'' = \operatorname{argmin}(f(x) : x \in X(0,1)).$$

Such a function can be given by any linear function $f(x) = \sum(f_j : j \in N)$, where $f_j > 0$ if $x'_j = 1$ and $f_j < 0$ if $x'_j = 0$. Let $H\text{-argmax}$ and $H\text{-argmin}$ denote heuristic counterparts of the argmax and argmin functions, which are based on applying heuristic methods that approximately maximize or minimize $f(x)$ over the more complex space XC . Then we define solutions $xMax$ and $xMin$ in XC that correspond to x' and x'' in $X(0,1)$ by

$$xMax = H\text{-argmax}(f(x) : x \in XC)$$

$$xMin = H\text{-argmin}(f(x) : x \in XC)$$

If $f(x)$ is a linear function as specified above, and XC corresponds to a bounded feasible linear programming (LP) region, which we denote by XLP , then the method that underlies $H\text{-argmax}$ and $H\text{-argmin}$ can in fact be an exact method for LP problems, to produce points $xMax$ and $xMin$ that are linear optima over XLP . This observation provides the foundation for a procedure to generate trial solutions for MIP problems.

Following the scatter search strategy of generating weighted centers of selected subregions, we apply the construction-by-objective approach by first identifying a small collection of *primary centers*. These constitute simple centers given by the midpoints of line segments that join pairs of points $xMin$ and $xMax$, together with the

center of all such points generated. Additional *secondary centers* are then produced by an accelerated process that does not rely on the exact solution of LP problems, but employs the primary centers to create approximate solutions. Both processes are accompanied by identifying *subcenters*, which are weighted centers of subregions composed by reference to the primary and secondary centers, and include the boundary points $xMax$ and $xMin$.

Method to Create Primary Centers

1. Identify a construction function $f(x)$ and initialize the set PC of primary centers and the set SubC of subcenters to be empty. Apply a Diversification Generator to produce a small collection of diversified points $x' \in X(0,1)$.
For each point generated, execute the following steps.
2. Identify points $xMax$ and $xMin$ by maximizing and minimizing $f(x)$ over XLP .
3. Let $xCenter = (xMax + xMin)/2$ and add $xCenter$ to PC. Also, create associated subcenters $xSubCenter = xMax + w(xMax - xMin)$, where the scalar weight w takes the values 0, 1/4, 3/4 and 1, and add these subcenters to the set SubC.
4. After the chosen x' vectors have been generated and processed, create a final point $xCenter^*$ which is the mean of all points $xMin$ and $xMax$ generated, and add $xCenter^*$ to PC.

The accelerated process for generating secondary centers, which augments the preceding method, continues to rely on a zero-one diversification generator to create 0-1 points $x' \in X(0,1)$, but bypasses LP optimization. The 0-1 points are first mapped into points of an associated region $X(0,U)$, where U is a vector of upper bounds chosen to assure that $0 \leq x \leq U$ is satisfied by all solutions $x \in XLP$. Then each point $x' \in X(0,1)$ produces a corresponding point $xTest \in X(0,1)$ by specifying that $x'_j = 0$ becomes $xTest_j = 0$, and $x'_j = 1$ becomes $xTest_j = U_j$.

Since these created points $xTest \in X(0,U)$ are likely to lie outside XLP , we map them into points of XLP by reference to the elements of PC. For a given $xTest$ and a given primary center $xCenter \in PC$, identify the line

$$L(w) = xTest + w(xCenter - xTest)$$

where w is a scalar parameter. Then a simple calculation identifies the values

$$wMin = \text{Min}(w : L(w) \in XLP)$$

$$wMax = \text{Max}(w : L(w) \in XLP)$$

The associated points of XLP , which are given by

$$xNear = L(wMin)$$

$$xFar = L(wMax),$$

constitute the two points of XLP nearest to and farthest from $xTest$ on the line through $xCenter$.

The secondary centers are chosen as midpoints of the line segments joining the endpoints $xNear$ and $xFar$. However, since such line segments may be created from more than one primary center $xCenter$, we limit consideration for a given $xTest$ to the

primary center that creates the longest of the indicated segments, i.e., which maximizes $D(xNear, xFar)$, where D is a distance metric such as the L1 or L2 norm. The method may be described as follows.

Method for Generating Secondary Centers

1. Generate a collection of points $x' \in X(0,1)$ and map each into an associated point $xTest \in X(0,U)$. Begin with the set SC of secondary solutions empty.
2. For each $xTest$, choose one or more elements $xCenter$ from PC, including $xCenter^*$.
3. For each $xCenter$ chosen, identify the pair of solutions $xNear$ and $xFar$ (by reference to $xCenter$ and $xTest$), and identify the specific pair $xNear^*$ and $xFar^*$ to be the pair that maximizes $D(xNear, xFar)$.
4. Let $xNewCenter = (xNear^* + xFar^*)/2$, and add $xNewCenter$ to SC. Also, create associated subcenters given by $xSubCenter = xNear + w(xFar - xNear)$, as the scalar weight w takes the values 0, 1/4, 3/4 and 1, and add these to SubC.

To take advantage of generating larger numbers of points x' by the Diversification Generator, the foregoing process may be modified to create each secondary center from multiple $xTest$ points. Specifically, for a chosen number $TargetNumber$ of such points, accumulate a vector $xSum$ that starts at 0 and is incremented by $xSum = xSum + xNear^* + xFar^*$ for each pair of solutions identified in Step 3 of the foregoing procedure. Once $TargetNumber$ of such points have been accumulated, set $xNewCenter = xSum/TargetNumber$, and start again with $xSum$ equal to the zero vector. The overall primary center $xCenter^*$ may optionally be updated in a similar fashion.

The values of w for generating subcenters can be varied, and used to generate more or fewer points. An alternative to choosing $xNear^*$ and $xFar^*$ by maximizing $D(xNear, xFar)$ is to specify that $xNear^*$ is selected to minimize $D(xTest, xNear)$. This latter criterion is also relevant for mapping an LP-infeasible point $xTest$ generated by a scatter search combination process into an LP-feasible point $xNear^*$, similarly using candidate points $xCenter$ as a basis for determining associated points $xNear$. The process for transforming the points ultimately selected into solutions that are also integer-feasible is examined next.

Creating Reference Solutions

The primary centers, secondary centers and subcenters provide the source solutions which, after improvement, become candidates for the set RefSet of reference solutions to be combined by scatter search. To create an initial RefSet, we cull out an appropriately dispersed subset of these source solutions by creating a precursor set PreRefSet, whose members will be made integer-feasible and hence become full fledged candidates for RefSet. This may be done by starting with the primary center $xCenter^*$ as the first element of PreRefSet, and then applying the criterion indicated in

Section 3 to choose each successive element of PreRefSet to be a source solution that maximizes the minimum distance from all solutions thus far added to PreRefSet.

Once PreRefSet reaches its targeted size, the final step is to modify its members by an adaptive rounding process that yields integer values for the discrete variables. For problems small enough that standard MIP software can be expected to generate feasible MIP solutions within a reasonable time, such software can be used to implement the rounding process by introducing an objective function that minimizes the sum of deviations of the integer values from the initial values. (The deviations may be weighted to reflect the MIP objective, or to embody priorities as used in tabu search.) Alternatively, a simple adaptive scheme of the following form can be used to exploit such an objective.

Adaptive Rounding Method to Create Reference Solutions

1. For each integer-infeasible variable in a given candidate solution, introduce a bound at an integer value neighboring its current value, and establish a large LP penalty for deviating from this bound. (Variables not constrained to be integer retain their ordinary objective function coefficients, or can be mildly penalized for deviating from their current values.)
2. Find an LP optimal solution for the current objective. If all integer-constrained variables receive integer values, stop. (The resulting solution is the one sought.)
3. If the LP solution is not MIP feasible, use postoptimality penalty calculations for each integer-constrained variable to identify the cost or profit that results by releasing its current penalty and seeking to drive the variable to a new bound at the closest integer value in the opposite direction from its current bound. Choose the variable that yields the greatest profit (or smallest cost) and impose a penalty for deviating from the indicated new bound. Then return to Step 2.

A natural priority scheme for Step 3 of the preceding approach is to give preference to selecting integer-infeasible variables to be those driven to new bounds, though this type of priority is not as relevant as it is in branch and bound methods. A simple tabu search memory can be used as a basis for avoiding cycling, while allowing greater flexibility of choices than provided by branch and bound.

After applying such a rounding procedure, the transformed members of PreRefSet are ready to be submitted to the customary scatter search processes of applying an Improvement Method and generating combined solutions. (Improvement heuristics can be included as part of the transformation process.) Future diversification phases of the search may be launched at various junctures by following the same pattern as indicated, where the diversification generator may be started from the point where it was discontinued in the preceding phase, or re-started by reference to a different seed solution. The initial PreRefSet for such future phases is populated by elements chosen to maximize the minimum distance from the collection of points previously generated as members of RefSet and PreRefSet, as well as from members currently added to PreRefSet.

APPENDIX 2: Checking for Duplicate Solutions

An additional source of potential duplications arises among solutions x' that are generated as combinations of other solutions (in the phase of Scatter Search or Path Relinking that generates such combined solutions). These solutions x' are inputs to the Improvement Method rather than outputs of this method. By the philosophy of scatter search and path relinking, it is valuable to avoid duplications in these input solutions as well as to avoid duplications in the solutions saved in RefSet. To do this, we store only the $r = rNow$ most recent solutions generated (allowing $rNow$ to grow to a maximum of $rMax$ different solutions recorded), following a scheme reminiscent of a simple short-term recency memory approach in tabu search. In particular, we keep these solutions in an array $xsave[r]$, $r = 1$ to $rNow$, and also keep track of a pointer $rNext$, which indicates where the next solution x' will be recorded once the array is full, i.e., once all $rMax$ locations are filled.

Let $E0$ and $Hash0$ be defined for x' as before, and denote associated values for the $xsave[r]$ array by $Esave(r)$ and $Hashsave(r)$. These are accompanied by a “depth” value, which is 0 if no duplication occurs, and otherwise tells how deep in the list – how far back from the last solution recorded – a duplication has been found. For example, $depth = 3$ indicates that the current solution duplicates a solution that was recorded 3 iterations ago. (This is not entirely accurate, since, for example, $depth = 3$ could mean the solution was recorded 5 iterations ago and then 2 other duplications occurred, which still results in recording only 3 solutions.)

An appropriate value for $rMax$ can be determined by initial testing that sets this value larger than expected to be useful. An array $CountDup(depth)$, for $depth = 1$ to $rMax$, can then be kept that counts how often duplications are found at various depths. If the array discloses that very few duplications occur for depths beyond a given value, then $rMax$ can be reduced to such a value, without the risk of having to process many solutions that duplicate others encountered. (Although the reduced value of $rMax$ will save some effort checking for duplications, it may be the effort will not be too great anyway, if a quick check based on using $Hash0$ can screen out most of the potential duplications.)

To keep track of auxiliary information we introduce counters corresponding to $DupCheck$, $FullDupCheck$ and $FullDupFound$ of the RefSet Update Routine, which we give the names $DupCheckA$, $FullDupCheckA$, and $FullDupFoundA$. Finally, we keep track of the number of times the routine is called by a value $DupCheckCall$.

Initialization Step:

```

rNow = 0
rNext = 0
CountDup(depth) = 0, for depth = 1 to rMax
DupCheckA = 0
FullDupCheckA = 0

```


FullDupFoundA = 0
 DupCheckCall = 0

Duplication Check Subroutine.

Begin Subroutine.

DupCheckCall = DupCheckCall + 1
 depth = 0

If rNow = 0 then:

 rNow = 1; rNext = 1;
 xsave[1] = x' (record x' in xsave[1]),
 Esave(1) = E0; Firstsave(1) = FirstIndex0
 End the Subroutine

Elseif rNow > 0 then:

(Go through the solutions in "depth order", from the one most recently stored to the one least recently stored. When a duplication is found, the loop index r (below) indicates the value of rMax that would have been large enough to identify the duplication.)

 i = rNext

 For r = 1 to rNow

 If Esave(i) = E0 then:

 DupCheckA = DupCheckA + 1

 If Hash0 = Hashsave(i) then:

 FullDupCheckA = FullDupCheckA + 1

 If $x' = x[i]$ then:

 (x' duplicates a previous solution)

 FullDupFoundA = FullDupFoundA + 1

 depth = r

 CountDup(depth) = CountDup(depth) + 1

 End the Duplication Check Subroutine

 Endif

 Endif

 Endif

 i = i-1

 if i < 1 then i = rNow

End r

(Here, no solutions were duplicated by x' . Add x' to the list in position rNext, which will replace the solution previously in rNext if the list is full.)

 rNext = rNext + 1

 If rNext > rMax then rNext = 1

 If rNow < rMax then rNow = rNow + 1

 xsave[rNext] = x'

 Esave(rNext) = E0

 Hashsave(rNext) = Hash0

Endif

End of Duplication Check Subroutine

Genetic Operators

Representations, Fitness Functions and Genetic Operators for the Satisfiability Problem

Jens Gottlieb and Nico Voss

Technische Universität Clausthal, Institut für Informatik,
Erzstraße 1, D-38678 Clausthal-Zellerfeld, Germany.
{gottlieb,nvoss}@informatik.tu-clausthal.de

Abstract. Two genetic algorithms for the satisfiability problem (SAT) are presented which mainly differ in the solution representation. We investigate these representations – the classical bit string representation and the path representation – with respect to their performance. We develop fitness functions which transform the traditional fitness landscape of SAT into more distinguishable ones. Furthermore, new genetic operators (mutation and crossover) are introduced. These genetic operators incorporate problem specific knowledge and thus, lead to increased performance in comparison to standard operators.

1 Introduction

The satisfiability problem (SAT) is the first problem proved to be NP-complete [GJ79] and can be stated as follows. Given a boolean function $f : \mathbb{B}^n \rightarrow \mathbb{B} = \{0, 1\}$, the question is: Does there exist a variable assignment $x = (x_1, \dots, x_n) \in \mathbb{B}^n$ with $f(x) = 1$?

In this paper we assume without loss of generality that f has conjunctive normal form $f = c_1 \wedge \dots \wedge c_m$ with each clause c_i being a disjunction of k_i literals, i.e. (positive) variables and negated variables. Furthermore, we suppose $k_i = k$ for all $i \in \{1, \dots, m\}$ and for a constant k .¹ Our goal is to find a variable assignment $x \in \mathbb{B}^n$ that satisfies all clauses.

It is challenging to develop heuristic methods for this problem because unless $P = NP$ there does not exist an exact algorithm with polynomial time complexity for SAT. As there is a growing interest in evolutionary computation techniques in the last years, it seems to be natural that many researchers apply genetic algorithms (GAs) to SAT and 3-SAT [DJS89, Fra94, Hao95, Par95, FF96, EH97].

We investigate two different solution representations of SAT, one of them being the classical bit string representation. The other representation is the path representation which emphasizes the satisfaction of clauses. For each representation we introduce new fitness functions which contain more heuristic information than the traditional approaches. Moreover, we propose problem specific mutation and crossover operators.

¹ Note that even for $k = 3$ (3-SAT) the problem is NP-hard, while all instances with $k = 2$ (2-SAT) are solvable in polynomial time [GJ79].

This paper is organized as follows. After a review of related work in Sect. 2, Sect. 3 introduces the ingredients (fitness function, mutation and crossover) of the genetic algorithm based on the bit string representation. Also, some computational results are presented. Section 4 deals with the path representation, its fitness function and genetic operators, and presents the obtained results. Finally, the conclusions and possible directions for future research are given in Sect. 5.

2 Related Work

De Jong and Spears [DJS89] are the first who have applied GAs to solve SAT. They use the bit string representation with two-point crossover and standard mutation, and do not assume f to be in conjunctive normal form. Thus, they introduce a fitness function with range $[0, 1]$ that recursively evaluates a given expression.

Frank [Fra94] reports that the use of hillclimbing before the other genetic operators significantly enhances the solution quality for 3-SAT problems. Furthermore, a specific parallel GA with interacting sub-populations seems to be inferior to a “normal” GA with only one population.

Hao [Hao95] proposes a (non-standard) representation that emphasizes the local effects of the variables in the clauses and that is strongly related to our path representation. Each clause is assigned a partial variable assignment that satisfies this clause; all these assignments form the chromosome. Some of these “local” variable assignments may be inconsistent and thus, the goal is to find a chromosome without any inconsistencies. Hao employs fitness functions that guide the GA search into regions with only few inconsistencies. He presents a special bit mutation that ensures local consistency, and local search operators.

Park [Par95] checks the effect of crossover and mutation in GAs with bit string representation for 3-SAT. He assumes a conjunctive normal form and uses the number of satisfied clauses as fitness function. He reports similar performance of uniform and two-point crossover, but comes to the conclusion that a GA with standard mutation alone is more effective than a GA including crossover.

Another bit string based approach is proposed by Fleurent and Ferland [FF96]. They use standard mutation and a heuristic crossover operator based on uniform crossover that exploits information about clauses that are not satisfied by both parents. Our crossover operators pursue a similar idea to make use of problem specific knowledge. Fleurent and Ferland report good results for GAs incorporating local optimization.

Eiben and van der Hauw [EH97] apply adaptive GAs based on the bit string representation to 3-SAT. They use standard mutation and investigate multi-parent operators but conclude that a GA using a population of size one (and thus, no crossover) yields sufficient good results. They employ an adaptive penalty function to guide the search towards solutions satisfying yet unsatisfied clauses. Their approach is noteworthy for its generality as it is principally applicable to any constrained problem, e.g. the graph coloring problem [EH96].

Besides the GA approaches many local search algorithms can be found in literature. Selman et al. [SLM92] report good results for their GSAT procedure. Many enhancements of this algorithm are proposed with respect to escaping from local optima [SKC94] or heuristic weights of clauses [Fra96, Fra97]. Gu [Gu94] gives an overview of other optimization algorithms for SAT. The most prominent exact algorithm originates from a method proposed by Davis and Putnam [DP60].

3 The Bit String Representation

3.1 Representation and Basic Fitness Function

The most obvious way to represent a solution of SAT is a bit string of length n where every variable x_j corresponds with one bit. As genetic algorithms use the fitness function to guide the search into promising regions of the search space it is very important to design a well suited fitness function. Obviously, the simplest approach is to take the given boolean function f as fitness function. The main drawback of this approach is that unless the GA has found a solution all individuals would have the same fitness value 0. Hence, the GA gets no information from f and consequently, the GA search degenerates to pure random search which is very ineffective.

In what direction of the search space should we guide the GA search? We must define a fitness function that is able to distinguish individuals x with $f(x) = 0$. Such a function should have higher values if the distance to an optimum is getting lower. Hence, it is reasonable to use $fit_B(x) = \text{"number of satisfied clauses"}$ as *basic fitness function*, see [Par95, EH97]. Note that an individual x with maximum fitness value solves our SAT instance, and that the range of fit_B is $\{0, \dots, m\}$, where m is the number of clauses of f . We interpret this problem formulation as a maximization problem, i.e. the goal is to find a solution with maximum objective (fitness) value. Hence, we use the word solution for an individual although it need not be a solution of the original SAT instance.

3.2 Refining Functions

The fitness landscape induced by the basic fitness function consists of at most $m + 1$ different heights. Due to the large search space there are many solutions having the same fitness value. Thus, any GA using this fitness function is not able to distinguish between such solutions. This would be acceptable if all these solutions have the same chance to lead to improved solutions or even the global optimum. But this need not be the case for most instances of SAT.

We define *refining functions* with range $[0, 1)$ which capture heuristic information about the solution quality. These functions are added to the basic fitness function yielding a fitness function with range $[0, m + 1)$. The reason for the restricted range $[0, 1)$ of the refining functions is that the main distinction between two individuals should remain the number of satisfied clauses. Hence, the

heuristic information contained in the refining function becomes most influential if some individuals have the same value with respect to fit_B .

Let p_j and n_j be the numbers of positive and negative occurrences of variable x_j in f , respectively. If $p_j > n_j$ holds it is a reasonable rule of thumb to set $x_j = 1$: This could cause a higher number of satisfied clauses and more alternatives for other variables in the clauses satisfied by x_j . Together with the corresponding argument for the case $p_j < n_j$, this idea leads to the first refining function

$$ref_{B1}(x) = \frac{1}{n} \sum_{j=1}^n \frac{x_j p_j + (1 - x_j) n_j}{1 + p_j + n_j}$$

which puts emphasis on satisfied clauses. The second refining function

$$ref_{B2}(x) = \frac{1}{n+1} \sum_{j=1}^n \frac{1}{(1 - x_j) p_j + x_j n_j + 1}$$

is based on the same idea, but emphasizes the number of unsatisfied clauses. Thus, ref_{B2} has increasing values for a decreasing number of unsatisfied clauses for a given variable. The third refining function

$$ref_{B3}(x) = \frac{1}{n} \sum_{j=1}^n \frac{x_j p_j + (1 - x_j) n_j}{(1 - x_j) p_j + x_j n_j + p_j + n_j + 1}$$

is simply a combination of ref_{B1} and ref_{B2} .

Whereas the first three refining functions are based on information about the variables in the conjunctive normal form, the fourth refining function uses information on the clauses in f . Therefore, let $s_i(x) \in \{0, \dots, k\}$ be the number of variables x_j that satisfy c_i . The higher $s_i(x)$ for an assignment x , the more variables contained in c_i may be changed without violating the constraint that x must satisfy c_i . As these changes could result in a higher number of satisfied constraints, we define the last refining function as

$$ref_{B4}(x) = \frac{1}{k(m+1)} \sum_{i=1}^m s_i(x) .$$

Adding one of these refining functions to the basic fitness function changes some plateaus in the fitness landscape into a more distinguishable landscape containing additional small hills. This should give more information to the GA to direct the search into promising regions of such plateaus and to leave such plateaus towards regions of higher quality. It is important to use the refining function together with the basic fitness function, as was confirmed by experiments. Otherwise the main information of any solution (the number of satisfied clauses) is lost and the search would probably be guided into regions of suboptimal solutions.

3.3 Mutation and Crossover

The standard mutation for bit strings (which changes each bit with the same probability) may destroy good parts of the chromosome and is not able to guide the search into regions of higher quality. Hence, we propose a mutation operator that tries to improve a solution while preserving its promising parts.

Our operator M_B changes only bits corresponding to variables which are contained in at least one unsatisfied clause. M_B checks all unsatisfied clauses in random order. The contained variables are changed with probability $p_M \cdot \text{fit}_B(a)/m$ which helps avoiding premature convergence as the mutation probability is increased if the distance of the solution a to the optimum gets lower. Note that every bit is changed at most once by one application of M_B . This operator is motivated by the observation that in the current solution at least one of the variables in each unsatisfied clause must be changed to obtain an optimal solution. The WSAT procedure of Selman et al. is based on the same idea [SKC94].

```

procedure MUTATION  $M_B$  (parent  $a \in \mathbb{B}^n$ )
  set  $\bar{a} = a$ 
  for  $c_i \in \{c_1, \dots, c_m\}$  do    {random order}
    if clause  $c_i$  is not satisfied by  $a$  then
      for variables  $x_j$  contained in clause  $c_i$  do
        if  $\bar{a}_j$  is unchanged and  $\text{random} < p_M \cdot \text{fit}_B(a)/m$  then
          set  $\bar{a}_j = 1 - \bar{a}_j$  and mark  $\bar{a}_j$  as changed
  return child  $\bar{a}$ 

```

Fig. 1. Pseudocode for mutation operator M_B

Due to preliminary experiments with traditional crossover operators for bit strings (uniform, one-point, two-point and some variants of n -point crossover) we find no crossover being able to produce acceptable results. Hence, we have designed other crossover operators and now present C_B , the most successful one which incorporates knowledge of the structure of SAT.

The procedure C_B duplicates the parents (yielding children $\bar{a}$ and $\bar{b}$) and checks the clauses in random order. For each clause c_i all contained literals are sequentially tested whether the corresponding variable assignments of $\bar{b}$ make these literals satisfy c_i . If they do so, these variable assignments $\bar{b}_j$ are copied to $\bar{a}_j$ with probability $r \in (0, 1]$. Otherwise the corresponding values of $\bar{a}$ are copied to $\bar{b}$ with probability r .² This crossover operator transfers good parts between

² This is motivated as follows: If $\bar{a}_j \neq \bar{b}_j$ and $\bar{b}_j$ does not satisfy c_i , then $\bar{a}_j$ must satisfy c_i . Furthermore, note that (i) the assignments $\bar{a}_j$ and $\bar{b}_j$ remain unchanged if $\bar{a}_j = \bar{b}_j$, (ii) each bit can be changed at most once during a single crossover operation (one flip yields $\bar{a}_j = \bar{b}_j$ and makes more flips impossible), and (iii) r is a specific parameter of C_B that must not be confused with the crossover probability p_C in the GA.

the parents, which causes the children to have a smaller Hamming distance than the parents with high probability. This may yield a loss of diversity in the current population and thus, illustrates the high importance of a mutation operator that helps escape from local optima, a condition that is satisfied by M_B .

```

procedure CROSSOVER  $C_B$  (parents  $a, b \in \mathbb{B}^n$ )
  set  $\bar{a} = a$  and  $\bar{b} = b$ 
  for  $c_i \in \{c_1, \dots, c_m\}$  do    {random order}
    for variables  $x_j$  contained in clause  $c_i$  do
      if  $\text{random} < r$  then
        if  $\bar{b}_j$  satisfies  $c_i$ 
          then set  $\bar{a}_j = \bar{b}_j$ 
          else set  $\bar{b}_j = \bar{a}_j$ 
  return children  $\bar{a}$  and  $\bar{b}$ 

```

Fig. 2. Pseudocode for crossover operator C_B

3.4 Computational Results

Mitchell et al. [MSL92] report the hardest satisfiable instances of 3-SAT having the ratio $m/n = 4.3$. Thus, we restrict our investigations to these instances. Each instance is produced by randomly generating m clauses. The literals of each clause are generated by randomly choosing variables and negating them with probability 0.5. Furthermore, we ensure satisfiability of the instance by restricting the alternatives for the generation of one literal in each clause.³

We use a generational GA with elitism and population size 12 (this yields good results with respect to the total number of evaluations). The initial population P_0 is generated randomly. Population P_{i+1} is computed by tournament selection (tournament size 2) from P_i , mating some pairs via crossover with probability $p_C = 0.7$ and mutating all individuals using the parameter $p_M = 0.3$. The additional crossover parameter is $r = 0.65$. To ensure elitism the best individual of P_i is inserted into P_{i+1} .

Our initial experiments show that classical GA operators are more time-consuming and that they do not find any solution for SAT in many cases. Figure 3 shows the typical behaviour of standard mutation and M_B (in combination with some crossover operators) for one specific representative large instance.

A GA using bit mutation⁴ is not able to locate the optimum, while M_B makes the GA solve the instance very quickly for all crossovers. In general, bit mutation

³ We do not employ a standard generator for SAT instances. However, we observe that the hardest instances produced by our generator are found at $m/n \approx 4.3$, too.

⁴ As a high mutation rate causes disruptive behaviour, we use a lower mutation probability for bit mutation. The probability is chosen to be 0.01 which seems to exhibit the best performance.

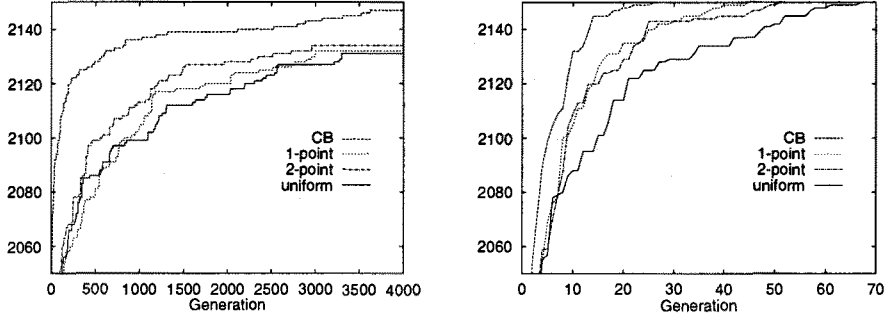


Fig. 3. Fitness progress of standard bit mutation (left) and M_B (right) for different crossover operators and one instance with $m = 2150$, $n = 500$ and $k = 3$

sometimes succeeds for smaller instances. However, it is clearly dominated by M_B which is faster and more robust in finding a solution. The crossover operator C_B speeds up the search and yields higher fitness values than the considered classical crossovers. One-point and two-point crossover seem to perform better than uniform crossover for this instance, but in general these crossovers show a very similar behaviour. To sum it up, M_B and C_B form the best observed combination of mutation and crossover operators.

Let us now compare the effect of the refining functions. For each number of variables we generate 10 instances and give each fitness function 10 runs. The numbers of generations needed to find a satisfying assignment are averaged over instances and runs, and are given in Table 1.

Table 1. Results for the bit string representation (population size 12)

m	n	fit_B	$fit_B + ref_{B1}$	$fit_B + ref_{B2}$	$fit_B + ref_{B3}$	$fit_B + ref_{B4}$
430	100	13.24	12.42	12.27	12.62	15.28
1075	250	23.88	21.10	20.31	19.72	20.01
2150	500	33.82	33.24	33.09	32.47	32.02
4300	1000	43.17	42.00	41.56	45.46	40.71
8600	2000	60.31	66.30	64.15	61.28	72.62

The five functions yield very similar performance. In many cases adding the refining function to fit_B yields small improvements, but there are also some cases where fit_B outperforms the refined fitness functions. For $n = 2000$, fit_B is superior to all refined fitness functions. One might expect an increasing influence of the refining functions for higher population sizes. This cannot be confirmed by experiments with population size 80: The differences between the five fitness

functions are even smaller than in Table 1.

We conclude that the use of refining functions not necessarily leads to improved performance when using the bit string representation. A reason for this might be the fact that the basic fitness function gives enough information to the GA. As it seems that the approach of static refining functions is not appropriate, one might guess that more elaborated and flexible refining functions could yield better results. More flexibility could be gained by dynamic or adaptive refining functions which opens a new direction for further research.

However, we observe good performance for all fitness functions. This mainly depends on the genetic operators which are very robust (as a solution has been found in all cases). The operators M_B and C_B make the GA solve even instances with $n = 2000$ within less than 75 generations, i.e. less than 900 evaluations are needed. Furthermore, the number of needed evaluations grows sublinear for increasing n . This should make the algorithm suitable for even larger instances.

4 The Path Representation

4.1 Representation and Basic Fitness Function

Any variable assignment solving a SAT instance must satisfy at least one literal in each clause. Thus, we may select one literal per clause that should be satisfied by a variable assignment. We call a sequence $p = (p_1, \dots, p_m)$ of indices $p_i \in \{1, \dots, k\}$, which select one literal within each clause, a *path*. In a path, two indices corresponding to the literals $\neg x_l$ and x_l cause an *inconsistency*, because there does not exist a variable assignment that satisfies both literals. A *feasible path* is a path without inconsistencies. Feasible paths can be used to construct a variable assignment for the SAT instance.

To clarify these concepts by an example, we consider the SAT instance with $m = 4$, $n = 4$, $k = 3$ and the boolean formula

$$(x_1 \vee \neg x_3 \vee \neg x_4) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_4)$$

which is illustrated in Fig. 4. The path $p = (1, 4, 2, 4)$ (which selects the literals $x_1, x_4, \neg x_2, x_4$ in the clauses c_1, c_2, c_3, c_4 , respectively) is shown by the lines between the clauses (depicted by ovals). This path p is feasible and hence, we can construct the feasible variable assignment $x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 1$ from p (as no literal with variable x_3 is selected by p , we may choose an arbitrary boolean value for x_3). On the other hand, the path $p' = (4, 4, 1, 1)$ contains two inconsistencies which prevent the construction of a feasible variable assignment from p' . Thus, p' is not feasible.

In general, we should search for a path with a minimal number of inconsistencies. If even the path with the least number of inconsistencies contains an inconsistency, then the SAT instance is not satisfiable. Otherwise, if there exists at least one feasible path, then there exists at least one feasible variable assignment. Paths containing lots of inconsistencies are judged as “bad” because many changes are necessary to obtain a path with minimal number of

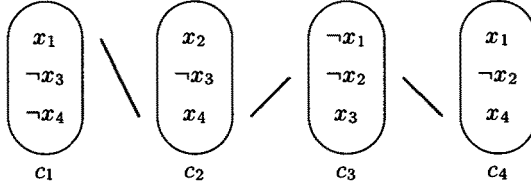


Fig. 4. An example for a path in a boolean formula in conjunctive normal form

inconsistencies. On the other hand “good” paths contain only few inconsistencies. Hence, it is convenient to define our basic fitness function to be maximized as $fit_P(p) = “w - \text{number of inconsistencies in } p”$ for a sufficient high constant $w \in \mathbb{N}$, which ensures positive fitness values.

For each path p , we can determine the set of variable assignments that can be constructed from p . This set contains 2^t ($t \geq 0$) solutions if p is feasible, and is empty otherwise. Here, t denotes the number of variables that are not selected by the path. A comparison with the bit string representation leads to another observation. The size of the path search space is k^m and therefore independent of the number n of variables – in contrast to the size 2^n of the bit string space which is independent of k and m . Hence, it could be justifiable to prefer the path representation for growing n and decreasing k and m , and bit strings for increasing k and m and decreasing n .

4.2 The Refining Functions

Like for the bit string representation it is possible to define refining functions that make the fitness landscape more distinguishable. We use basically the same idea as for fit_{B1} , fit_{B2} and fit_{B3} and thus, we only briefly sketch the refining functions for the path representation. Given a path p and a literal in p , we count the number of occurrences of this literal in f . The ratio of this number and the number of total occurrences of the corresponding variable (positive and negative) is computed. $ref_{P1}(p)$ is the normalized sum of the ratios for all literals in p . The normalization ensures that the basic fitness function remains the dominating quality criterion in $fit_P + ref_{P1}$. Note that ref_{P1} can be seen as a path representation equivalent to ref_{B1} . The other refining functions ref_{P2} and ref_{P3} are calculated in a similar fashion and correspond directly to ref_{B2} and ref_{B3} , respectively.

4.3 Mutation and Crossover

The simplest possible mutation operator just makes random changes of p_i for some $i \in \{1, \dots, m\}$ such that $p = (p_1, \dots, p_m)$ remains a path.⁵ Experimental

⁵ Note that not every random change yields a path: Suppose a change of p_i to a variable that is not contained in the corresponding clause c_i .

results indicate the poor performance of this operator which we call random change mutation. Hence, we propose a mutation that works more goal-oriented. Our operator M_P checks all clauses and selects for each clause c_i a subset L of the contained literals, where each literal is selected with probability p_M . The procedure determines one literal $j \in L$ which causes the least number of inconsistencies in p . If this literal j causes less inconsistencies than p_i in p , the child gets the value $\bar{p}_i = j$. It should be remarked that the traversing order of the clauses has no effect on the resulting child, therefore M_P checks the clauses in sequential order.

```

procedure MUTATION  $M_P$  (parent path  $p$ )
  set  $\bar{p} = p$ 
  for  $i = 1$  to  $m$  do
    set  $L = \emptyset$ 
    for variables  $x_j$  contained in clause  $c_i$  do
      if  $\text{random} < p_M$  then set  $L = L \cup \{j\}$ 
    let  $j$  be a literal from  $L$  that causes the least number of inconsistencies in  $p$ 
    if literal  $j$  causes less inconsistencies than  $p_i$  in  $p$ 
      then set  $\bar{p}_i = j$ 
  return child  $\bar{p}$ 

```

Fig. 5. Pseudocode for mutation operator M_P

We have adapted the ideas of the bit string crossover C_B to the path representation. The resulting operator C_P checks each clause c_i with probability $r \in (0, 1]$ and transfers information between the parents p and q as follows. If p_i causes less inconsistencies in parent p than q_i in parent q , then the value p_i is copied to child $\bar{q}$, i.e. $\bar{q}_i = p_i$. In the other case the value q_i is transferred to $\bar{p}$, i.e. $\bar{p}_i = q_i$. Above remark about the traversing order of the clauses in M_P is valid for C_P , too. It is obvious that the produced children have more similarities than the parents. Hence, the same observation as for the bit string representation can be made – M_P is able to alter these children significantly which helps overcome premature convergence.

4.4 Computational Results

We use the same GA with some changed parameters which we found suitable for the path representation (population size 20, $p_C = 0.7$, $r = 0.65$).⁶ The GA is stopped when a generation limit of 4000 is reached. First, we compare our operators M_P and C_P with random change mutation and classical crossovers, respectively. The results for one representative instance are shown in Fig. 7. Crossover

⁶ The probabilities for random change mutation are selected as 0.02 and 0.01 for C_B and the classical crossovers, respectively. The corresponding probabilities for M_P are chosen as 0.5 and 0.3.

```

procedure CROSSOVER  $C_P$  (parents  $p, q$ )
  set  $\bar{p} = p$  and  $\bar{q} = q$ 
  for  $i = 1$  to  $m$  do
    if  $\text{random} < r$  then
      if  $p_i$  causes less inconsistencies in  $p$  than  $q_i$  in  $q$ 
        then set  $\bar{q}_i = p_i$ 
        else set  $\bar{p}_i = q_i$ 
  return children  $\bar{p}$  and  $\bar{q}$ 

```

Fig. 6. Pseudocode for crossover operator C_P

C_P yields a faster increase of fitness for both mutation operators. Nevertheless all crossovers fail within 4000 generations when random change mutation is used. Tests with a higher generation limit show that random change mutation finds a solution in only very few cases. Independent of the used crossover, the time needed to obtain a solution is unpredictable and can be arbitrary large. M_P shows a better behaviour in contrast to random change mutation, especially in combination with C_P (a solution is found very quickly, see Fig. 7). While the classical crossovers sometimes fail (1-point and uniform crossover do not find a solution in 4000 generations for the considered instance), C_P makes the use of M_P more effective.

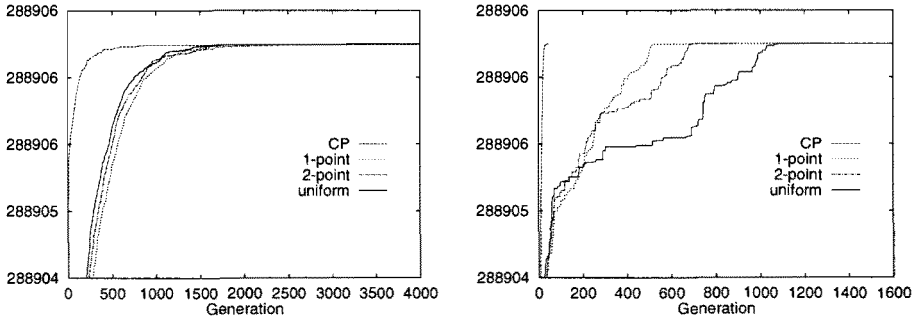


Fig. 7. Fitness progress of random change mutation (left) and M_P (right) for different crossover operators and one instance with $m = 1075$, $n = 250$ and $k = 3$

We test the fitness functions for the path representation on the same test set as the bit string representation. The results are presented in Table 2. It is important to remark that in ca. 2 % of all runs the GA fails to find a solution (the failure probability increases for larger instances). These runs are contained (with generation limit 4000) in the averages given in Table 2.

There is no clearly dominant refining function. In many cases the refined

Table 2. Results for the path representation (population size 20)

m	n	fit_P	$fit_P + ref_{P1}$	$fit_P + ref_{P2}$	$fit_P + ref_{P3}$
430	100	48.45	43.78	43.34	46.14
1075	250	114.24	71.57	118.30	209.75
2150	500	264.25	266.72	211.75	186.21
4300	1000	296.52	242.27	203.25	207.17
8600	2000	547.05	361.77	461.86	497.11

functions show better behaviour than fit_P – but not in all cases. However, for large instances the basis fitness function fit_P is inferior.

For the results with population size 80 see Table 3 (in 0.5 % of all runs the GA fails to find a solution). We observe that an increase of the population size makes the refined fitness functions dominate fit_P . The effect is getting higher for larger instances. Thus, for the path representation it makes sense to use refining functions. One reason for this might be that the basic fitness function induces a not sufficient continuous fitness landscape. Another reason could be a general difficulty to leave local optima in this search space. Hence, the GA needs additional information to avoid getting trapped in local optima. The danger of being misled by fit_P can be diminished by the refining functions.

Table 3. Results for the path representation (population size 80)

m	n	fit_P	$fit_P + ref_{P1}$	$fit_P + ref_{P2}$	$fit_P + ref_{P3}$
430	100	21.31	20.37	22.32	21.29
1075	250	62.81	28.76	29.43	29.61
2150	500	152.45	118.23	74.86	45.24
4300	1000	76.92	51.76	45.62	45.36
8600	2000	244.15	65.91	80.14	96.25

A comparison with the bit string based GA shows the clear inferiority of the path representation. Even instances with $n = 100$ need about 50 generations of size 20 (i.e. 1000 evaluations) – that is more than the bit string GA needs for instances with $n = 2000$. As some instances are not solved, we may conclude that the path representation is much less robust than the bit string representation.

5 Conclusions

We have presented two representation schemes for SAT, together with refined fitness functions and new problem specific genetic operators. One part of our

investigation aims at the incorporation of additional heuristic information into the fitness function. The bit string GA seems to be insensitive to these refined fitness functions. On the other hand, the performance of the GA for the path representation is improved by refining functions. This effect increases with higher population sizes and larger instances. It might be interesting to design other refining functions: As the presented functions are static, it could be worthwhile to investigate dynamic or adaptive refining functions. Perhaps these more flexible types of refining functions could even improve the performance for the bit string representation.

Besides the concept of refined fitness functions, new genetic operators form the second aspect of our study. The obtained results are strongly influenced by our problem specific operators. This can easily be verified by the comparison with standard operators (e.g. bit mutation, 1-point, 2-point and uniform crossover) which exhibit a clearly inferior behaviour. Thus, it could be promising to further improve our operators. One way to achieve this could be the use of even more problem specific knowledge, or the incorporation of local optimization capabilities. It should be noticed that our crossover and mutation operators can easily be adapted to other constraint satisfaction problems. It could be worthwhile to investigate this in a later study, as their success for these problems might give more insight into possible improvements.

The third aspect of this study is the comparison between the bit string and the path representation for SAT problems. There is a clear winner in this competition of the two representations: the bit string representation. Probably, the reason is that it is the most natural representation for SAT which enables us to use an accurate fitness function. From the view of GA constraint handling techniques (see [Mic96] for a survey) the path representation has similarities with the decoder approach. Such approaches often suffer from a lack of continuity in the search space, which may explain the need for additional information (e.g. given by a refining function).

To sum it up, the results indicate that despite the use of refining functions the path representation is dominated by the most natural representation, the bit string representation. However, it must not be neglected that a representation can only be successful if there are good genetic operators available.

References

- [DP60] M. Davis and H. Putnam. A Computing Procedure for Quantification Theory. *Journal of the ACM*, Volume 7, 201 – 215, 1960
- [DJS89] K. A. De Jong and W. M. Spears. Using Genetic Algorithms to Solve NP-Complete Problems. In J. D. Schaffer (ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, 124 – 132, Morgan Kaufmann Publishers, San Mateo, CA, 1989
- [EH96] A. E. Eiben and J. K. van der Hauw. Graph Coloring with Adaptive Genetic Algorithms. Technical Report 96-11, Department of Computer Science, Leiden University, 1996

- [EH97] A. E. Eiben and J. K. van der Hauw. Solving 3-SAT with Adaptive Genetic Algorithms. In *Proceedings of the 4th IEEE Conference on Evolutionary Computation*, 81 – 86, IEEE Service Center, Piscataway, NJ, 1997
- [FF96] C. Fleurent and J. A. Ferland. Object-oriented Implementation of Heuristic Search Methods for Graph Coloring, Maximum Clique and Satisfiability. In D. S. Johnson and M. A. Trick (eds.), *Cliques, Coloring and Satisfiability: 2nd DIMACS Implementation Challenge*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Volume 26, 619 – 652, 1996
- [Fra94] J. Frank. A Study of Genetic Algorithms to Find Approximate Solutions to Hard 3CNF Problems. Golden West International Conference on Artificial Intelligence, 1994
- [Fra96] J. Frank. Weighting for Godot: Learning Heuristics for GSAT. In *Proceedings of the 13th National Conference on Artificial Intelligence and the 8th Innovative Applications of Artificial Intelligence Conference*, 338 – 343, 1996
- [Fra97] J. Frank. Learning Short-Term Weights for GSAT. Submitted to 15th International Joint Conference on Artificial Intelligence, 1997
- [GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco, CA, 1979
- [Gu94] J. Gu. Global Optimization for Satisfiability (SAT) Problem. *IEEE Transactions on Knowledge and Data Engineering*, Volume 6, Number 3, 361 – 381, 1994
- [Hao95] J.-K. Hao. A Clausal Genetic Representation and its Evolutionary Procedures for Satisfiability Problems. In D. W. Pearson, N. C. Steele, and R. F. Albrecht (eds.), *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms*, 289 – 292, Springer, Wien, 1995
- [Mic96] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Third Edition, Springer, 1996
- [MSL92] D. Mitchell, B. Selman, and H. Levesque. Hard and Easy Distributions of SAT Problems. In *Proceedings of the 10th National Conference on Artificial Intelligence*, 459 – 465, 1992
- [Par95] K. Park. A Comparative Study of Genetic Search. In L. J. Eshelman (ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms*, 512 – 519, Morgan Kaufmann, San Mateo, CA, 1995
- [SKC94] B. Selman, H. A. Kautz, and B. Cohen. Noise Strategies for Improving Local Search. In *Proceedings of the 12th National Conference on Artificial Intelligence*, 337 – 343, 1994
- [SLM92] B. Selman, H. Levesque, and D. Mitchell. A New Method for Solving Hard Satisfiability Problems. In *Proceedings of the 10th National Conference on Artificial Intelligence*, 440 – 446, 1992

Genetic Algorithms at the Edge of a Dream

Cathy Escazut¹ and Philippe Collard²

¹ Evolutionary Computation Research Group,
Napier University
219 Colinton Road,
Edinburgh, EH14 1DJ — SCOTLAND
e-mail: cathy@dcs.napier.ac.uk
or escazut@unice.fr

² Laboratory I3S — CNRS-UNSA
250 Av. A. Einstein,
Sophia Antipolis, 06560 Valbonne — FRANCE
e-mail: pc@unice.fr

Abstract. This paper describes a *dreamy genetic algorithm* scheme, emulating one basic mechanism of chronobiology: the alternation of awake and sleeping phases. We use the metaphor of the *REM sleep* during which the system is widely disconnected from its environment. The dream phase allows the population to reorganize and maintain a needed diversity. Experiments show that dreamy genetic algorithms improve on standard genetic algorithm, for both stationary (deceptive) and non-stationary optimization problems.

A theoretical and experimental analysis suggests that dreamy genetic algorithms are better suited to complex tasks than standard genetic algorithms, due to the preservation of the population diversity.

1 Introduction

We propose to implement within Genetic Algorithms (GAs), one of the basic mechanisms of the chronobiology: the *REM sleep*. Neurobiologists established that states of sleep in human beings are the achievement of a long evolution [13]. Taking our inspiration from this work, we dictate *circadian* cycles to the genetic population with an external control. Such an approach has not been yet explored in an artificial context, on the one hand because only a few evolutionary theories are interested in the problem of the function of the dream, and on the other hand because the metaphorical basis of GAs is extremely simplified in comparison with the natural model.

Dreamy-GAs are implemented within the framework of *dual-GAs* presented in [3, 5, 4, 6]; more precisely, they alternate awake and dream phases. The awake phase corresponds to the standard mode of dual-GA; the dream phase allows the population to reorganize in order to maintain its diversity.

We first briefly present dual-GAs and the effects of the duality on crossover and mutation operators. Then, we introduce *dreamy-GAs*, their implementation

and analyze their effects. Finally, we present some experimental results on deceptive functions and non-stationary environments.

2 Dual genetic algorithms

The problem of premature convergence is usually addressed by explicitly enforcing the population diversity via *ad hoc* operators or selection procedures [10, 14, 15]. This contrasts with dual-GAs keeping standard operators and selection procedures: dual-GAs rather enforce diversity at the representation level.

2.1 Basic principles

The form of dual Genetic Algorithms is the same as the one of standard GAs. They only differ in the representation of the individuals. Within dual-GAs, individuals are enhanced with an additional bit, termed *head bit*, that controls the interpretation of the individual and the computation of its fitness. More precisely, individual $\underline{0}\omega$ is intended as ω , while individual $\underline{1}\omega$ is intended as $\bar{\omega}$, the complementary of ω . In the new genotypic space Ω^* given as $\{0, 1\} \times \Omega$, where Ω is the standard genotypic space, we distinguish *chromosomes* (elements of 0Ω) and *anti-chromosomes* (elements of 1Ω). For instance, *dual individuals* $\underline{0}0100$ and $\underline{1}1011$ (complementary to each other) both represent the single individual 0100: they form a *dual pair*.

This increased search space allows an improved use of the schemas (see [5] for more details). Let us now focus on another characteristic induced by dual-GAs: *implicit mutations*.

2.2 Implicit mutations

Within conventional GAs the mutation rate is mostly handled as a global external parameter which is constant over time [19]. We showed in [4], that using a dual-GA with only the crossover operator, mutations are *implicit*. Indeed, within dual-GAs, crossing over two dual individuals 0ω and $1\bar{\omega}$ actually amounts to mutating ω . It is well known that standard crossover can be viewed as a biased mutation [7, 12, 18], with the advantage that the corresponding mutation rate is automatically controlled from the diversity of the population; but inversely, crossover does not allow for restoring a population diversity.

This limitation of standard crossover disappears in dual-GAs, provided that there exists dual pairs (0ω and $1\bar{\omega}$) in the population. For instance, let us consider the individual 101 represented by the dual pair ($\underline{0}101, \underline{1}010$). If a crossover applied on locus 2 to the pair is followed by a crossover applied on locus 3 of obtained offspring, i.e. $\underline{0}110$ and $\underline{1}001$, we have the two individuals $\underline{0}111$ and $\underline{1}000$ representing the individual 111. These two consecutive dual crossovers have the same effect than an *explicit* mutation applied to the individual 101 on the second locus.

The implicit mutation rate due to the crossover effects therefore depends on the number of dual pairs in the population. The reason why the number of dual pairs decreases as the dual-GA converges has been examined in [4]. It follows that the rate of implicit mutations decreases from an initial value to zero. This ensures that the dual-GA achieves some tradeoff between the exploration and exploitation task: it works on the borderline of efficient optimization and almost random walk.

The *Implicit Mutation Rate* is defined as follows:

$$Imr = p_c \cdot \sum_{\omega \in \Omega} \min(P(\underline{0} \omega), P(\underline{1} \bar{\omega}))$$

where p_c is the crossover probability, $P(\xi)$ the proportion of the individual ξ in the population.

Populations having for each chromosome, the corresponding counterpart in the same proportion ($P(\underline{0} \omega) = P(\underline{1} \bar{\omega})$) are called *mirror* populations. We note that such populations have an optimal implicit mutation rate ($Imr = 0.5 \times p_c$).

2.3 The mirroring operator

The mirroring operator transforms any individual in its complementary: the mirror image of $\underline{1} 0100$ is $\underline{0} 1011$. It applies on each individual in the population with a small probability (usually around .01). Thereby, it introduces genotypic diversity while preserving the phenotypic distribution of the population. Used together with crossover operator, it expectedly allows for a new dynamic equilibrium, balancing genotypic diversity and phenotypic convergence.

The mirroring induces some interesting properties. Here are the most important ones:

1. A mirror population is invariant under the mirroring operator.
2. The space of the mirroring operators is *closed* under composition¹.
3. Repeated uses of the mirroring operator alone gives a mirror population.

Let us now present why and how we defined *Dreamy-GAs* from the basis of dual-GAs.

3 Dreamy genetic algorithms

Our goal is to design a GA with persistent diversity. Taking a clue from the basic mechanism of chronobiology, we now introduce periodic dynamics in dual-GAs: we dictate circadian cycles (*autogenous rhythm*) to the genetic population. Awake and dream phases are then integrated to the course of dual-GAs.

¹ The composition of two mirroring operators applied with a rate respectively sets to τ_1 and τ_2 , is the mirroring operator whose rate is $\tau_1 + \tau_2 - 2\tau_1\tau_2$.

3.1 The dream phase

The alternation of awake and dream phases should allow the dreamy-GA to converge, while keeping its ability to explore the search space. Although natural awake/dream cycles have to do with individuals, the dream metaphor makes sense for the population as well, considered as a single individual facing a circadian environment. Dreamy-GAs are (loosely) inspired from the *REM sleep*, during which the system is widely disconnected from its environment and there are only a few interactions between both cerebral hemispheres. Here, the spaces of chromosomes and anti-chromosomes 0Ω and 1Ω are taken as analogs of hemispheres: no interaction between those hemispheres is allowed during dream phases.

More precisely, *dual crossovers* (i.e. crossovers combining individuals of both half spaces) are inhibited during the dream phase : a crossover is only allowed between two chromosomes or two anti-chromosomes. Let us note that the space of chromosomes and the space of anti-chromosomes as well are closed under crossover. This approach resembles restricted mating [16] in the sense that it forbids the mating of individuals in different sub-populations. Still, the sub-populations of chromosomes and anti-chromosomes exchange information via the selection procedure, and overall, via the mirroring operator.

3.2 A theoretical analysis of dreamy-GAs

New genetic algorithms schemes usually undergo an experimental validation on a set of well-studied benchmark functions [8]. However, experimental studies suffer from a number of biases, e.g. the different control parameters of the GA [11, 17].

Therefore, we first analyze the dynamics of dreamy-GAs using the Whitley's executable model [23]. It uses an infinite population, no mutation and requires the enumeration of all points in the search space. Assuming proportionate reproduction, one point crossover, and random mating, an equational system can be used for modelling the proportion of the individuals (remind that this analysis is exhaustive) over time. Let us note that it is possible to develop exact equations for problems up to 15 bits in the coding. These equations generalize those given by Goldberg as far as the *Minimal Deceptive Problem* [9] is concerned. This system may be compared to the Vose and Liepins' formalism and their interpretation using matrices [21].

We simulate the behavior of a dual-GA under effects of selection, crossover and mirroring, alternating awake and dream phases. The fitness function we use involves the fully deceptive $F2$ function, defined over 4 bits [22]; the function values for the 16 individuals are:

$F2(0000) = 28$	$F2(1000) = 20$
$F2(0001) = 26$	$F2(1001) = 12$
$F2(0010) = 24$	$F2(1010) = 10$
$F2(0011) = 18$	$F2(1011) = 2$

$F2(0100) = 22$	$F2(1100) = 8$
$F2(0101) = 16$	$F2(1101) = 4$
$F2(0110) = 14$	$F2(1110) = 6$
$F2(0111) = 0$	$F2(1111) = 30$

The fitness function studied here is composed of 2 copies of $F2$. Thus, individuals are composed of 8 bits. The fitness is the sum of the $F2$ fitness of the first 4 bits, and the $F2$ fitness of the last four bits, divided by 2. The optimal individual is 11111111, with fitness 30. In the dual-GA framework, individuals are composed of 9 bits, the two optimal individuals being 0 11111111 and 1 00000000. Two different simulations are performed.

Simple transitions:

Here, the crossover probability is set to 1 and the mirroring rate to .01. Each phase lasts for 250 generations. During an awake phase the population converges toward a polymorphic population containing only one dominant individual: the individual 0 11111111 (see figure 1). The Imr goes to 0. During the dream phase, the population goes to a mirror one in which only one chromosome and its anti-chromosome coexist in the same proportions. This dual pair represents the optimum. This limit population is phenotypically homogeneous while presenting a maximum genotypical diversity.

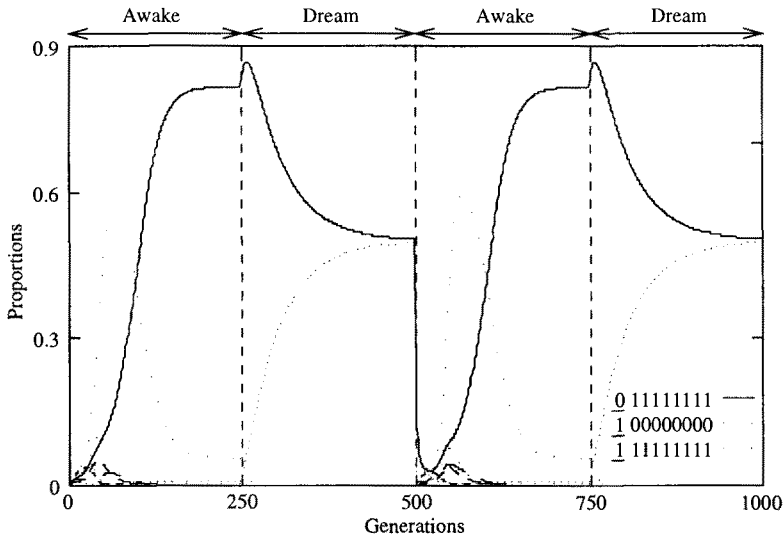


Fig.1. Simple transitions: Evolution of proportions of best individuals.

If an individual comes to dominate the population during an awake phase, the corresponding dual pair appears during the following dream phase, and comes

to dominate the mirror population. We note that during a dream phase the *Imr* goes to its upper bound. In consequence the system maximizes its exploration abilities. This maximization is not performed to the detriment of the exploitation abilities since the dominant individual goes on reproducing. Moreover, during this dream phase if changes in the environment occur, the system becomes less and less sensitive to these changes. Indeed, the population goes to contain only two dual individuals without any possibility of crossover between them; this is in harmony with the metaphor of the *REM sleep*.

Note the smooth transition from awake to dream phases. The inverse transition is indeed abrupt; it reveals a phenomenon of *hysteresis*, and depending on the duration of the dream the transition is not performed on the same population. Such a transition can be interpreted as a *break of symmetry* [2].

Note also that the population in the first generations of an awake phase is fairly close to a uniform distribution (with all individuals being equally represented).

The results of this simulation show that the number of dual pairs dynamically adapts itself during the genetic search: it decreases during the awake phase and increases during the dream phase. So does the ratio exploitation/exploration. A second simulation is concerned with continuous transitions.

Continuous transitions:

In this simulation, dual crossovers apply with a gradually increasing probability p_{dc} , given as:

$$\begin{aligned} \text{IF} \quad & (g < \frac{P}{4}) \\ \text{THEN} \quad & p_{dc} = 1 \\ \text{ELSE} \quad & p_{dc} = \cos\left(\pi \cdot \left(\frac{g}{P} - \frac{1}{4}\right)\right)^2 \end{aligned}$$

where P is the alternation period between phases (1000 in our test), and g is the current generation. Figure 2 shows that as p_{dc} decreases to 0, the *Imr* increases toward its upper bound (about generation 800): the population diversifies. Then it converges toward a polymorphic population dominated by the optimum as p_{dc} goes to 1 (about generation 1000). Finally, with the decrease of p_{dc} , the population converges toward a mirror one containing the optimum individuals.

4 Experimental results

The dreamy-GA is validated on a stationary deceptive problem, and a non-stationary problem. Both are known to be hard to be optimized with a GA.

4.1 A deceptive function

The function used is obtained from 16 copies of the Whitley's *F2* function. Individuals are represented by 64 bits for the standard GA, and 65 bits for the

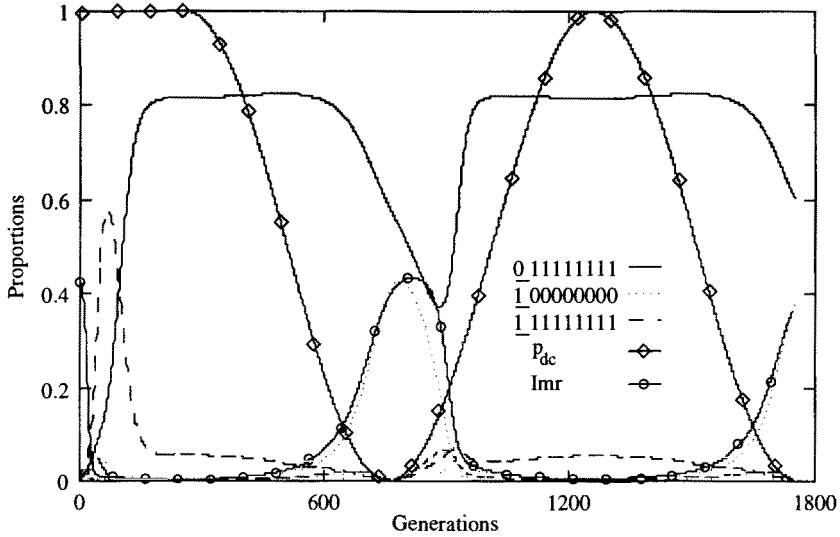


Fig. 2. Continuous transitions: Evolution of proportions of best individuals.

dual and dreamy-GAs. Populations are composed of 100 individuals. The reproduction is controlled by a version of the roulette wheel proportional selection. We use a single-point crossover which probability is set to 1. The mutation rate is set to .9 per chromosome. In the case of the dual and dreamy-GAs, the mirroring rate is fixed to .02. The awake phase lasts for 20 generations; the dream phase, for 5 generations. Results are averaged on 50 independent runs. Figure 3 shows the dynamics of evolution (average best fitness reached for a number of generations). For convenience, only fitnesses between 27 and 30 are plotted.

Note that all three GAs behave the same during the first 200 generations, with a slight advantage for standard GA over dual-GA. About the generation 230, the dreamy-GA breaks away from the two other GAs whose performances are very close. Finally, the dual-GA has the upper hand and tends to come up with the dreamy-GA.

4.2 Non-stationary environments

The tradeoff between exploration and exploitation is more than ever crucial when dealing with non-stationary environments. Dream phases are purposely devised to maintain population diversity (i.e. preserve the exploration ability), while minimizing the phenotypic disruption (loss of fittest individuals). The dynamic environment we used for this test is the *pattern tracking* problem proposed by Vavak and Fogarty [20].

Using this dynamic function, the fitness of an individual is its Hamming distance to a target individual. This target individual is the optimum, and the environment changes as the target individual is modified. The difficulty of change

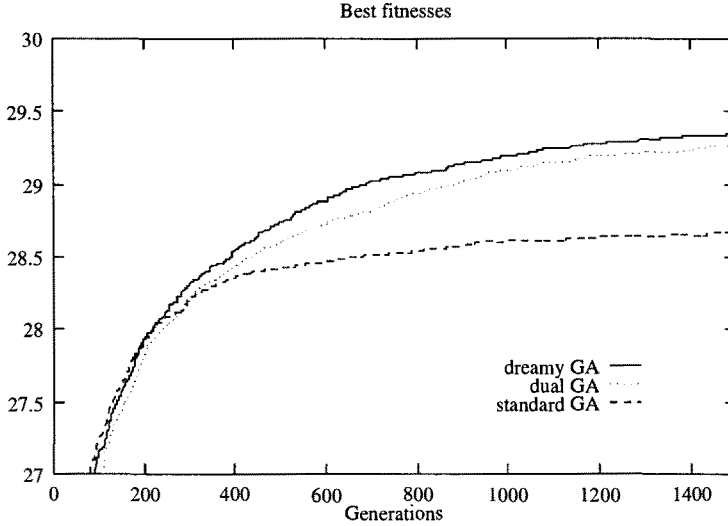


Fig. 3. Fitness evolution for 16 copies of the Whitley's $F2$ function.

relates to the Hamming distance between two consecutive target individuals, which is gradually set to 10, 20, 30 and 40.

The size of the problem is $N = 40$, with population size 50. We use single point crossover with a probability of 1. The mutation rate is .9 for the standard GA, and .2 for the dual and dreamy-GAs; in the latter cases, the mirror rate is .02. The awake time is set to 40 generations and the dream time to 10. Every 50 generations, a change of optimum occurs, so the change is performed while the population has quasi converged. The results are averaged over 50 independent runs. Figure 4 shows the average best fitness reached for a given number of generations.

During the first period (generation 1 to 50), all GAs present similar performances, with a slight advantage for the standard GA. The 10 last generations of this period (generations 40 to 50) correspond to a dream phase. The change that occurs at generation 50 is a small one (the target individuals only differ by ten bits), and all GAs appear equally able to follow the optimum, with a slight advantage for the dreamy-GA. The standard GA appears unable to cope with greater changes (generations 100, 150 and 200): it needs at least 40 generations to catch up. Note that the performance fall is proportional to the magnitude of the change for the standard GA, whereas it is limited to the half size of the problem (20) for the dual-GA (due to the fact that if x and y are very different, then x is close to the dual of y) [6].

These experiments show that dreamy-GAs outperform standard GAs in a non-stationary environment. Indeed a dreamy-GA offers a *natural* way for tracking down changing optima. Its better performance can be explained by the fact that during the dream phase the system tends to maximize its abilities to ex-

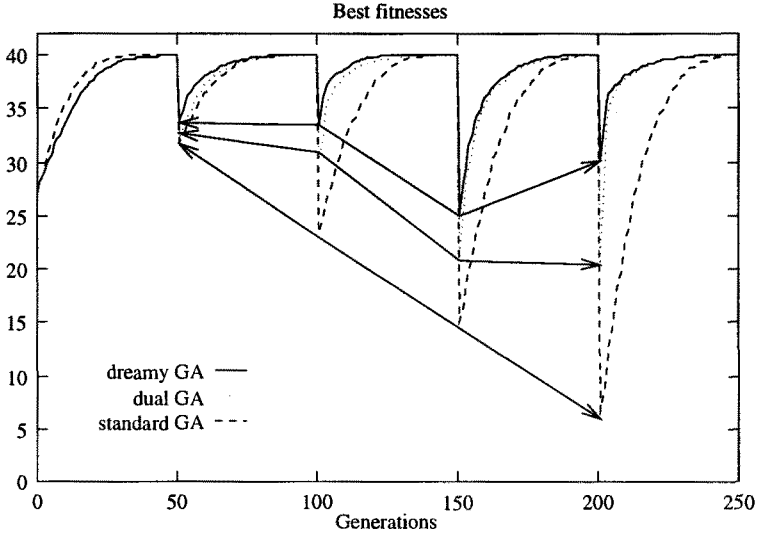


Fig. 4. Fitness evolution for the non-stationary environment.

plore, while keeping its exploitation capabilities. This confirms that changing environment can benefit from periodic reorganization of the population, in order to break symmetries [1].

Let us now, focus on how diversity evolves. Figures 5 represents respectively the genotypical and phenotypical diversities. The genotypical diversity is the mean of the normalized Hamming distance across all pairs of chromosomes in the population. In a random population this value will approximately be .5. For the dreamy-GA the genotypic diversity increases during dream phases, while the phenotypic diversity decreases to the standard one. This definitely proves its abilities to explore without endanger the quality of acquired information. A change in the environment leads all GAs to increase diversity (phenotypic in the case of standard GA, and both phenotypic and genotypic in the case of dual and dreamy-GAs). This period of increased diversity lasts longer for standard GAs than for the two other GAs; further, the amplitude of diversity is higher for dreamy-GA, than for dual or standard GAs.

5 Conclusion and further work

Our approach aims at overtaking the limits of standard GAs. First, *duality* is introduced in the population thanks to individuals with the same *phenotype* but complementary *genotypes*. Then, one of the mechanisms of chronobiology is implemented: the alternation of awake and dream phases. During the latter the population reorganizes in order to perpetuate a needed diversity. We used the metaphor of the *REM sleep* during which the system is widely disconnected from its environment. A theoretical analysis showed that during the dream phase the

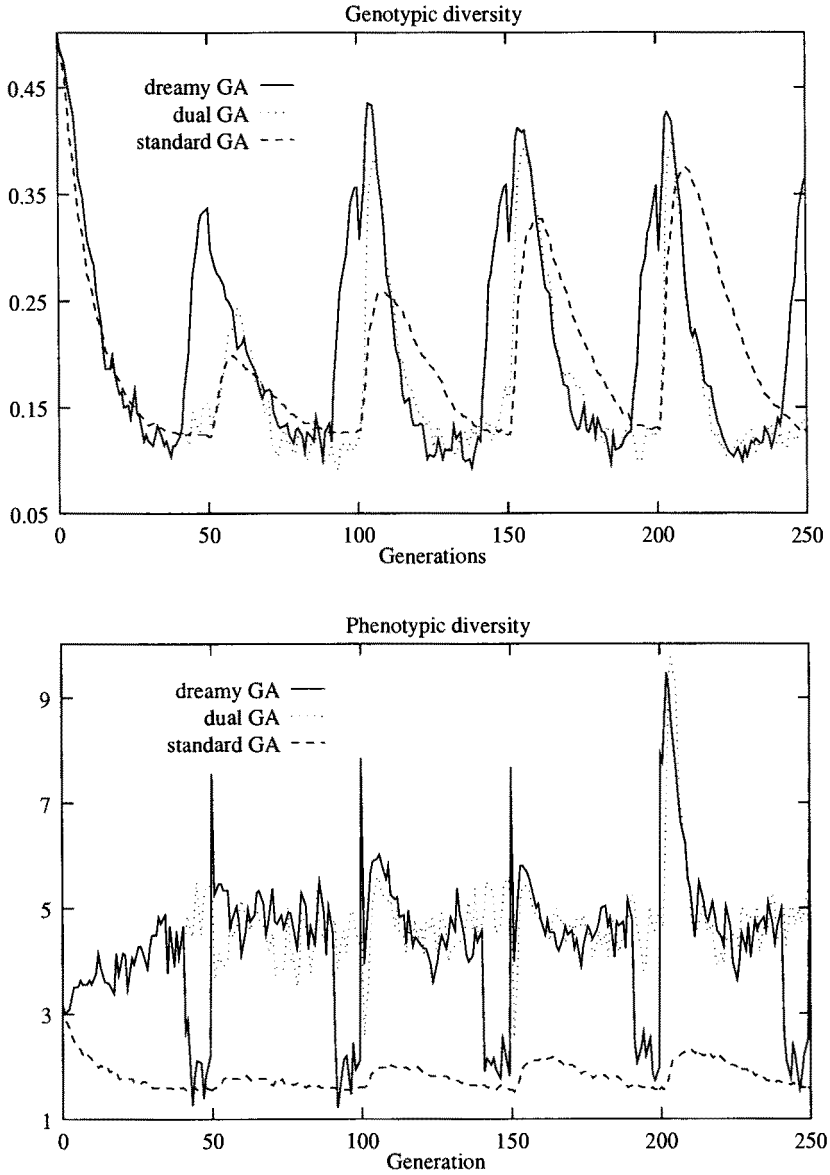


Fig.5. Diversity evolution for the non-stationary environment.

system optimally maximizes its exploration abilities while going on to exploit available information. The exploration/exploitation ratio is related to the number of dual pairs which is dynamically adapted: it decreases during the *awake* phase and increases during the *dream* one. As a consequence, a dreamy-GA allows one to increase the capabilities of evolution on rugged fitness landscapes.

In the case of a dynamic environment the preservation of diversity causes the dreamy-GA to perform better.

A natural pursuit of this work is to exploit more extensively the basic mechanisms of chronobiology. We could distinguish some rhythmic activities depending on an *endogenous rhythm*. Thus, we could have a biological clock controlled by a gene. We can imagine the profit taken from a synchronization between endogenous and autogenous rhythms.

Acknowledgments

The authors would like to thank Michèle Sebag for her detailed comments and very useful suggestions on earlier versions of this paper. Thanks are also due to the referees for their helpful remarks.

References

1. Eric Bonabeau and Guy Theraulaz. *L'intelligence collective*, chapter 8, pages 225–261. Hermes, 1994.
2. P. Chossat. *Les symétries brisées*. Ed. Belin, 1996.
3. P. Collard and J.P. Aurand. DGA: An efficient genetic algorithm. In A.G. Cohn, editor, *ECAL'94: European Conference on Artificial Intelligence*, pages 487–491. John Wiley & Sons, 1994.
4. P. Collard and C. Escazut. Genetic operators in a dual genetic algorithm. In *ICTAI'95: Proceedings of the seventh IEEE International Conference on Tools with Artificial Intelligence*, pages 12–19. IEEE Computer Society Press, 1995.
5. P. Collard and C. Escazut. Relational schemata: A way to improve the expressiveness of classifiers. In L. Eshelman, editor, *ICGA'95: Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 397–404, San Francisco, CA, 1995. Morgan Kaufmann.
6. P. Collard and C. Escazut. Fitness Distance Correlation in a Dual Genetic Algorithm. In W. Wahlster, editor, *ECAL 96: 12th European Conference on Artificial Intelligence*, pages 218–222. Wiley & Son, 1996.
7. J. Culberson. Mutation-crossover isomorphisms and the construction of discriminating functions. *Evolutionary Computation*, 2(3):279–311, 1995.
8. K. A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, University of Michigan, 1975.
9. D. E. Goldberg. Simple genetic algorithms and the minimal deceptive problem. In L. Davis, editor, *Genetic Algorithms and Simulated Annealing*, pages 74–88. Morgan Kaufmann, Los Altos, California, 1987.
10. D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In J.J Grefenstette, editor, *ICGA'87: Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49. Lawrence Erlbaum Associates, 1987.
11. J. J Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Trans. Systems, Man, and Cybernetics*, 16(1):122–128, 1986.
12. T. Jones. Crossover, macromutation and population-based search. In L. Eshelman, editor, *ICGA'95: Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 73–80. Morgan Kaufmann, 1995.

13. M. Juvet. Phylogeny of the states of sleep. *Acta psychlat. belg.*, 94:256–267, 1994.
14. Samir W. Mahfoud. *Niching methods for genetic algorithms*. PhD thesis, University of Illinois at Urbana-Champaign, 1995. IlliGAL Report 95001.
15. C. Melhuish and T. C. Fogarty. Applying a restricted mating policy to determine state space niches using immediate and delayed reinforcement. In T. C. Fogarty, editor, *Evolutionary Computing: AISB Workshop*, volume 865 of *Lecture Notes in Computer Science*, pages 224–237. Springer Verlag, 1994.
16. Edmund Ronald. When selection meets seduction. In L. Eshelman, editor, *ICGA'95: Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 167–173. Morgan Kaufmann, 1995.
17. J. David Schaffer, Richard A. Caruana, Larry J. Eshelman, and Rajarshi Das. A study of control parameters affecting online performance of genetic algorithms for function optimization. In J. D. Schaffer, editor, *ICGA'89: Proceedings of the Third International Conference on Genetic Algorithms*, pages 51–60. Morgan Kaufmann, 1989.
18. M. Sebag and M. Schoenauer. Mutation by imitation in Boolean evolution strategies. In H-M Voigt, W. Ebeling, I. Rechenberg, and H-P Schwefel, editors, *PPSN IV: The Fourth International Conference on Parallel Problem Solving from Nature*, number 1141 in *Lecture Notes in Computer Science*, pages 356–365, 1996.
19. W. M. Spears. Crossover or mutation ? In L. D. Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 221–233. Morgan Kaufmann, San Mateo, CA, 1993.
20. F. Vavak and T. C. Fogarty. A comparative study of steady state and generational genetic algorithms for use in nonstationary environments. In T. C. Fogarty, editor, *Proceedings of Evolutionary Computing, AISB Workshop*, number 1143 in *Lecture Note in Computer Science*, pages 297–304. Springer, 1996.
21. M. D. Vose and G. E. Liepins. Punctuated equilibria in genetic search. *Complex Systems*, 5:31–44, 1991.
22. L. D. Whitley. Fundamental principles of deception in genetic search. In G. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 221–241. Morgan Kaufmann, San Mateo, CA, 1991.
23. L. D. Whitley. An executable model of a simple genetic algorithm. In L. D. Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 45–62. Morgan Kaufmann, 1993.

Mimetic Evolution

Mathieu Peyral¹

Antoine Ducoulombier²

Caroline Ravisé^{1,2}

Marc Schoenauer¹

Michèle Sebag^{1,2}

(1): CMAP & LMS
URA CNRS 756 & 317
Ecole Polytechnique
91128 Palaiseau Cedex

(2): Equipe I & A, LRI
URA CNRS 410
Université d'Orsay
91405 Orsay Cedex

{*Prenom.Nom*}@polytechnique.fr

Abstract. Biological evolution is good at dealing with environmental changes: Nature ceaselessly repeats its experiments and is not misled by any explicit memory of the past. This contrasts with artificial evolution most often considering a fixed milieu, where re-generating an individual does not bring any further information.

This paper aims at avoiding such uninformative operations, via some explicit memory of the past evolution: the best and the worst individuals previously met by evolution are respectively memorized within two virtual individuals. Evolution may then use these virtual individuals as social models, to be imitated or rejected. In mimetic evolution, standard crossover and mutation are replaced by a single operator, *social mutation*, which moves individuals farther away or closer toward the models. This new scheme involves two main parameters: the social strategy (how to move individuals with respect to the models) and the social pressure (how far the offspring go toward or away from the models).

Experiments on large-sized binary problems are detailed and discussed.

1 Introduction

Biological evolution takes place in a changing environment. Being able to repeat previously unsuccessful experiments is therefore vital. As the result of previous experiments might change, any explicit memory of the past might provide misleading indications. This could explain why all knowledge gathered by evolution is actually contained in the current genetic material and dispatched among the individuals.

Inversely, artificial evolution most often tackles optimization problems and considers *fixed* fitness landscapes, in the sense that the fitness of an individual does not vary along time and does not depend on the other individuals in the population. In this framework, which is the only one considered in the rest of the paper, the evaluation of an individual produces reliable information, and generating this individual again does not provide any further information. Memorizing somehow the past of evolution thus make sense, as it could prevent evolution from some predictable failures. This paper focuses on gathering an explicit collective memory (EC-memory) of evolution, as opposed to both the

implicit memory of evolution contained in the genetic material in the population, and the local parameters of evolution conveyed by the individuals, such as the mutation step size in Evolution Strategies [23]), or the type of crossover applicable to an individual [26].

Many works devoted to the control of evolution ultimately rely on some explicit collective memory of evolution. The memorization process can acquire numerical information; this is the case for the reward-based mechanism proposed by Davis to adjust the operator rates [5], the adjustment of penalty factors in SAT problems [6] or the construction of discrete gradients [11], among others. The memorization process can also acquire symbolic information, represented as rules or beliefs characterizing the disruptive operators (so that they could be avoided) [20], or the promising schemas [22].

Memory-based heuristics can control most steps of evolution: e.g. selection via penalty factors [6], operator rates [5], operator effects [11, 20]... Memory can even be used to "remove genetics from the standard genetic algorithm" [4, 3] as in the *Population Based Incremental Learning* (PBIL) algorithm. PBIL deals with binary individuals (in $\{0, 1\}^N$) and it maintains the memory of the most fit individual encountered so far. This memory can be thought of as a *virtual individual*, belonging to $[0, 1]^N$. It provides an alternative to the genetic-like transmission of the information between successive populations: any population is generated from scratch by sampling the discrete neighbors of this virtual individual. And the virtual individual is then updated from the best current individual.

Another approach, termed *Evolution by Inhibitions* (EBI), is inversely based on memorizing the worst individuals encountered so far; the memory is also represented by a virtual individual, termed the *Loser* [25]. This memory is used to evolve the current population, by means of a single new operator termed *flee-mutation*. The underlying metaphor is that the offspring aim at being farther away from the loser, than their parents. Incidentally, this evolution scheme is biased against exploring again unfit regions previously explored.

A new evolutionary scheme, restricted to binary search space and combining PBIL and *Evolution by Inhibitions*, is presented in this paper. The memory of evolution is thereafter represented by two virtual individuals, the *Winner* and the *Loser*¹. These virtual individuals, or *Models*, respectively summarize the best and the worst individuals encountered so far by evolution. An individual can independently imitate, avoid, or ignore each one of the two models; a wide range of, so to speak, social strategies, can thereby be considered. For instance, the Entrepreneur imitates the Winner and ignores the Loser; the Sheep imitates the Winner and rejects the Loser; the Phobic rejects the Loser and ignores the Winner (the dynamics is that of Evolution by inhibitions [25]); the Ignorant ignores both models and serves as reference to check the relevance of the models. This new scheme of evolution, termed *mimetic evolution*, is rather inspired by social than genetic metaphors.

¹ Other metaphors, all likely politically incorrect, could have been used: the leader and the scapegoat, the yang and the yin, the knight and the villain,...

This paper is organized as follows. Section 2 briefly reviews related work dealing with virtual or imaginary individuals. Section 3 describes mimetic evolution and the social mutation operator replacing crossover and mutation. Social mutation is controlled from the user supplied social strategy, which defines the preferred direction of evolution of the individuals. Section 4 discusses the limitations of mimetic evolution, and studies the case where the dynamics of the models and the population go in a deadlock. Section 5 examines how far the offspring must go in the direction of the models; or, metaphorically, which social pressure should be exerted on the individuals. Mimetic mutation is validated on several large-sized binary problems, and the experimental results are detailed in section 6. Last, we conclude and present some perspectives of research.

2 State of the art

With no pretention to exhaustivity, this section examines how imaginary or virtual individuals have been used to support evolution. The central question still is the respective contribution of crossover and mutation to the dynamics of evolution [8, 18, 23]. Though the question concerns any kind of search space, only the binary case will be considered here.

The efficiency of crossover is traditionnally explained by the Building Block hypothesis [12, 9]. But a growing body of evidence suggests that crossover is also efficient because it operates large step mutations. In particular, T. Jones has studied the macro-mutation operator defined as crossing over a parent with a random individual². Macro-mutation obviously does not allow the offspring to combine the building blocks of their two parents; still, macro-mutation happens to outperform standard crossover on benchmark problems, everything else being equal [14].

In retrospect, crossover can be viewed as a biased mutation. The bias depends on the population and controls both the strength and the direction of the mutation. The "mutation rate" of standard crossover, e.g. the Hamming distance between parents and offspring, depends on average on the diversity of the population; and the "mutation direction" of standard crossover (which genes are modified) also depends on the population.

On the other hand, binary mutation primarily aims at preserving the genetic diversity of the population. This can be done as well through crossover with specific individuals, deliberately maintained in the population to prevent the loss of genetic diversity. For instance, the Surrogate GA [7] maintains imaginary individuals such as the complementary of the best current individual, or all-0 and all-1 individuals; crossover alone thus becomes sufficient to ensure the genetic diversity of the population, and mutation is no longer needed. Another possibility is to deliberately introduce genotypic diversity by embedding the search space

² Note that this macro-mutation fairly resembles standard crossover during the first generations of evolution, especially for large populations.

Ω into $\{0, 1\} \times \Omega$ and identifying the individuals 0ω and $1\bar{\omega}$, as done in Dual Genetic Algorithms [19]. Provided that the number of dual pairs (0ω and $1\bar{\omega}$) is above a given threshold, crossover can similarly replace mutation and ensure genetic diversity.

Evolution can also be supported by virtual individuals, i.e. individuals belonging neither to the population nor to the search space. This is the case in the PBIL algorithm, mentioned in the introduction, where the best individuals in the previous populations are memorized within a vector of $[0, 1]^N$. This vector noted $\mathcal{M}$ provides an alternative to crossover and mutation, in that it allows PBIL to generate the current population from scratch: for each individual X and each bit i , value X_i is randomly selected such that $P(X_i = 1) = \mathcal{M}_i$ (where A_i denotes as usual the i -th component of A). $\mathcal{M}$ is initialized to $(0.5, 0.5, \dots, 0.5)$ and it is updated from the best individual³ X_{max} at each generation, by relaxation:

$$\mathcal{M} \leftarrow (1 - \alpha)\mathcal{M} + \alpha X_{max}$$

where α in $[0, 1]$ is the relaxation factor, which corresponds to the fading of the memory. The main advantage of PBIL is its simplicity: it does not involve any modification of the genetic material. The only information transmitted from one generation to another is related to the best individual; still, it is not necessarily sufficient to reconstruct this best individual. This might hinder evolution in narrow highly fit regions, such as encountered in the Long Path problem [13]. Practically, one sees that even if $\mathcal{M}$ is close to the path, the population constructed from $\mathcal{M}$ poorly samples the path [24].

Evolution by Inhibition involves the opposite memory, that is, the memory of the worst individuals in the previous populations. This memory noted $\mathcal{L}$ (for *Loser*) is also a vector of $[0, 1]^N$, constructed by relaxation:

$$\mathcal{L} \leftarrow (1 - \alpha)\mathcal{L} + \alpha X_{min}$$

x where X_{min} denotes the average of half the worst offspring, and α is the relaxation factor. In contrast with PBIL which uses $\mathcal{M}$ to generate a new population, $\mathcal{L}$ is actually used to evolve the current population via a specific operator termed *flee-mutation*. Flee-mutation replaces both mutation and crossover; for each individual X , it selects and flips the bits most similar to those of the loser (minimizing $|X_i - \mathcal{L}_i|$). The offspring thus is farther away from the loser, than the parent was. Metaphorically, the goal of this evolutionary scheme is: Be different from the Loser ! And incidentally, this reduces the chance for exploring again low fit regions.

The potential of evolution by inhibitions is demonstrated for appropriate settings of the flee-mutation rate (number of bits mutated): EBI then significantly outperforms PBIL [3, 25], which itself outperforms most standard discrete optimization algorithms (Hill-Climbers with multiple restarts, standard GAs, binary

³ A more robust variant is to update $\mathcal{M}$ from the *two* best individuals and the worst individual in the population [3].

evolution strategies). But the adjustment of the flee-mutation rate remains an open question.

3 Mimetic evolution

This section focuses on combining evolution by inhibitions and PBIL. Two models memorizing respectively the best and worst individuals met in past generations, are constructed. These models are used to evolve individuals through a single evolution operator.

3.1 Winner-driven evolution of the population

The winner is built by relaxation from the best individuals of the current population, in the same way as in PBIL (section 2). Table 1 illustrates how the winner $\mathcal{W}$ and the loser $\mathcal{L}$ are built from the current population.

	1	2	3	4	5	Fitness
X	0	0	1	0	0	high
Y	1	1	1	1	1	high
Z	0	1	1	0	1	high
$d\mathcal{W}$	0.33	0.66	1	0.33	0.66	
S	0	0	0	1	0	low
T	1	0	1	1	1	low
U	1	0	0	1	1	low
$d\mathcal{L}$	0.66	0	0.33	1	0.66	

$$\mathcal{W} \leftarrow (1 - \alpha_w)\mathcal{W} + \alpha_w d\mathcal{W}$$

$$\mathcal{L} \leftarrow (1 - \alpha_l)\mathcal{L} + \alpha_l d\mathcal{L}$$

Table 1. *Individuals and virtual individuals*

Let us first examine how $\mathcal{W}$ can help evolving individual X . Given the most fit individuals of the population (X , Y and Z), some possible causes for being fit are ($bit_2 = 1$), or ($bit_3 = 1$), or ($bit_5 = 1$) (a majority of the most fit individuals has those bits set to this value). Thus, one might want for instance to flip bit_2 and let bit_3 unchanged in X ; this amounts to making X more similar to $d\mathcal{W}$, which goes to $\mathcal{W}$ in the limit. Metaphorically, X thus "imitates" the winner $\mathcal{W}$.

Practically, a model-driven mutation termed *social mutation* is implemented as follows. Given the number M of bits to flip (see section 5), one selects these M bits by tournament among the bits. For each one of these M bits, T bits $i_1, \dots, i_T$ are uniformly selected in $1..N$, and the bit i such that it maximizes $|X_i - \mathcal{W}_i|$, is flipped. This way, the offspring actually reduces its distance to the winner.

This mechanism can be compared to the majority crossover of Syswerda, setting the value of the offspring to the majoritary value of the bit in the population when the two parents differ on that bit [27]. The difference between majority crossover and social mutation is twofold: the majority crossover takes into account *all* individuals in the *current* population; social mutation takes into account the *best* individuals in all *past* populations.

Social mutation is easily refined to also account for the loser. For instance, according to $d\mathcal{W}$, it might be a good idea to mutate bit 5; but $d\mathcal{L}$ suggests that ($bit_5 = 1$) is not a factor of high fitness. This leads to select the bits to mutate, so that the offspring "imitates" $\mathcal{W}$ and "rejects" $\mathcal{L}$. Practically, one only modifies the tournament criterion: the winner of the tournament is the bit maximizing $|X_i - \mathcal{W}_i| - |X_i - \mathcal{L}_i|$.

Note that the relaxation factors α_w and α_l respectively associated to the winner and the loser could be different, though they are set to the same value (10^{-2}) throughout this paper. In this case, $\mathcal{L}$ changes faster than $\mathcal{W}$.

3.2 Social strategies

However, there is no reason why an individual could only imitate the winner and reject the loser. A straightforward generalization is to define a pair (δ_W, δ_L) in

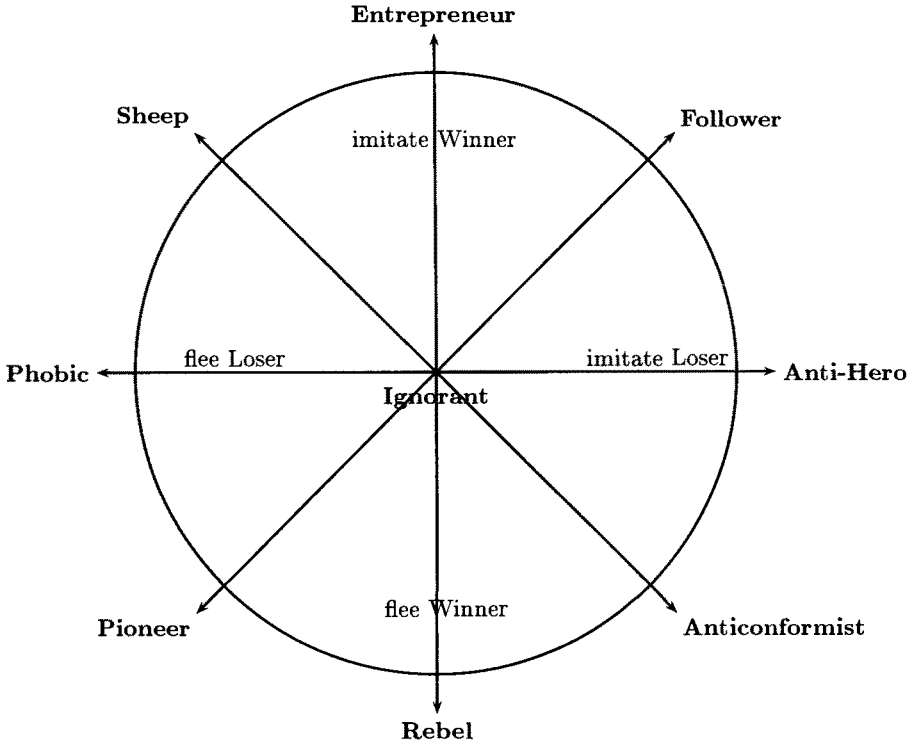


Fig. 1. Social Strategies

$\mathbb{R}^2$, and to select the bits to mutate as those maximizing

$$\delta_W |X_i - \mathcal{W}_i| + \delta_L |X_i - \mathcal{L}_i|$$

One sees that X imitates model $\mathcal{M}$ ($= \mathcal{W}$ or $\mathcal{L}$) if $\delta_M > 0$, rejects $\mathcal{M}$ if $\delta_M < 0$, and ignores $\mathcal{M}$ if $\delta_M = 0$. Social mutation finally gets parameterized by the pair (δ_W, δ_L) , termed *social strategy*. Some of these strategies have been given names for the sake of convenience; obviously, other systems of metaphors could have been imagined. We distinguish mainly:

- The *entrepreneur*, that imitates the winner and ignores the loser;
- The *sheep*, that imitates the winner and rejects the loser;
- The *phobic*, that rejects the loser and ignores the winner;
- The *pioneer*, that rejects both the winner and the loser;
- The *rebel*, that rejects the winner and ignores the loser;
- The *ignorant*, that ignores both the loser and the winner.

One notices that social mutation is unchanged if δ_W and δ_L are multiplied by a positive coefficient. With no loss of information, social strategies are thus represented by angles ($\mathbb{R}^2$ being projected onto the unit circle). This angle sets the direction of evolution of the individuals, in the changing system of coordinates given by the winner and the loser. Figure 1 shows the directions corresponding to the main social strategies, with angle 0 corresponding to imitating the loser, and angle $\pi/2$ to imitating the winner.

4 Limitations

As expected, not all social strategies are relevant and some turn out to be misleading. A clear such case is when the winner is close to a local optimum; imitating the winner just drives the population to the local optimum, which does not allow for modifying the winner; a deadlock of mimetic evolution thus occurs.

In a general way, the population and the models depend on each other: at each generation, the best (worst) individuals in the population are used to maintain the models; and these models are used to evolve the population. This dynamics can produce several kinds of deadlock.

If the population does not make any progress, so does the winner; hence any strategy based on solely imitating the winner will fail, that is, fail to re-orient the search. In the meanwhile, the loser will probably fluctuate; either imitating or rejecting the loser could help sidestep this trap.

The loser can also mislead the population. If the loser is close to the optimum, rejecting the loser will deceptively lead the population in bad regions; the loser will then change, providing more reliable beacons and allowing the population to resume its pilgrimage toward the optimum. This may produce an oscillation, satellizing the population around the optimum. Concretely, let us consider a population climbing a hill. As long as the loser is on the same side of the hill as the population, fleeing the loser will make the population duly climb the hill. But

when individuals are equally distributed on both sides of the hill, the loser gets close to the optimum, and now exerts a repelling influence on the population. This perpetuates until the symmetry is broken. Still worse, the loser may be on the other side of the hill than the majority of the population (if the relaxation factor is too large, for instance), and make the population go down the hill...

The distribution of individuals may also cause any strategy to be deceptive. Consider an even distribution of the population between the two schemas, and assume that model $\mathcal{M}$ reflects this distribution (Table 2).

Schema 1	0	1	*	*	*	...	*
Schema 2	1	0	*	*	*	...	*
$\mathcal{M}$	0.5	0.5	*	*	*	...	*

Table 2: Stable distribution of the population

Any individual X most differs from $\mathcal{M}$ by bits 1 and 2 ($|X_i - \mathcal{M}_i| = .5, i = 1, 2$). Imitating $\mathcal{M}$ means reducing the differences between X and $\mathcal{M}$, hence flipping preferably bits 1 and 2 (if the number of bits to mutate is greater than 2 — this will be discussed in the next section). But mutating bits 1 and 2 would perpetuate the distribution of the population between schemas 1 and 2.

Inversely, if the strategy is to reject $\mathcal{M}$, one wants to preserve the differences between X and $\mathcal{M}$, hence bits 1 and 2 will never be modified; and, everything being equal, this would preserve the distribution of the population too.

To sum up, observing a single model may perpetuate the traps visited by the model or the population. This is confirmed by the fact that PBIL enriches the computation of the winner with random perturbations [4], or computes the winner from the two best and the worst individuals [3]. We experimented similar heuristics based on gaussian perturbations of the models.

However, it turns out that guiding evolution according to *two* models is much more robust than with only one model: the influence of each model somehow moderates the influence of the other one, and makes it less harmful. Still, there certainly exists cases where the two combined models deliver deceptive indications. Further research will focus on the deceptivity of social strategies.

5 Social Pressure

Binary mutation is traditionnally parameterized by the mutation rate, usually very low, setting the average number of bits to mutate in the population. And the selection of the mutated bits is done at random.

In opposition, social mutation primarily concentrates on ordering the bits to mutate depending on the individual, via defining its desired direction of evolution (section 3.2). But how far the individual should go in this direction, i.e. the number M of bits to mutate, is still to determine. This parameter, termed *social pressure*, controls the balance of evolution between exploitation and exploration, respectively achieved for low and high values of M . In any case, the social

pressure must correspond to a much higher mutation rate than for standard mutation, as social mutation is meant to replace both mutation and crossover.

A previous work investigated two heuristics for the auto-adjustment of M in the framework of evolution by inhibitions [25]. The first heuristic is inspired from Davis [5], and proceeds by rewarding the values of M leading to fitness increases. The second one is inspired from self-adaptation [23, 1]: the number M is encoded within each individual, and evolution supposedly optimizes M as well as the genotypic information of the individual.

Unfortunately, none of our attempts succeeded in determining relevant global or local values of M ; rather, all heuristics rapidly lead to setting $M = 1$. Mimetic evolution thereafter behaves as a standard hill-climber, and soon gets trapped in a local optimum.

In retrospect, reward-based adjustment tends to be risk-averse, and favor options that bring small frequent improvements, over options that bring rare, though large, improvements; this explains why $M = 1$ is preferred.

On the other hand, the self-adaptation of M fails because the strong causality principle [21] is violated: finding the optimal M amounts to finding an optimal *discrete* value in a very restricted range (say $[2..N/10]$); no wonder that the convergence results of evolution strategies [23, 1] do not apply.

We therefore used fixed schedules to determine M_t at generation t . The simplest possibility is to set M_t to a fixed, user-supplied value M_0 . A more sophisticated possibility is to decrease M_t from an initial, user-supplied, value M_0 . We used a decreasing hyperbolic schedule borrowed from [2]:

$$M_t = \frac{1}{\frac{1}{M_0} + t \cdot \frac{1 - \frac{1}{M_0}}{T-1}} \quad (1)$$

where T is the maximum number of generations and t denotes the current generation. Social mutation then mutates exactly the integer part of M_t bits in each individual⁴.

It now remains to set the initial (or fixed) value of M_t , M_0 . We used off-line adjustments loosely inspired from Grefenstette's Virtual GA [10]. More precisely, a (1+10) binary ES is run for a few generations, for each considered value of M_0 (2.5 in the constant schedule, and {45, 90, 450} in the decreasing schedule), and one chooses the value of M_0 leading to the best performance.

6 Experimental Validation

This section discusses the experimental results obtained by mimetic evolution on large-sized problems, and focuses on the role of social strategy.

⁴ Another possibility, not investigated here, is to mutate on average M_t bits on the population [25].

6.1 Problems

The experiments consider some standard test functions: the OneMax, the Twin-Peaks and the deceptive) Ugly⁵ functions, taken on $\{0, 1\}^{900}$.

Other functions are taken from [3], and respectively correspond to the binary coding and the Gray coding of the continuous functions on $[-2, 56, 2.56]^{100}$ below:

$$\begin{aligned}
 y_1 &= x_1 \\
 y_i &= x_i + y_{i-1}, \quad i \geq 2 \\
 F_1 &= \frac{100}{10^{-5} + \sum_i |y_i|} \\
 F_3 &= \frac{100}{10^{-5} + \sum_i |.024 * (i + 1) - x_i|}
 \end{aligned}$$

In the latter case, each continuous interval $[-2, 56, 2.56]$ is mapped onto $\{0, 1\}^9$; individuals thus belong to $\{0, 1\}^{900}$.

The importance of the coding is witnessed, if necessary, by the fact that F_3 only reaches a maximum of 416, 63 in its binary version, whereas the continuous optimum is 10^7 ; this is due to the fact that the continuous optimum ($X_i = .024 * (i + 1)$) does not belong to the discrete space considered.

6.2 Experimental setting

The evolution scheme is a (10+50)-ES: 10 parents produce 50 offspring and the 10 best individuals among parents plus offspring are retained in the next population. A run is allowed 200,000 evaluations; all results are averaged on 10 independent runs.

The winner and the loser respectively memorize the best and the worst offspring of the current generation. The relaxation factors α_w and α_l are both set to .01. The tournament size is set to 20.

Several reference algorithms have been experimented on functions F_1 and F_3 : two variants of GAs (GA1 and GA2), two variants of Hill-climbers (HC1 and HC2) [3] and two variants of evolution strategies (ES1 and ES2) [25]. These algorithms served as reference for PBIL and evolution by inhibitions (INH). Another reference algorithm is given by mimetic evolution following an ignorant social strategy (an offspring is generated by randomly mutating 3 bits).

Function	HC1	HC2	AG1	AG2	AES	TES	PBIL	INH	IGNOR
F1 Binary	1.04	1.01	1.96	1.72	2.37	1.87	2.12	2.99	2.98
F3 Gray	416.65	416.65	28.35	210.37	380.3	416.65	366.77	246.23	385.90

Table 3. Reference results on F_1 and F_3 .

More results, and the detailed description of the reference algorithms, are found in [25]. In this paper, we focus on the influence of the social strategy

⁵ Defined as 300 concatenations of the elementary deceptive function U defined on $\{0, 1\}^3$ as follows: $U(111) = 3$; $U(0XX) = 2$; otherwise $U = 0$.

parameter, and in particular, the social pressure (nb of bits mutated) is fixed to 3.

6.3 The influence of social strategies

A new visualization format is proposed on Figure 2, in order to compare the results obtained for different strategies on a given problem. As social strategies can be represented as angles, results are plotted in polar coordinates: point (ρ, θ) illustrates the results obtained for strategy θ , with ρ being the best average fitness obtained for this strategy. Two curves are plotted by joining all points (ρ, θ) : the internal curve gives the results obtained for 50,000 evaluations, and the external one gives the results obtained for 200,000 evaluations. Both curves overlap (e.g. for OneMax and Ugly) when evolution reaches the optimum in about 50,000 evaluations or less.

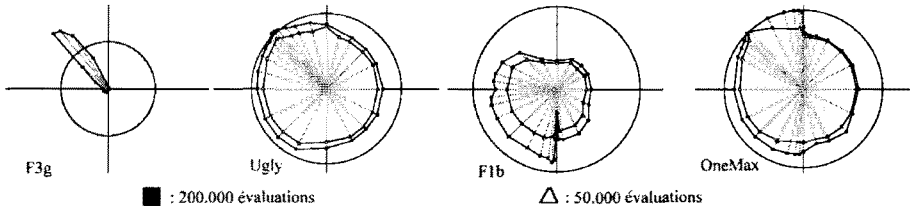


Fig. 2. Summary results of social strategies

The graphics obtained for functions TwinPeaks and F_{1g} , quite similar to respectively those of OneMax and F_{1g} , are omitted given the space limitations.

Note that the shape of the curve obtained for 50,000 evaluations is very close to that obtained for 200,000 evaluations; in other words, the social strategy most adapted to a problem does not seem to vary along evolution.

In all cases, strategies in the left half space are significantly better than those in the right half space: for all functions considered, the recommended behavior with respect to the loser is the flight. In opposition, the recommended behavior with respect to the winner depends on the function: flight is strongly recommended in the case of F_1 , where the Pioneer strategy is the best. Inversely, imitation is strongly recommended in the case of F_3 , OneMax and Ugly, where the Sheep strategy is the best. This is particularly true for F_3 , where the Sheep is significantly better than other strategies: the respective amounts of imitation of the winner and avoidance of the loser must be carefully adjusted.

7 Conclusion

Our working hypothesis is that the memory of evolution can provide worth information to guide the further steps of evolution.

Instead of memorizing only the best [4] or the worst [25] individuals, mimetic evolution memorizes all striking past individuals, the best and the worst ones. A population can thereby access two models, featured as the Winner and the Loser.

These models define a changing system of coordinates in the search space; an individual can thus be given a direction of evolution, or social strategy, expressed with respect to both models. And mimetic evolution proceeds by moving each individual in this direction. This paper has only considered the case of a single social strategy, fixed for all individuals, and focus on the influence of the strategy on the dynamics of evolution.

Evolution by inhibitions is a special case, driven by the loser model only, of mimetic evolution. As mimetic evolution involves both loser and winner models, the possible strategies get richer. As could be expected, 2-model-driven strategies appear more robust than 1-model-driven ones. Indeed, the more models are observed, the less predictable and deterministic the behavior of the population gets; and the less likely evolution goes in a deadlock.

Experimental results show that for each test function there exists a range of optimal social strategies (close to the Sheep in most cases; and to the Pioneer/Rebel in the others). And on most functions, these optimal strategies significantly outperform the "ignorant" strategy, which serves as reference and ignores both models.

In any case, it appears that memory contains relevant information, as the way we use it fortunately often allows for speeding up evolution. Obviously, other and more clever exploitations of this information remain to be invented. On the other hand, the use of the memory is tightly connected to its content: which knowledge exactly should be acquired during evolution ? Currently the models reflect, or "capitalize", the individuals; another level of memory would capitalize the dynamics of the population as a whole, i.e. the sequence over all generations of the social strategy leading from the best parents to the best offspring.

A further perspective of research is to self-adapt the social strategy of an individual, by enhancing the individual with its proper strategy parameters δ_W and δ_L . This way, evolution could actually benefit from a mixture of different social strategies, dispatched within the population⁶.

Another perspective is to extend mimetic evolution to continuous search spaces. Computing the winner and the loser from continuous individuals is straightforward; but the question of how to use them is still more open, than in the binary case.

⁶ Incidentally, this would reflect more truly the evolution of societies. But indeed social modelling is far beyond the scope of this work.

A last perspective is to see to what extent the difficulty of the fitness landscape for a standard GA is correlated to the optimal social strategy for this landscape (among which the ignorant strategy). Ideally, the optimal strategy would allow one to compare diverse components of evolution, in the line of the Fitness Distance Correlation criterion [15]. The advantage of such criteria is to provide a priori estimates of the adequacy of e.g., procedures of initialization of the population [16], or evolution operators [17].

References

1. T. Bäck. *Evolutionary Algorithms in theory and practice*. New-York:Oxford University Press, 1995.
2. T. Bäck and M. Schütz. Intelligent mutation rate control in canonical GAs. In Z. W. Ras and M. Michalewicz, editors, *Foundation of Intelligent Systems 9th International Symposium, ISMIS '96*, pages 158–167. Springer Verlag, 1996.
3. S. Baluja. An empirical comparizon of seven iterative and evolutionary function optimization heuristics. Technical Report CMU-CS-95-193, Carnegie Mellon University, 1995.
4. S. Baluja and R. Caruana. Removing the genetics from the standard genetic algorithms. In A. Frieditis and S. Russel, editors, *Proceedings of ICML95*, pages 38–46. Morgan Kaufmann, 1995.
5. L. Davis. Adapting operator probabilities in genetic algorithms. In J. D. Schaffer, editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 61–69. Morgan Kaufmann, 1989.
6. A.E. Eiben and Z. Ruttkay. Self-adaptivity for constraint satisfaction: Learning penalty functions. In T. Fukuda, editor, *Proceedings of the Third IEEE International Conference on Evolutionary Computation*, pages 258–261. IEEE Service Center, 1996.
7. I.K. Evans. Enhancing recombination with the complementary surrogate genetic algorithm. In T. Bäck, Z. Michalewicz, and X. Yao, editors, *Proceedings of the Fourth IEEE International Conference on Evolutionary Computation*, pages 97–102. IEEE Press, 1997.
8. D.B. Fogel and L.C. Stayton. On the effectiveness of crossover in simulated evolutionary optimization. *BioSystems*, 32:171–182, 1994.
9. D. E. Goldberg. *Genetic algorithms in search, optimization and machine learning*. Addison Wesley, 1989.
10. J. J. Grefenstette. Virtual genetic algorithms: First results. Technical Report AIC-95-013, Navy Center for Applied Research in Artificial Intelligence, 1995.
11. N. Hansen, A. Ostermeier, and A. Gawelczyk. On the adaptation of arbitrary normal mutation distributions in evolution strategies: The generating set adaptation. In L. J. Eshelman, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 57–64. Morgan Kaufmann, 1995.
12. J. H. Holland. *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor, 1975.
13. J. Horn and D.E. Goldberg. Genetic algorithms difficulty and the modality of fitness landscapes. In L. D. Whitley and M. D. Vose, editors, *Foundations of Genetic Algorithms 3*, pages 243–269. Morgan Kaufmann, 1995.

14. T. Jones. Crossover, macromutation and population-based search. In L. J. Eshelman, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 73–80. Morgan Kaufmann, 1995.
15. T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In L. J. Eshelman, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 184–192. Morgan Kaufmann, 1995.
16. L. Kallel and M. Schoenauer. Alternative random initialization in genetic algorithms. In Th. Bäck, editor, *Proceedings of the 7th International Conference on Genetic Algorithms*. Morgan Kaufmann, 1997. To appear.
17. L. Kallel and M. Schoenauer. A priori predictions of operator efficiency. In *Artificial Evolution'97*. CMAP - Ecole Polytechnique, October 1997.
18. J. R. Koza. *Genetic Programming: On the Programming of Computers by means of Natural Evolution*. MIT Press, Massachusetts, 1992.
19. P. Collard and J.P. Aurand. Dual ga: An efficient genetic algorithm. In *Proceedings of European Conference on Artificial Intelligence*, pages 487–491. Amsterdam, Wiley and sons, August 1994.
20. C. Ravisé and M. Sebag. An advanced evolution should not repeat its past errors. In L. Saitta, editor, *Proceedings of the 13th International Conference on Machine Learning*, pages 400–408, 1996.
21. I. Rechenberg. *Evolutionstrategie: Optimierung Technischer Systeme nach Prinzipien des Biologischen Evolution*. Fromman-Holzboog Verlag, Stuttgart, 1973.
22. R.G. Reynolds. An introduction to cultural algorithms. In *Proceedings of the 3rd Annual Conference on Evolutionary Programming*, pages 131–139. World Scientific, 1994.
23. H.-P. Schwefel. *Numerical Optimization of Computer Models*. John Wiley & Sons, New-York, 1981. 1995 – 2nd edition.
24. M. Sebag and M. Schoenauer. Mutation by imitation in boolean evolution strategies. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Proceedings of the 4th Conference on Parallel Problems Solving from Nature*, pages 356–365. Springer-Verlag, LNCS 1141, 1996.
25. M. Sebag, M. Schoenauer, and C. Ravisé. Toward civilized evolution: Developing inhibitions. In Th. Bäck, editor, *Proceedings of the 7th International Conference on Genetic Algorithms*. Morgan Kaufmann, 1997.
26. W. M. Spears. Adapting crossover in a genetic algorithm. In R. K. Belew and L. B. Booker, editors, *Proceedings of the 4th International Conference on Genetic Algorithms*. Morgan Kaufmann, 1991.
27. G. Syswerda. A study of reproduction in generational and steady state genetic algorithm. In G. J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 94–101. Morgan Kaufmann, 1991.

Adaptive Penalties for Evolutionary Graph Coloring

A.E. Eiben, J.K. van der Hauw

Leiden University, The Netherlands, {gusz,jvdhauw}@wi.leidenuniv.nl

Abstract. In this paper we consider a problem independent constraint handling mechanism, Stepwise Adaptation of Weights (SAW) and show its working on graph coloring problems. SAW-ing technically belongs to the penalty function based approaches and amounts to modifying the penalty function during the search. We show that it has a twofold benefit. First, it proves to be rather insensitive to its technical parameters, thereby providing a general, problem independent way to handle constrained problems. Second, it leads to superior EA performance. In an extensive series of comparative experiments we show that the SAW-ing EA outperforms a powerful graph coloring heuristic algorithm, DSatur, on the hardest graph instances and has a linear scale-up behaviour.

1 Introduction

In this paper we consider an adaptive mechanism for constraint handling (called SAW-ing) on graph 3-coloring problems. In [13] SAW-ing was applied to 3SAT problems and the resulting EA turned out to be superior to WGSAT, the best heuristics for 3SAT problems known at the moment. It is interesting to note that optimizing the population size and the operators in the the SAW-ing EA for 3SAT resulted in an algorithm that was very similar to WGSAT itself. This EA was, however, obtained independently from WGSAT, starting with a full blown EA with a large population and using crossover. It was an extensive test series that showed that a $(1, \lambda)$ selection scheme using mutation only is superior. Despite the different origin of the two compared methods, their similarity in the technical sense might suggest that the superior performance of the SAW-ing EA is just a coincidence: it holds for 3SAT, but not for other constraint satisfaction problems. In this paper we show that this is not the case.

Graph coloring falls in the category of grouping problems. Several authors [14, 15, 24] have considered grouping problems and gave arguments that they cannot be successfully solved by usual genetic algorithms, e.g. using traditional representations and the corresponding standard operators, and proposed special representation and crossovers for such problems. In this paper we show the viability of another approach to solve a grouping problem based on an adaptively changing fitness function in an EA using a common representation and standard operators.

We restrict our investigation to graph 3-coloring problems that are pure constraint satisfaction problems, unlike the constrained optimization version as

studied by for instance Davis [7]. To evaluate the performance of our EA we also run a powerful traditional graph coloring algorithm on the same problems. The final comparison shows that the SAW-ing EA is superior to the heuristic method on the hardest problem instances.

The rest of the paper is organized as follows. In Section 2 we specify the problems we study and give a brief overview on traditional graph coloring algorithms. We select one of them, DSatur, as a competitor we compare the performance of EAs with. Section 3 summarizes our results on graph coloring obtained with a EA, and compares this EA with DSatur, as well as a hybridized EA+DSatur system. Thereafter, in Section 4 we present an adaptive mechanism that is changing the penalty function, thus the fitness landscape, during a EA run. We show that the adaptive EA highly outperforms other tested EA variants, including the hybrid system. Finally, in Section 5 we compare the adaptive EA with DSatur and conclude that the EA is superior with respect to performance on hard problem instances as well as concerning scale-up properties.

2 Graph Coloring

In a graph 3-coloring problem the task is to color each vertex $v \in V$ of a given undirected graph $G = (V, E)$ with one of three colors from $\{1, 2, 3\}$ so that no two vertices connected by an edge $e \in E$ are colored with the same color. This problem in general is NP-complete [16] making it theoretically interesting; in the meanwhile there are many specific applications like register allocation [3], time-tabling [25], scheduling and printed circuit testing [17]. In the literature there are not many benchmark 3-colorable graphs and therefore we create graphs to be tested with the graph generator written by Culberson.¹ Creating 3-colorable test graphs happens by first pre-partitioning the vertices in three sets (3 colors) and then drawing edges randomly by a certain probability p , the edge density. We generated *equi-partite 3-colorable graphs* where the three color sets are as nearly equal in size as possible, as well as *flat 3-colorable graphs*, where also the variation in degree for the vertices is kept to a minimum. Determining the chromatic number of these two types of graphs is very difficult, because there is no information a (heuristic) coloring algorithm could rely on, [6]. Our tests showed that they are also tough for 3-coloring. Thorough this paper we will denote graph instances by, for example $G_{eq,n=500,p=0.10,s=1}$, standing for an equi-partite 3-colorable graph with 500 vertices, edge probability 10% and seed 1 for the random generator.

Cheeseman *et al.* [4] found that NP-complete problems have an ‘order parameter’ and that the hard problems occur at a critical value or *phase transition* of such a parameter. For graph coloring, this order parameter is the edge probability or edge connectivity p . Theoretical estimations of Clearwater and Hogg [5] on the location of the phase transition, supported by empirical validation, improved the estimates in [4] and indicate that the hardest graphs are those with

¹ Source code in C is available via <ftp://ftp.cs.ualberta.ca/pub/joe/GraphGenerator/generate.tar.gz>

an edge connectivity around $7/n - 8/n$. Our experiments confirmed these values. We will use these values in the present investigation and study large graphs with up to 1500 vertices.

To compare the performance of our EAs with traditional graph coloring algorithms we have looked for a strong competitor. There are many (heuristic) graph coloring algorithms in the literature, for instance an $O(n^{0.4})$ -approximation algorithm by Blum [1], the simple Greedy algorithm [20], DSatur from Brélaz [2], Iterated Greedy (IG) from Culberson and Luo [6], XRLF from Johnson *et al.* [19]. We have chosen DSatur as competitor for its high performance. DSatur uses a heuristic to dynamically change the ordering of the nodes and then applies the greedy method to color the nodes:

- A node with highest saturation degree (= number of differently colored neighbors) is chosen and given the smallest color that is still possible.
- In case of a tie, the node with highest degree (= number of neighbors that are still in the uncolored subgraph) is chosen.
- In case of a tie a random node is chosen.

Because of the random tie breaking, DSatur is a stochastic algorithm and, just like for the EA, results of several runs need to be averaged to obtain useful comparisons. For the present investigation we implemented the backtracking version of Turner [23], which backtracks to the lastly evaluated node that still has available colors to try.

3 The Evolutionary Algorithm

We implemented different steady-state algorithms using worst fitness deletion, with two different representations and tested different operators and population sizes for their performance. These tests were intended to check the hypotheses in [14, 15, 24] on the disadvantageous effects of crossover in standard representations, as well as to find a good setup for our algorithm. For a full overview of the test results see [12], here we only present the most interesting findings. It turned out that mixing information of different individuals by crossover is not as bad as is generally assumed. Using integer representation, each gene in the chromosome belongs to one node and can take three different values as alleles (with the obvious semantics). Applying heavy mixing by multi-point crossovers and multi-parent crossovers, [8], improves the performance. In Figure 1 we give an illustration for the graph $G_{eq, n=200, p=0.08, s=5}$, depicting the Average Number of Evaluations to a Solution (AES) as a function of the number of parents in diagonal crossover, [8], respectively as a function of the number of crossover points in m-point crossover. The results are obtained by averaging the outcomes of 100 independent runs.

Integer representation, however, turned out to be inferior to order-based representation. Using order-based representation, each chromosome is a permutation of nodes, and a decoder is needed to create a coloring from a permutation.

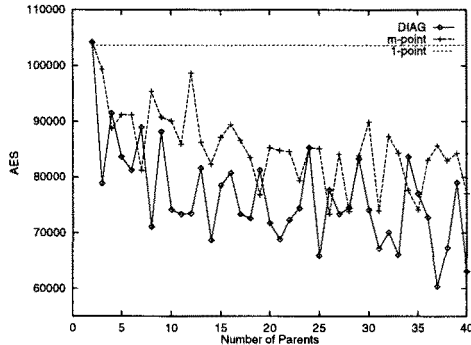


Fig. 1. Effect of more parents and more crossover points on the Average Number of Evaluations to a Solution (AES) for $G_{eq,n=200,p=0.08,s=5}$

We used a simple coloring decoder, encountering the nodes in the order they occur in a certain chromosome and giving each node the smallest² possible color. If each of the three colors leads to constraint violation, the node is left uncolored, and the fitness of a chromosome (to be minimized) is the number of uncolored nodes. After performing numerous tests the best option turned out to be an order-based GA without crossover, using mutation only with population size 1 in a (1+1) preservative selection scheme. Because of the lack of crossover we call this algorithm an EA (evolutionary algorithm), rather than a GA (genetic algorithm). This EA forms the basis of our further investigation: we will try to improve it by hybridization and by adding the SAW-ing mechanism.

Let us note that DSatur also uses an ordering of nodes as the basis to construct a coloring (in particular, it uses an ordering based on the degrees to break ties). It is thus a natural idea to use an EA to find better orderings than those used by DSatur. Technically, DSatur would still use the (dynamically found) saturation degree to select nodes and to color them with the first available color, but now it would break ties when two nodes have equal saturation degree by using a permutation (an individual in the EA) for ordering the nodes. From an EA point of view we can see DSatur as a new decoder for the EA which creates a coloring when fed with a permutation. We also tested this hybridized EA+DSatur system where the fitness value of a given permutation is the same as for the greedy decoder. The comparison between the order-based EA, DSatur with backtracking and the hybrid system is shown in the first three rows of Table 1 in Section 4.1 for the graph $G_{eq,n=1000,p=0.010}$. These results are based on four random seeds for generating graph instances, for each instance 25 independent runs were executed with $T_{max} = 300.000$ as the maximum number of evaluations for every algorithm. In the table column SR stands for Success Rate, i.e. the % of cases where the graph could be colored, the column AES is again

² Colors are denoted by integers 1,2,3

the average number of evaluations to a solution.

Based on Table 1 we can make a number of interesting observations. First, the results show even the best EA cannot compete with the DSatur, but hybridizing the two systems leads to a coloring algorithm that outperforms both of its components. Second, the performance of the algorithms is highly dependent on the random seeds used for generating the graphs. This means that they are not really powerful, they cannot color graphs with a high certainty.

4 The SAW-ing Evolutionary Algorithm

In this section we extend the EA used in the previous tests. Let us first have a look on the applied penalty function that concentrates on the nodes (variables to be instantiated), rather than on the edges (constraints to be satisfied). Formally, the function f to be minimized is defined as:

$$f(x) = \sum_{i=1}^n w_i \cdot \chi(x, i) \quad (1)$$

where w_i is the local penalty (or weight) assigned to node x_i and

$$\chi(x, i) = \begin{cases} 1 & \text{if node } x_i \text{ is left uncolored} \\ 0 & \text{otherwise} \end{cases}$$

In the previous section we were simply counting the uncolored nodes, thus used $w_i \equiv 1$. This weight distribution does not distinguish between nodes, although it is not reasonable to assume that all nodes are just as hard to color. Giving hard nodes a high weight is a very natural idea, since this gives the EA a high reward when satisfying them, thus the EA will ‘concentrate’ on these nodes. A major obstacle in doing this is, of course, that the user does not know which nodes are hard and which ones are easy. Heuristic estimates of hardness can circumvent this problem, but still, the hardness of a node is most certainly also depending on the problem solver, i.e. coloring algorithm, being applied, that is a node that is hard for one problem solver may be easy for another one. Furthermore, being hard may be also context dependent, i.e. may depend on the information the problem solver collected in a certain phase of the search. This means that even for one specific problem solver, a particular setting of weights may become inappropriate as the search proceeds. A simple answer to these drawbacks is embodied by the following mechanism.

4.1 Stepwise Adaptation of Weights

Motivated by the above reasons we decided to leave the decisions of the hardness of different nodes to the EA itself, moreover we allow the EA to revise its decisions during the search. Technically this means that we apply a varying fitness function that is repeatedly modified, based on feedback concerning the progress of the search process. Similar mechanisms have been proposed earlier in

another context by, for instance, Moris [21] and Selman and Kautz [22]. In evolutionary computation varying parameters can be divided into three classes [18], dynamic, adaptive and self-adaptive parameter control. Our approach falls in the adaptive category. The general idea is now implemented by repeatedly checking which nodes in the best individual³ violate constraints and raising the penalty w_i belonging to these nodes. Depending on when the weights are updated we can distinguish an off-line (after the run, used in the next run) and an on-line (during the run) version of this technique. In [9] and [10] the off-line version was applied, here we will use the on-line version. In particular, the EA starts with a standard setting of $w_i \equiv 1$ for each node. After each T_p fitness evaluations the best individual in the population is checked and the weights belonging to its uncolored nodes are increased by Δw , i.e. setting $w_i = w_i + \Delta w$. This mechanism introduces two new parameters, T_p and Δw and it is important to test whether the EA performance is sensitive for the values of these parameters. Limited by space requirements here we can only give an illustration. Figure 2 shows the success rates of an asexual (only SAWP mutation) and a sexual EA (OX2 crossover and SWAP mutation) for different values of Δw and T_p . These

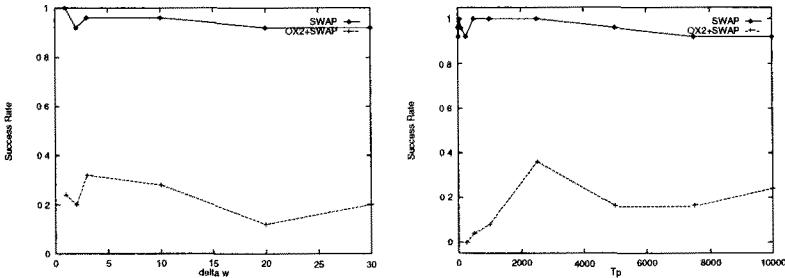


Fig. 2. Influence of Δw (left) and T_p (right) on SR. Tested with $T_{max} = 300000$ on $G_{eq, n=1000, p=0.01, s=5}$, $T_p = 250$ is used for Δw , $\Delta = 1$ is used for T_p .

tests reconfirm that using mutation only is better than using crossover and mutation, furthermore they indicate that the parameters of the SAW mechanism have only neglectable influence on the EA performance. For no specific reason we use $\Delta w = 1$ and $T_p = 250$ in the sequel.

Obviously, an EA using the SAW mechanism searches on an adaptively changing fitness landscape. It is interesting to see how the fitness (actually the penalty expressing the error belonging to a certain chromosome) changes under the SAW regime. Figure 3 shows a typical run.

As we see on the plot, the error is growing up to a certain point and then

³ More than one individuals could be also monitored, but preliminary tests did not indicate advantage of this option.

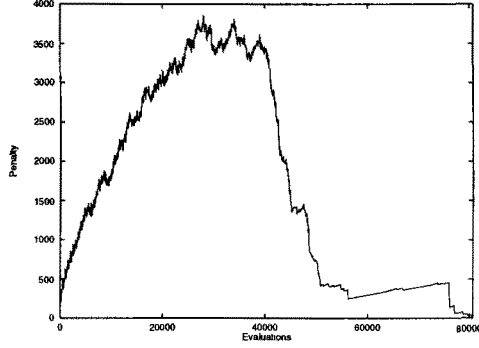


Fig. 3. Penalty curve of the best chromosome in an asexual SAW-ing EA, tested on $G_{eq,n=1000,p=0.01,s=5}$ with $T_{max} = 300000$.

it quickly decreases, finally hitting the zero-line, indicating that a solution has been found. In high resolution (not presented here) the curve shows decreasing error rates in periods of fixed T_p , followed by a sudden increase when T_p is reset, that is, we see the shape of a **saw**!

4.2 Performance of the SAW-ing EA

In Table 1 we present the performance results of the SAW-ing EA and the earlier tested EA versions. Comparing the results we see that the performance of the EA highly increases by the usage of SAW-ing, the success rates become very high and the algorithm is twice as fast as the other ones. Besides, the SAW-ing EA performs well independently from the random seeds, i.e., it is very robust.

	$s = 0$		$s = 1$		$s = 2$		$s = 3$		all 4 seeds	
	SR	AES	SR	AES	SR	AES	SR	AES	SR	AES
DSatur	0.08	125081	0.00	300000	0.00	300000	0.80	155052	0.22	220033
EA	0.24	239242	0.00	300000	0.00	300000	0.12	205643	0.09	261221
EA+DSatur	0.44	192434	0.24	198748	0.00	300000	0.64	114232	0.33	201354
EA+SAW	0.96	76479	0.88	118580	0.92	168060	0.92	89277	0.92	113099

Table 1. Comparing DSatur with backtracking, the EA, the hybrid EA and the SAW-ing EA for $n = 1000$ and $p = 0.010$ with different random seeds.

5 The SAW-ing EA vs. DSatur

The experiments reported in the previous sections clearly indicate the superiority of the SAW-ing EA with respect to other EAs. The real challenge to our weight adaptation mechanism is, however, a comparison with a powerful heuristic graph coloring technique. We have performed an extensive comparison between the SAW-ing EA and DSatur on three different types of graphs (arbitrary 3-colorable, equi-partite 3-colorable and flat 3-colorable) for different sizes ($n = 200, 500, 1000$) and a range of edge connectivity values comparing SR as well as AES. Space limitations prevent us from presenting all figures; the inter-

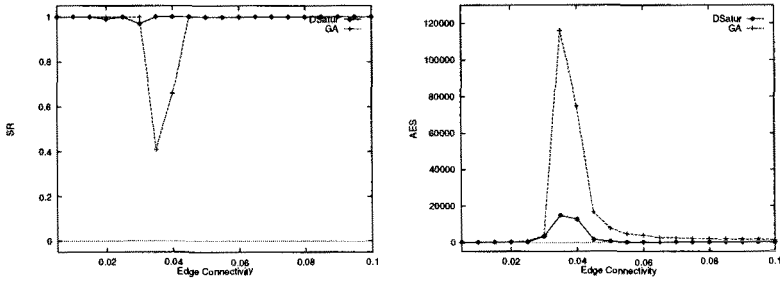


Fig. 4. Comparison of SR (left) and AES (right) for $n = 200$.

ested reader is again referred to [12]. Here we give an illustration on the hardest case: flat 3-colorable graphs. Comparative curves of success rates and the number of evaluations to a solution are given in Figure 4 and Figure 5 for $n = 200$ and $n = 1000$ respectively. The phase transition is clearly visible on each figure: the

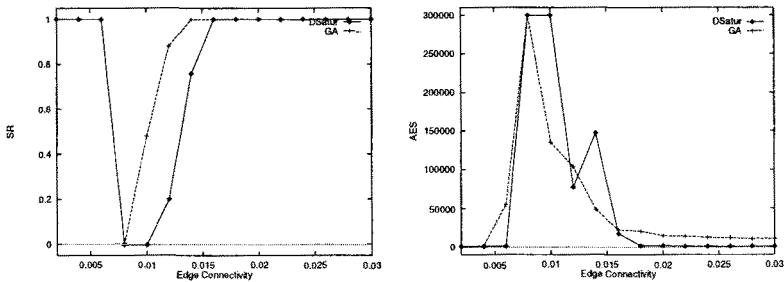


Fig. 5. Comparison of SR (left) and AES (right) for $n = 1000$.

performance of both algorithms drops at certain values of the edge connectivity p . On small graphs ($n = 200$), the deterioration of performance is smaller for DSatur, while on large graphs ($n = 1000$) the valley in the SR curve and the peak in the AES curve are narrower for the EA, showing its superiority to DSatur on these problems. The reason for these differences could be that on the small instances DSatur with backtracking is able to get out of local optima and find solutions, while this is not possible anymore for large graphs where the search space becomes too big. On the other two graph topologies (arbitrary 3-colorable and equi-partite 3-colorable) the results are similar.

An additional good property of the SAW-ing EA is that it can take more advantage of extra time given for search. For instance, on $G_{eq,n=1000,p=0.008,s=5}$ both algorithms fail (SR=0.00) when 300.000 evaluations are allowed. If we increase the total number of evaluations from 300.000 to 1.000.000 DSatur still has SR=0.00, while the performance of the EA raises from SR=0.00 to SR=0.44 (AES=407283). This shows that the EA is able to benefit from more time given, where the extra time for backtracking is not enough to get out of the local optima.

Finally, let us consider the issue of scalability, that is the question of how the performance of an algorithm changes if the problem size grows. Experiments on the hardest instances at the phase transition for $p = 8/n$ show again that DSatur is not able to find solutions on large problems. Since this leads to SR = 0 and undefined AES (or $AES = T_{max}$), we rather perform the comparison on easier problem instances belonging to $p = 10/n$. The results are given in Figure 6. These figures clearly show that the SAW-ing EA outperforms DSatur. Moreover, the comparison of the AES curves suggests a linear time complexity of the EA. Taking the scale-up curves into consideration also eliminates a possible drawback of using AES for comparing two different algorithms. Recall, that AES is the average number of search steps to a solution. For an EA a search step is the creation and evaluation of a new individual (a new coloring), i.e. AES is the average number of fitness evaluations. A search step of DSatur, however, is a backtracking step, i.e. giving a node a new color. Thus, the computational

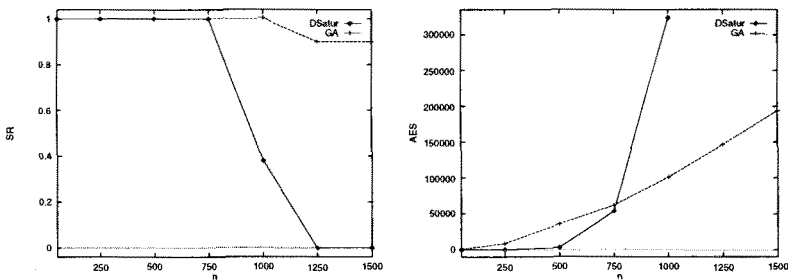


Fig. 6. Scale-up curves for SR (left) and AES (right) for $p = 10/n$.

complexity of DSatur is measured differently than that of an EA - a problem that cannot be circumvented since it is rooted in the different nature of these search algorithms. However, if we compare how the AES changes with growing problem sizes then regardless to the different meaning of 'AES' this comparison is fair.

6 Conclusions

In this paper we considered the Stepwise Adaptation of Weights mechanism that changes the penalty function based on measuring the error of solutions of a constrained problem during an EA run. Comparison of the SAW-ing EA with a simple EA, DSatur and a hybrid EA+DSatur system as showed in Table 1 discloses that SAW-ing is not only powerful (highly increases the success rate and the speed), but also robust (the performance becomes independent from the random seeds).

Besides comparing different EA versions we also conducted experiments to compare EAs with a traditional graph coloring heuristic. These experiments show that our SAW-ing EA outperforms the best heuristic we could find in the literature, DSatur.

The exact working of the SAW mechanism is still an open research issue. The plot in Figure 3 suggests that a SAW-ing EA solves the problem in two phases. In the first phase the EA is *learning* a good setting for the weights. In this phase the penalty increases a lot because of the increased weights. In the second phase the EA is *solving* the problem, exploiting the knowledge (appropriate weights) learned in the first phase. In this phase the penalty drops sharply indicating that using the right weights (the right penalty function) in the second phase the problem becomes 'easy'. This interpretation of the fitness curves is plausible. We, however, do not claim that the EA could learn universally good weights for a given graph instance. First of all, another problem solver might need other weights to solve the problem. Besides, we have applied a SAW-ing EA to a graph and recorded the weights at termination. In a following experiment we have applied an EA to the same graph using the learned weights non-adaptively, i.e. keeping them constant along the evolution. The results became *worse* than in the first run when adaptive weights were used. This suggests that reason for the success of the SAW mechanism is not that SAW-ing enables the problem solver to discover some hidden, universally good weights. This seems to contradict our interpretation that distinguishes two phases of search. Another plausible explanation of the results is based on seeing the SAW mechanism as a technique that allows the EA to shift the focus of attention, by changing the priorities given to different nodes. It is thus not the weights that is being learned, but rather the proportion between the weights. This can be perceived as an implicate problem decomposition: 'solve-these-nodes-first'. The advantage of such a (quasi) continuous shift of attention is that it finally guides the population through the search space, escaping local optima.

At the moment there are a number of evolutionary constraint handling techniques known and practiced on constraint satisfaction as well as on constrained optimization problems, [11, 26, 27]. Penalty functions embody a natural and simple way of treating constraints, but have some drawbacks. One of them is that the composition of the penalty function has a great impact on the EA performance, in the meanwhile penalty functions are mostly designed in an ad hoc manner. This implies a source of failure, as wrongly set weights may cause the EA fail to solve the given problem. The SAW mechanism eliminates this source of failure in a simple and problem independent manner.

Future research issues concern variations of the basic SAW mechanism applied here. These variations include using different Δw 's for different variables or constraints, as well as subtraction of Δw from w_i 's that belong to well instantiated variables, respectively satisfied constraints. An especially interesting application of SAW concerns constrained optimization problems, where not only a good penalty function needs to be found, but also a suitable combination of the original optimization criterion and the penalty function.

References

1. A. Blum. An $O(n^{0.4})$ -approximation algorithm for 3-coloring (and improved approximation algorithms for k -coloring). In *Proceedings of the 21st ACM Symposium on Theory of Computing*, pages 535–542, New York, 1989. ACM.
2. D. Brélaz. New methods to color vertices of a graph. *Communications of the ACM*, 22:251–256, 1979.
3. G.J. Chaitin. Register allocation and spilling via graph coloring. In *Proceedings of the ACM SIGPLAN 82 Symposium on Compiler Construction*, pages 98–105. ACM Press, 1982.
4. P. Cheeseman, B. Keneffsky, and W. M. Taylor. Where the really hard problems are. In *Proceedings of the IJCAI-91*, pages 331–337, 1991.
5. S.H. Clearwater and T. Hogg. Problem structure heuristics and scaling behavior for genetic algorithms. *Artificial Intelligence*, 81:327–347, 1996.
6. J.C. Culberson and F. Luo. Exploring the k -colorable landscape with iterated greedy. In *Second DIMACS Challenge, Discrete Mathematics and Theoretical Computer Science*. AMS, 1995. Available by <http://web.cs.ualberta.ca/~joe/>.
7. L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, 1991.
8. A.E. Eiben. Multi-parent recombination. In T. Bäck, D. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*. Institute of Physics Publishing Ltd, Bristol and Oxford University Press, New York, 1997. Section C3.3.7, to appear in the 1st supplement.
9. A.E. Eiben, P.-E. Raué, and Zs. Ruttkay. Constrained problems. In L. Chambers, editor, *Practical Handbook of Genetic Algorithms*, pages 307–365. CRC Press, 1995.
10. A.E. Eiben and Zs. Ruttkay. Self-adaptivity for constraint satisfaction: Learning penalty functions. In *Proceedings of the 3rd IEEE Conference on Evolutionary Computation*, pages 258–261. IEEE, IEEE Press, 1996.
11. A.E. Eiben and Zs. Ruttkay. Constraint satisfaction problems. In Th. Bäck, D. Fogel, and M. Michalewicz, editors, *Handbook of Evolutionary Algorithms*, pages C5.7:1–C5.7:8. IOP Publishing Ltd. and Oxford University Press, 1997.

12. A.E. Eiben and J.K. van der Hauw. Graph coloring with adaptive evolutionary algorithms. Technical Report TR-96-11, Leiden University, August 1996. also available as <http://www.wi.leidenuniv.nl/~gusz/graphcol.ps.gz>.
13. A.E. Eiben and J.K. van der Hauw. Solving 3-SAT with adaptive Genetic Algorithms. In *Proceedings of the 4th IEEE Conference on Evolutionary Computation*, pages 81–86. IEEE, IEEE Press, 1997.
14. E. Falkenauer. A new representation and operators for genetic algorithms applied to grouping problems. *Evolutionary Computation*, 2(2):123–144, 1994.
15. E. Falkenauer. Solving equal piles with the grouping genetic algorithm. In S. Forrest, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 492–497. Morgan Kaufmann, 1995.
16. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freedman and Co., 1979.
17. M.R. Garey, D.S. Johnson, and H.C. So. An application of graph coloring to printed circuit testing. *IEEE Trans. on Circuits and Systems*, CAS-23:591–599, 1976.
18. R. Hinterding, Z. Michalewicz, and A.E. Eiben. Adaptation in Evolutionary Computation: a survey. In *Proceedings of the 4th IEEE Conference on Evolutionary Computation*, pages 65–69. IEEE Service Center, 1997.
19. D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation; part II, graph coloring and number partitioning. *Operations Research*, 39(3):378–406, 1991.
20. L. Kučera. The greedy coloring is a bad probabilistic algorithm. *Journal of Algorithms*, 12:674–684, 1991.
21. P. Moris. The breakout method for escaping from local minima. In *Proceedings of the 11th National Conference on Artificial Intelligence, AAAI-93*. AAAI Press/The MIT Press, 1993.
22. B. Selman and H. Kautz. Domain-independent extensions to GSAT: Solving large structured satisfiability problems. In R. Bajcsy, editor, *Proceedings of IJCAI'93*, pages 290–295. Morgan Kaufmann, 1993.
23. J.S. Turner. Almost all k -colorable graphs are easy to color. *Journal of Algorithms*, 9:63–82, 1988.
24. G. von Laszewski. Intelligent structural operators for the k -way graph partitioning problem. In R.K. Belew and L.B. Booker, editors, *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 45–52. Morgan Kaufmann, 1991.
25. D. De Werra. An introduction to timetabling. *European Journal of Operations Research*, 19:151–162, 1985.
26. Michalewicz Z. and Michalewicz M. Pro-life versus pro-choice strategies in evolutionary computation techniques. In Palaniswami M., Attikiouzel Y., Marks R.J., Fogel D., and Fukuda T., editors, *Computational Intelligence: A Dynamic System Perspective*, pages 137–151. IEEE Press, 1995.
27. Michalewicz Z. and Schoenauer M. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1):1–32, 1996.

Applications

An Agent System for Learning Profiles in Broadcasting Applications on the Internet

Cristina Cuenca and Jean-Claude Heudin

International Institute of Multimedia
Pôle Universitaire Léonard de Vinci
92916 Paris La Défense - FRANCE
Cristina.Cuenca@devinci.fr
Jean-Claude.Heudin@devinci.fr

Abstract. In this paper we present the research results about the adequation of evolutionary algorithms and multi-agent systems to learn user's preferences during his interactions with a digital assistant. This study is done in the framework of "broadcasting" on the Internet. In our experiment, a multi-agent system with a Genetic Algorithm is used to globally optimize a user's selection of "channels" among a very large number of choices. We show that this approach could solve the problem of providing multiple optimal solutions without losing diversity.

1. Introduction

In the Internet age, users can be often overwhelmed by the very large amount of information they can find related to their search. With the basic search engines available today, based on text-retrieval, they are forced to begin querying using a few words. In most cases, this approach results in a large amount of irrelevant text that seems to contain words or concept derivations related to their search.

Facing this problem, another paradigm has been recently proposed, based on a "broadcasting" approach [Lemay97]. When you turn on a television set, you get a large number of channels to pick from: sports, news, home shopping, movies, music, etc. If you are like most of the people, you have probably got a handful of favorite channels you tune in all the time, some that you watch occasionally and others you always click past. The idea is to apply this scheme to Internet, providing "transmitters", "channels" and "tuners".

Following Marimba's terminology¹, a channel is an application or content to be distributed across the Internet. A transmitter runs in a server to deliver and maintain these channels. A tuner installed in your computer allows channel subscription and maintains, receives and manages the subscribed channels. With a tuner software

¹ URL: <http://www.marimba.com>

installed on your computer, you must be able to "tune in" to channels in the Internet in a similar way you tune in your favorite television channels.

However, even if simpler by nature, the problem remains in how to find the channels you are looking for.

In most existing approaches, you must "teach" a software about the information that is most useful to you –the music and movies you like best, for example, by storing your tastes and preferences in a form. This information is restricted to a selection of terms or a few keywords and stored in a database. Then, the system must be able to connect you directly to "channels" that fit your requirements or personal interests. This approach assumes that you know precisely what you are looking for. In addition, you must explicitly update the database if the results are irrelevant or if you simply change your points of interest.² The most recent approaches infer your preferences statistically, by making analogies between you and other peoples with similar tastes. That is the case of Firefly³, a system derived from the HOMR (Helpful On-Line Music Recommendation) project at the MIT Media Lab [Sheth93].

The aim of our project is to design a software prototype based on a multi-agent architecture which automatically learns and updates profiles in order to select channels and subscribe to programs more effectively. Transmitters and tuners are therefore considered as agents which will work and make decisions based on communication and agreements with other agents [Rosenschein94]. In particular, each tuner will include a personal digital assistant which will help the user to find the channels he is looking for. The kind of digital assistants we are interested in are self-motivated and based on an Artificial Life approach [Langton88][Heudin94].

In this paper we explore the adequation of a multi-agent system and a Genetic Algorithm on the problem of learning a user's preferences during interactions between the digital assistant and the user. We present our experiment and describe the general outlines of our solution. Then, we give our first results and discuss them. Finally, some future research directions are sketched.

2. The Experiment

The goal of the experiment is to create a digital assistant which proposes an Internet user the channels he enjoys. This assistant will learn the user's preferences during the utilization process. That means that it should be capable of dealing with this in an evolutive way, considering that a person's tastes can vary over time, temporarily or definitively. Another aspect is that a person's taste is not forced to be unique (that could be boring for him to "see" the same and only the same "channels" all the time), but he is likely to accept (and probably want) propositions of topics he doesn't know or he doesn't ask for explicitly.

² You can experience this approach at PointCast site, URL: <http://www.pointcast.com>

³ URL: <http://www.firefly.com>

2.1 Project Architecture Overview

The entire project includes two parts: the server part and the client part. The server has to provide contents to the subscribing clients and for this reason, it has to maintain client accounts and information related to its content offer. The client, by means of a multi-agent system, interacts with the user, proposes him programs, keeps track of his preferences, and negotiates the information it needs with the different servers. The general scheme is shown in Fig. 1. This paper describes only the client part of the system.

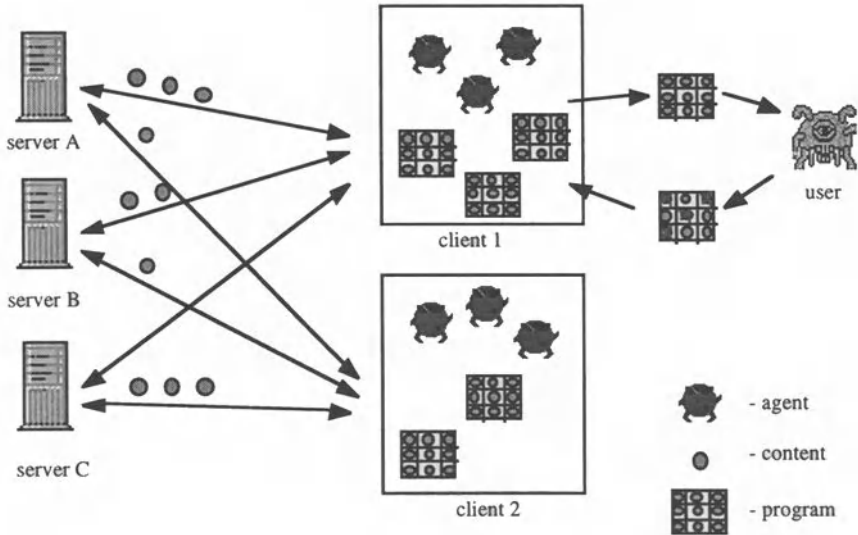


Fig. 1. The system

2.2 The Approach

Our main interest in this paper was testing the applicability of a Genetic Algorithm working in a multi-agent system as a solution to the problem of learning a user's preferences, making accurate propositions reflecting his tastes while preserving diversity in the offer. In a first stage of our work, the fact of modeling and categorizing the information to be chosen by the user was not a main objective. That's why we chose to begin facing the problem by the means of a metaphor to represent this information variety. This metaphor is explained in more details in the next section.

In a second stage, we are going to transpose this representation to its real context. It means that we are going to change the metaphor in order to work with real information, that could be video files, music files, image files, html files, etc. This process requires also a modeling of the information that is going to be processed, because the used techniques need a data quantification, as you will see later in this

paper. A start point for doing this could be the work of Sheth and Maes in *Personalized Information Filtering* [Sheth93].

2.3 The Metaphor

We use a graphical color panel as an abstraction of a recommendation of "channels". Color squares in the panel simulate different information, movies, news, videos, etc. Thus, when the system suggests a color panel (a "program"), the user can select its preferences by simply clicking in the color squares with the mouse. In our experiment, we use panels composed by 16 color squares.

Each color is encoded using the RGB format where the red, blue and green components are each represented by an integer in the range 0-255. Thus, we are working with 16 777 216 colors and $(16 \text{ squares} \times 16\,777\,216 \text{ colors}) = 268\,435\,456$ possible panels. This approach has the advantage of providing a large search space and also an enormous diversity of choice, which can be regarded also as categories (color families).

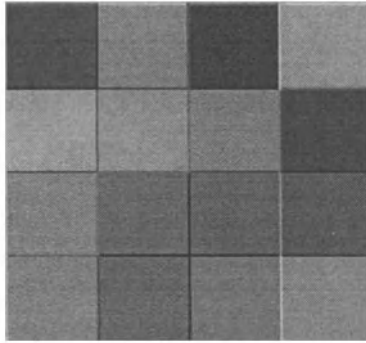


Fig. 2. A gray representation of the color panel

2.4 The Multi-Agent Architecture

We have designed a multi-agent architecture [Ferber95] based on a pyramidal model of data fusion which have been already used in the Electronic Copilot project [Gilles91]. The agents communicate with each others in a pyramidal way, according to the fact of reducing the information volume and increasing its semantics in each pyramid level. In its current implementation, the system includes five agents (cf. Fig 3).

Each agent can be purely reactive, cognitive and adaptive, depending on its objectives. Each agent's specific task will be explained in the next sections.

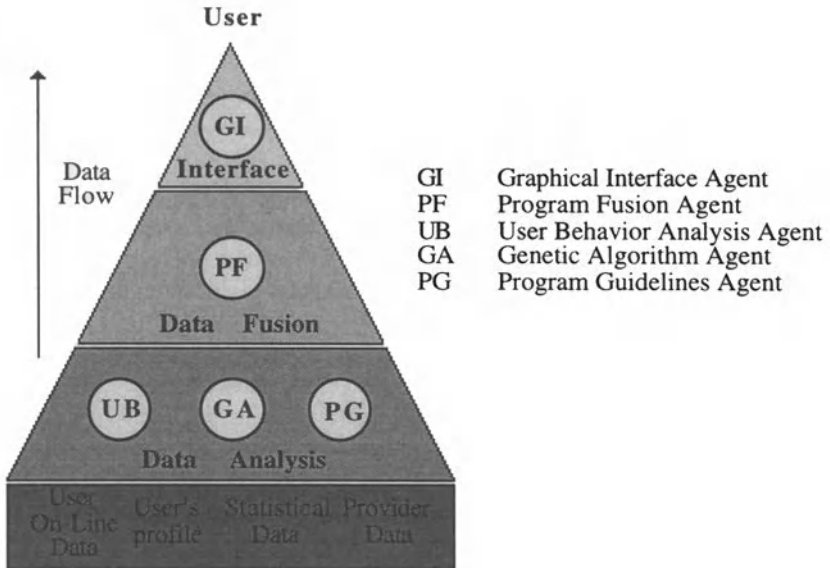


Fig. 3. The scheme

Graphical Interface Agent

It takes the "program" from the Fusion Agent and shows it to the user. It also allows the user's interaction with the panel. When a program is proposed to the user, a color panel is shown, and then, the user can select some of the colors in the panel with the mouse –to indicate the channels he wants to "tune in"(cf. figure 4).

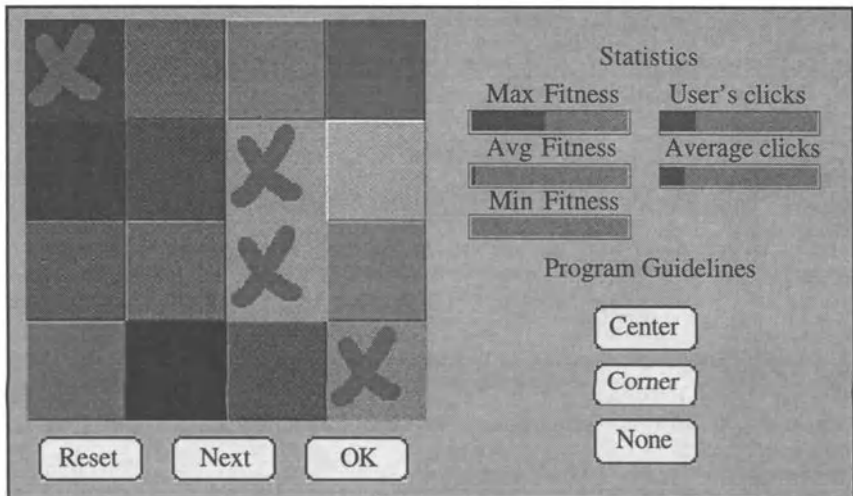


Fig. 4. A selection of colors in the graphical interface

Program Fusion Agent

It is in charge of the fusion of the Genetic Algorithm's results, and the Programs Proposition agent choice, in order to generate the "program".

User's Behavior Analysis Agent

This agent is in charge of the detection of the user's behavior, creating and maintaining a user's profile that will be used by other agents. In this stage of our work, this agent takes into account the user's number of clicks on the color panel, but in the next step it will use this information in order to identify its different states.

These states are:

- **Learning:** the user begins to use the system and start to teach it his preferences.
- **Accepting the suggestions:** The system has learnt the users preferences and proposes suitable programs.
- **Changing his mind:** The user has modified its preferences and begin to ask for new propositions. At this point, it is necessary to distinguish between a temporary change of mind and a definitive one.

It will also detect specific user's preferences, considering the clicks on specific panel positions.

Programs Guidelines Agent

A program guidelines server, charged of the generation of "programs" guidelines. A "program" is what the user is finally going to see, the selected channels presented to him in a coherent fashion. Using the metaphor, it includes the color panel the user is going to interact with but it considers also the way of reordering the colors in the panel in the most appropriate way. In this first stage, the reordering proposition agent proposes three different ways to "view the program":

- Without a specific organization, just as the proposition is done by the Genetic Algorithm.
- Putting the user's preferred colors in the center of the square.
- Putting the user's preferred colors in the corners of the square.

The behavior of this agent will be studied in a next step of our work.

Genetic Algorithm Agent

Responsible for modeling and learning the user's interests and preferences. It takes the user's profile information to make the color panel proposition evolve in order to please the user. A detailed explanation comes later.

2.5. Genetic Algorithm Agent

As we introduced previously, this is the central part of our work. This agent is responsible for the learning process and for the evolution of the program proposition.

The Algorithm

The general outline of the XY algorithm is a usual generational scheme [Heudin94]. A population Π of P individuals undergoes a succession of generations :

- **Select** two “good” and two “bad” individuals according to their relative fitness using a selection table;
- **Recombine** the two “good” selected individuals with probability C_{prob} using a single-point crossing over operator;
- **Mutate** offspring of the two parents with probability M_{prob} using an adaptive mutation operator;
- **Replace** in the population the two “bad” individuals by the offspring.

Instead of a classical iteration with a stopping criterion, and because we are not looking for an optimal and final solution (the user's interaction never ends) the algorithm develops each new generation continuously. When the user demands a new program, the algorithm selects its best individual and send it to the fusion agent in order to be displayed using the graphical interface.

The user's interactions are kept in his profile, and the algorithm can consider them at each step. The main drawback is that it must propose relatively “good” solutions in few cycles since the user cannot wait hundreds of cycles. However, it is common to have a few “not-so-bad” individuals in a population of mediocre colleagues in the first iterations. The real problem is to avoid these “good” individuals taking over a significant proportion of the finite population leading to premature convergence.

Coding Panels

We select a natural expression of the problem for coding. Each individual (a panel of 16 colors), is represented by a string of 16 32-bit integers. Each integer is composed of 8 unused bits and a red, blue, and green 8 bits components.

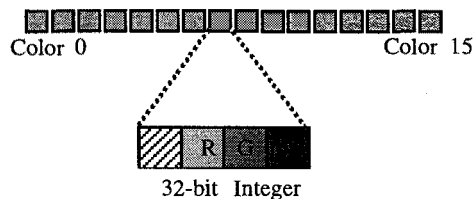


Fig. 5. The coding of an individual

Selection

Proportional selection with and without linear fitness scaling have been used. In order to reduce the stochastic errors generally associated with roulette wheel selection, we use a remainder stochastic sampling without replacement as suggested by Booker [Booker82]. Our implementation is based on a selection table and is closed to the one proposed by Goldberg [Goldberg89].

Single-point Crossover

The two selected parents are recombined using a single-point crossover operator. It is achieved by a linear combination of both parents and produces two offspring. This is simply done by choosing a random site in the interval $[0, 15]$ and swapping the right-hand parts of the parents. The probability of recombination is $Cprob = 0.7$.

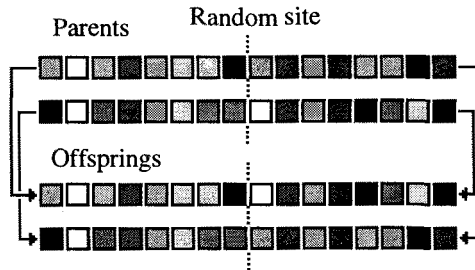


Fig. 6. Single-point crossover

Adaptive Mutation

Mutation is applied to the offspring created by crossover. Our algorithm uses an adaptive mutation operator inspired from Evolution Strategies [Rechenberg73] and real-encoded Genetic Algorithms [Eschelman93]. Each color in an individual includes its own mutation probability $Mprob$. This probability is also used for computing a deviation interval in order to add some Gaussian noise to the selected color. All colors in an individual are initialized with $Mprob = 0.4$. Then, this value is increased or decreased according to the number of times it has been selected by the user.

Fitness Function

The fitness function depends on the history of the user selections. As we explained in the previous section, each color in an individual includes its own mutation probability. When the user clicks on a color square, that means that he likes this color, so the corresponding mutation probability will be decreased. On the contrary, if the user doesn't click on a color, this value will be increased.

That allows us to use the mutation probabilities of an individual as a measure of the user preferences, in order to compute its fitness value.

$$f(x) = (1 / N) \cdot n_x$$

where x is the current panel (an individual), N is the number of color squares in the panel and n_x is the number of color squares in the panel with a mutation probability that is inferior to the default mutation probability value (Mprob). Note that the fitness function is only based on the number of selected colors. In other terms, the algorithm doesn't distinguish colors, as it works at the panel level, not at the color level.

3. First Results

3.1. Quantitative Overview

Three basic user's behavior cases have been studied. In the first case, the user has a very restricted taste, and begins selecting less than 20% of the colors squares in the panel. In the second one, he begins selecting more than 20% of the colors squares. In both cases, it is assumed that the user has made a choice of the colors he wants to see and does not change his mind during the experiment. In the third case the user makes a first choice and after a long interaction period, he suddenly changes to another color.

The three cases are illustrated by graphics which represent the population statistics (maximum, minimum and average fitness values of the population) considering two runs for each case: applying the fitness function as it is, and applying a fitness scaling, during some generations of the genetic algorithm. As the algorithm works independently of the user interaction, we computed the statistics in relation to the number of generations, but we estimated also the number of user interactions with the system during this time.

First Case: The user has a wide interest

In this case, the user is interested in two colors and all the color range between both. For example, if he likes blue and green, he clicks on all their tonalities, their lights and darks, and all the mixtures between both. In practice, this reduce the search space from 16 777 216 to about 10. Therefore, the probability for one of this colors to appear is 1/10 and we have a good chance to select 3-4 squares in each panel.

After 100 generations (during this time, the user has interacted only 7 times with the system), the maximum fitness is 0.8, and 1 after 14 user interactions. The use of a fitness scaling reduces these numbers to approximately 0.2.

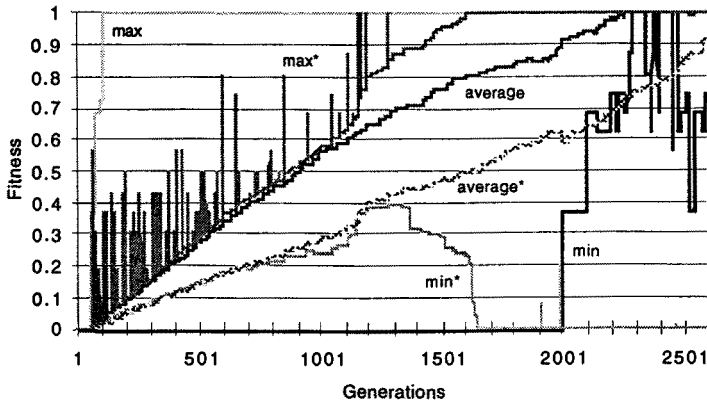


Fig. 7. First case with (*) and without fitness scaling

Second Case: The user has a precise interest

This time, the user chooses only one color, and no derivations are selected. This behavior leads to click in no more than 3 colors in a panel. This produces a slower convergence rate. In the example, after 400 generations (25 user interactions), the maximum fitness value is 0.5. After 45 user interactions, its value is 0.8. With the scaling mode, in the same conditions, this values are approximately 0.2 and 0.4.

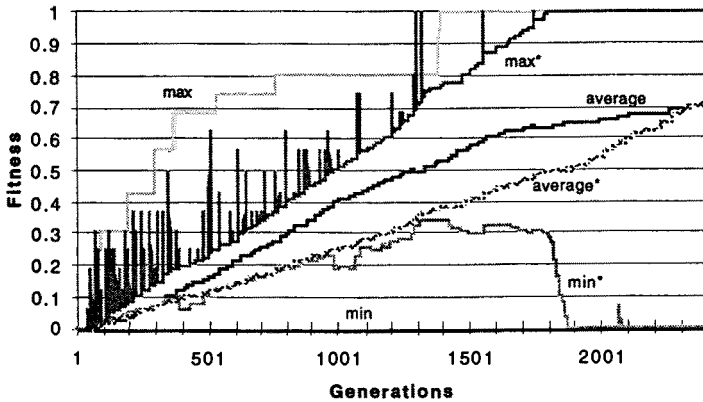


Fig. 8. Second case with (*) and without fitness scaling

Third Case: The user changes his mind

In this case, the user first chooses two colors during a long period of time, allowing the algorithm to quickly have good fitness values. Then, he decides to change his mind and to choose another color. The algorithm will not answer to this

change until a long period of time, since it will continue to propose the well adapted individuals according to the initial criteria.

In this example, after 2000 generations (200 interactions), the user tries to select a different color. The average fitness begins slowly to go down, but after 400 interactions the population is always full with the old good individuals.

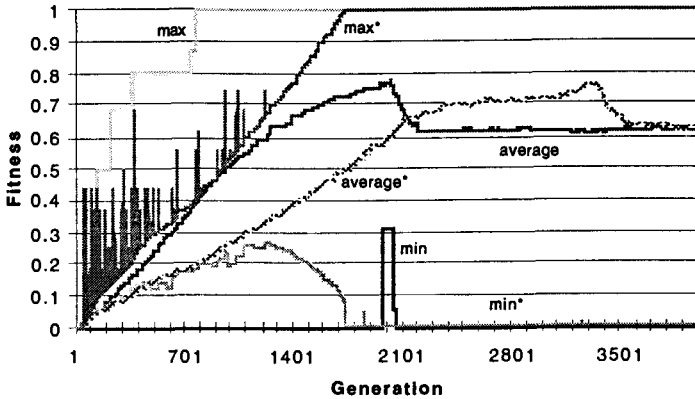


Fig. 9. Third case with (*) and without fitness scaling

3.2. Qualitative Overview

We have observed that the system learns quickly the user's preferences. That is a result of the continuous action of the genetic algorithm, which runs independently of the user's interaction, contributing to a faster convergence. Better than that, as a consequence of always displaying the best population member, the user soon receives programs that fits his taste.

However, the biggest inconvenient in those cases was the fact that, once the population has been filled with "goods individuals", it is very difficult for the algorithm to react to a user's change of mind.

On the other hand we can see that, as the algorithm computes the fitness function value for an individual if it is presented to the user, there are some members of the population which are not exposed to evaluation (in the three graphics, the min fitness value of the population remains low almost all the time). This is not really a problem since in this prototype we are working with a static population. In the real application, this will not be the case, because new individuals will arrive constantly from the servers and the system should be able to evaluate them in relation to the existing population fitness values.

4. Discussion

Our first results lend support to the hypothesis that an evolutionary approach is well-suited to the problem of generating selections of channels based on the learning of a user's preferences. After few cycles of interactions with a user, quite "good"

selections are proposed. In the long run, the population converges towards an optimal point without losing too much diversity. This diversity of channels selections is important since our goal is improvement. Attainment of the optimum is much less important in our application. We never judge a television channel by an attainment-of-the-best criterion. Perfection, that is convergence to the best, is not an issue in most walks of life.

However, we feel that the major problem with the approach we have presented here, in the actual stage of our work, is that the system does not respond correctly when the user suddenly changes his points of interest. That's why our next step is to extend the tasks of the User Behavior Analysis Agent to model the user's different states and to improve the user's profile representation.

A complementary approach to solve this problem is the use of diploidy and dominance [Goldberg89]. The redundant memory of diploidy permits multiple solutions to be carried along with only one particular solution expressed. Then, when a dramatic change in environmental condition occurs, a shift in dominance allows to express alternative solutions held in the background, allowing a rapid adaptation to the selective pressures of the changing environment.

On the other hand, the use of the metaphor is not longer suitable, because it begins to be a constraint for the experiment. A modeling of the problem considering real "documents" is one of our next steps.

5. Implementation

The experiment has been implemented as a JAVA applet [Gosling94]. This choice is motivated by the nature of the application framework and the language features (object-oriented, architecture-neutral, easy prototyping, etc.) The agents were implemented using the JAVA multi-thread mechanism.

6. Conclusions and Future Works

We have presented an application of evolutionary computing to the learning of a user's preferences. Our experiment based on a multi-agent architecture and the use of a Genetic Algorithm confirms the idea that such approach could be applied successfully in the real framework of "broadcasting" in the Internet. This preliminary results are encouraging.

Future works include the study of an algorithm that combines evolution and other learning techniques (like the Evolutionary Reinforcement Learning [Ackley91]). Our next steps are also the improvement of the user's profile representation and the User's Behavior Analysis Agent, combined with changing the metaphor in a more realistic model.

7. References

Ackley D., Littman M. (1991). *Interactions Between Learning and Evolution*.

Artificial Life II. SFI Studies in the Sciences of Complexity, vol. X, edited by C.G. Langton, C. Taylor, J.D. Farmer, S. Rasmussen, Addison-Wesley.

Booker L.B. (1982). *Intelligent Behavior as an Adaptation to the Task Environment*. Doctoral Dissertation, Technical Report n. 243, University of Michigan, Ann Arbor.

Eshelman L., Schaffer J.D. (1993). Real-coded Genetic Algorithms and Interval-Schemata. In *Foundations of Genetic Algorithms 2*, p. 187-202, edited by L.D. Whitley, Morgan Kaufmann, Los Altos.

Ferber J. (1995). *Les Systèmes Multi-Agents*, p. 15, InterEditions.

Gilles A., Lebiannic Y., Montet P., Heudin J.C. (1991). *A Parallel Multi-Expert Architecture for the Copilote Electronique*. 11th International Conference on Expert Systems and their applications, EC2, Avignon, France.

Goldberg D.A. (1989). *Genetic Algorithms in Search, Optimization & Machine Learning*. p 121-124, Addison-Wesley.

Gosling J. (1994). *The Java Language : A White paper*. Sun Microsystems.

Heudin J.C. (1994). *La Vie Artificielle*. Hermès, Paris.

Langton C.G. (1988). *Artificial Life*. *Artificial Life*. SFI Studies in the Sciences of Complexity, vol. VI, edited by C.G. Langton. Addison-Wesley.

Lemay L. (1997). *The Official Marimba Guide to Castanet*. Sams.Net.

Rechenberg I. (1973). *Evolutionstrategie : Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*. Fromman-Holzboog Verlag, Stuttgart.

Rosenschein J.S., Zlotkin G. (1994). Designing Conventions for Automated Negotiation, *AI Magazine*, Vol. 15, n. 3, p. 29, AAAI, Menlo Park.

Sheth B., Maes P. (1993). Evolving Agents for Personalized Information Filtering, *Proceedings of the Ninth Conference on Artificial Intelligence for Applications '93*, Orlando, Florida, IEEE Computer Society Press

Application of Evolutionary Algorithms to Protein Folding Prediction

A. Piccolboni and G. Mauri

Dipartimento di Scienze dell'Informazione
Università di Milano
Via Comelico 39/41, 20135 Milano, Italy

Abstract. The aim of this paper is to show how evolutionary algorithms can be applied to protein folding prediction. We start reviewing previous similar approaches, that we criticize emphasizing the key issue of representation. A new evolutionary algorithm is described, based on the notion of distance matrix representation, together with a software package that implements it. Finally, experimental results are discussed.

1 Protein folding prediction

Proteins are molecules of extraordinary relevance for live beings. They are chains of aminoacids (called also residues) that assume very rich and involved shapes *in vivo*. The prediction of protein tertiary structure (3D shape) from primary structure (sequence of aminoacids) is a daunting as well as a fundamental task in molecular biology. A large amount of experimental data is available as far as it concerns sequence, and large projects are creating huge amounts of sequence data. But to infer the biological function (the ultimate goal for molecular biology) from sequence we have to pass through tertiary structure, and how to accomplish this is still an open problem (*ab initio* prediction). Indeed, experimental resolution of structures is a difficult, costly and error-prone process. The prediction problem can be recast as the optimization of the energy function of a protein, under the assumption that an accurate enough approximation of this function is available. Indeed, according to the so-called *Anfinsen hypothesis* [AHSJ61], native conformations correspond, at a first approximation, to global minima of this function.

2 Evolutionary algorithms

Evolutionary algorithms (EAs) [Hol75, De 75, FOW66, BS93] are optimization methods based on an evolutionary metaphor that showed effective in solving difficult problems. Distinctive features of EAs are:

- a set of candidate solutions is considered at each time step instead of a single one (population);
- candidate solutions are combined to form new ones (mating operator);

- solutions can be randomly slightly modified (mutation operator);
- better solutions according to the optimization criterion (fitness) are given more reproductive trials.

These basic principles result in an overall population dynamics that can be roughly described as the spreading of good features throughout the population. This naive idea is made more precise in the so called “schema theorem” [Hol75].

According to [Gol90], the best performances can be obtained when hyperplanes in the solution space with under average energy exist (building blocks). According to the more general definition found in [Vos91], a building block is a property of solutions which is inherited with high probability by offsprings, i.e. which is (almost) preserved after crossover and mutation and is rewarding from the point of view of fitness. In the following we will refer also to genetic algorithms (GAs) which are a special case of EAs.

3 Previous work

The influence of representation on the dynamics and effectiveness of EAs has been already recognized [BBM94]. Three main representation techniques have been proposed for protein structures:

Cartesian coordinates are unsuitable for a population based algorithm, since basically identical structures (up to a roto-translation) can have completely different coordinates;

internal coordinates define aminoacid positions w.r.t. neighboring aminoacids, specifying distances and angles; this is the choice of all genetic approaches to protein folding so far;

distance geometry describes a structure by means of the matrix of all the distances between every couple of points and has been proposed as a tool for energy minimization since [NS77]; our main contribution is its joint use together with EAs.

To the best of our knowledge, all evolutionary approaches to folding prediction so far have been based on an internal coordinate representation. It is straightforward to show that relevant structural features can not be described as hyperplanes under this approach.

Schulze-Kremer [SK93, SK95] defines a real-coded GA in internal coordinate space and a simplified energy function, but fails *ab initio* prediction for a test protein. His algorithm proves useful for side chain placement.

Unger and Moulton [UM93] compare a GA against Monte Carlo methods using an idealized 2D lattice model and simplified internal coordinates. Large performance gains over Monte-Carlo are achieved but no comparison is possible with real proteins (Patton *et al.* [PPG95] report an improvement to this approach and we will compare our results to theirs).

Dandekar and Argos [DA94] use a standard GA with an heuristic, heavily tailored fitness and an internal coordinate discretized representation. Results are encouraging, but the generality of the method is questionable.

Herrmann and Suhai [HS95] use a standard genetic algorithm in internal coordinate space together with local search and a detailed model. It proved interesting only for very small structures.

A simple observation against internal coordinate representation is the following: typical force fields are a sum of terms like relaxation distances or relaxation angles that are convex functions of pairwise distances. It is likely that structures minimizing these relaxation terms will have a lower energy and optimal structures have a lot of relaxation terms close to zero. So distances and angles can act as building blocks for genetic search. But internal coordinates include only distances between neighboring residues in the sequence. The distance between an arbitrary pair of residues can be calculated only using a complex formula involving coordinates of all the residues that appear between the two in the sequence, i.e., in the worst case, the whole representation. The same is true for angles. It is very difficult, thus, to describe useful schemas in internal coordinate space and guarantee some minimal properties such as *stability* [Vos91], *low epistasis* [BBM93] and others that can not guarantee the success of a GA but are believed to be key ingredients of it [Gol90]. On the contrary, we show that some of these properties hold for suitable genetic operators in distance matrix space.

4 Energy function

We tried to keep our model as simple as possible. We model each residue as a unique point and according to [Dil85] we consider the hydrophobic/hydrophilic interaction as the dominant force that drives folding. Under this assumption, protein instances can be seen as sequences of either hydrophobic or hydrophilic aminoacids. Thus our energy function E_{tot} is made up of three terms only

$$E_{tot} = E_{rep} + E_{chn} + E_{hyd}$$

that we will describe briefly. E_{rep} is a “solid body” constraint penalty term, that prevents two residues from occupying the same place. Namely we have

$$E_{rep} = k_{rep} \sum_{i=1}^{N-2} \sum_{j=i+2}^N g(d_{ij})$$

where N is the length of the protein, k_{rep} a suitable constant, d_{ij} is the distance between residues i and j and g is defined as

$$g(x) = \begin{cases} x - 2 + \frac{1}{x} & 0 < x \leq 1 \\ 0 & x > 1 \end{cases}$$

E_{chn} is a chain constraint penalty term, that forces neighboring residues in the sequence to lay spatially close, whose form is

$$E_{chn} = k_{chn} \sum_{i=1}^{N-1} (d_{i,i+1} - 1)^2$$

where k_{chn} is a constant. Finally, E_{hyd} is an hydrophobic interaction term, rewarding closeness between hydrophobic aminoacids and is defined as

$$E_{hyd} = k_{hyd} \sum_{i=1}^{N-2} \sum_{j=i+2}^N h(d_{ij})$$

where k_{hyd} is a constant and

$$h(x) = \begin{cases} \log((x-1)^2) + \frac{1}{e} & \text{if residue } i \text{ and } j \text{ are both hydrophobic} \\ 0 & \text{otherwise} \end{cases}$$

Although the exact form of the energy is subject to wide variations [Neu93], this energy function models at least qualitatively the most important structural properties of proteins.

From an evolutionary algorithm point of view this function satisfies the ideal condition of *no epistasis* [BBM93], since a change in a variable always produces the same change in energy, despite the values of all other variables. This is the main advantage of a distance based representation.

5 Distance matrix representation

We describe a different approach to folding prediction with EAs. The solution space is the set of aminoacid distance matrices, that is the set of $N \times N$ symmetric, zero diagonal matrices with positive entries representing pairwise distances between aminoacids. Unfortunately, it turns out that such a representation describes a superset of possible configurations. To cope with this fact and, in general, to deal with such a kind of representation we need some concepts belonging to *distance geometry* [HKC83].

Definition 1 A distance matrix $\mathbf{D} = \{d_{ij}\}$ is embeddable in $\mathbf{R}^n$ if there exist a set of N points in $\mathbf{R}^n$, $\mathbf{C} = \{\mathbf{c}_i\}$ s.t.

$$||\mathbf{c}_i - \mathbf{c}_j|| = d_{ij}$$

and such a $\mathbf{C}$ is called an embedding of $\mathbf{D}$.

We use the same notation for the set of points $\mathbf{C} = \{\mathbf{c}_i\}$ and for the $n \times N$ matrix with $\{\mathbf{c}_i\}$ as columns.

Definition 2 Given a matrix $\mathbf{D}$, its Gram matrix is $\mathbf{M} = \{m_{ij}\}$, with

$$m_{ij} = d_{i1}^2 + d_{j1}^2 - d_{ij}^2$$

Definition 3 Given a set of points $\mathbf{C}$, its metric matrix is $\mathbf{C}^T \mathbf{C}$.

Theorem 1. In an inner product space, if $\mathbf{C}$ is an embedding of $\mathbf{D}$ and $\mathbf{M}$ is the Gram matrix of $\mathbf{D}$, then $\mathbf{M}$ is equal to the metric matrix of $\mathbf{C}$.

In our context this equality will always hold. From an algorithmic point of view, $\mathbf{C}$ is computable from $\mathbf{M}$ by Cholesky factorization [Van92].

Theorem 2. *A matrix $\mathbf{D}$ is embeddable in $\mathbf{R}^n$ iff its Gram matrix $\mathbf{M}$ is positive semidefinite of rank at most n .*

We observe that, since $\mathbf{M}$ is $N \times N$ its rank can be at most N . We have obtained thus a very elegant way to define our solution space: it is the set of symmetric, zero diagonal matrices with positive entries whose metric matrix is positive semidefinite of rank at most 3.

6 EA specification

According to [MS96], among different constraint handling strategies for EAs we explored these two possibilities:

- whenever an unfeasible solution is produced “repair” it, i.e. find a feasible solution “close” to the unfeasible one (*repair strategy*);
- admit unfeasible individuals in the population, but penalize them adding a suitable term to the energy (*penalize strategy*).

6.1 The repair algorithm

We are given a distance matrix $\mathbf{D}$ which is, for some reason, unfeasible and we want to find a feasible solution “close” to it, where “close” is to be specified. First of all, we may safely assume that $\mathbf{D}$ is symmetric, zero diagonal with positive entries, since, as we will see it, is easy to enforce such properties in the population, whatever kind of GA we are defining.

Next, we turn our attention to positive semidefiniteness. We can test this property in polynomial time by evaluating the smallest eigenvalue (it is positive iff the matrix is positive semidefinite), but in case it isn’t verified there’s nothing we can do (apart from rejecting the solution). This shortcoming is common to other similar algorithms in the literature [HKC83] and we are currently investigating this issue. Our guess is that without positive semidefiniteness the problem of embedding a distance matrix could be much harder.

Let us suppose we are given a symmetric, zero diagonal matrix $\mathbf{D}$ with positive entries whose metric matrix $\mathbf{M}$ is positive semidefinite of rank $n > 3$, so that the condition on rank only is not satisfied. The repair algorithm proceeds as follows (it is a modification and generalization of the one in [LLR95]):

1. find a coordinate set $\mathbf{C}$ in $\mathbf{R}^n$, $n \leq N$ by Cholesky factorization;
2. compute the projection of $\mathbf{C}$ (with distance matrix $\mathbf{D}' = \{d'_{ij}\}$) onto a random hyperplane P through the origin; call it $\mathbf{C}'$;
3. multiply $\mathbf{C}'$ element-wise by $\sqrt{n/3}$ (in the following we will consider, for the sake of generality, an m -dimensional hyperplane and a multiplicative factor of $\sqrt{n/m}$).

We have the following

Theorem 3. *With probability greater than $1/e$, for each i, j*

$$\left| \frac{d_{ij}^2 - d_{ij}'^2}{d_{ij}^2} \right| \leq O(\sqrt{n \log N})$$

Proof. (It is a generalization of the one in [Tre96]) Let us introduce the random variable (call it *distortion*)

$$X = \left| \frac{d_{ij}^2 - d_{ij}'^2}{d_{ij}^2} \right|$$

and the quantities

$$y_k = \|c'_{ik} - c'_{jk}\|^2$$

(we can temporarily forget about the dependence of y_k from i and j since what we are going to prove with y_k 's is true for every i, j). Without loss of generality we may assume that P is a coordinate hyperplane (whose dimension is m to be more general, but we are interested in $m = 3$). This is because we are interested in properties of distances, which are invariant under rotation. Since

$$c'_{ik} = \sqrt{\frac{n}{m}} c_{ih}$$

for some h , depending only on P , we have that

$$d_{ij}'^2 = \frac{n}{m} \sum_{k=1}^m y_{I_k}$$

whereas

$$d_{ij}^2 = \sum_{k=1}^n y_k$$

where I_k are the indices of dimensions parallel to P , so that $d_{ij}'^2$ is a random variable which is the sum of a subset of size m of $\{y_k\}$ multiplied by n/m . Let

$$Y_k = \frac{\frac{n}{m} y_{I_k} - \frac{1}{m} \sum_{i=1}^n y_i}{\sum_{y=1}^n y_i}$$

so that

$$X = \left| \sum_K = 1^m Y_k \right|$$

Straightforward calculations show that $E[Y_k] = 0$ and $-1 \leq Y_k \leq 1$. We are ready to apply Hoeffding inequality [Hoe63] to obtain

$$P \left(X \leq \frac{1}{2} \sqrt{n(\log 2 + \log(N^2 + 1))} \right) \geq 1 - \frac{1}{N^2 + 1}$$

The probability of the same bound to hold for all N^2 distances at once is greater or equal than $(1 - \frac{1}{N^2+1})^{N^2}$ which is greater than $1/e$. Q.E.D.

6.2 The penalization term

We would like to be able to measure the “level of unfeasibility” of a solution $\mathbf{D}$. As with the repair algorithm, we take for granted that $\mathbf{D}$ has positive entries and zero diagonal. Next, in case $\mathbf{D}$ metric matrix is not positive semidefinite, we assign $\mathbf{D}$ a conventional, very high penalization. Thus we have to deal only with positive semidefinite matrices. If we measure the difference between two configurations $\mathbf{C}$ and $\mathbf{C}'$ (the first one of higher dimension than the second) by the Frobenius norm of their difference

$$\|\mathbf{C} - \mathbf{C}'\|_F = \sqrt{\sum_{ij}^N c_{ij} - c'_{ij}}$$

we have the following theorem (adapted from [HKC83])

Theorem 4. *If $\mathbf{C}$ has rank n the matrix $\mathbf{C}'$ of rank $m \leq n$ that minimizes*

$$\|\mathbf{C} - \mathbf{C}'\|_F$$

is obtained as follows

1. *compute $\mathbf{C}$ metric matrix $\mathbf{M}$;*
2. *decompose $\mathbf{M}$ as $\mathbf{M} = \mathbf{Y}^T \mathbf{\Lambda} \mathbf{Y}$, where $\mathbf{Y}$ is unitary and $\mathbf{\Lambda}$ is $\text{Diag}(\lambda_1, \dots, \lambda_N)$ with $\lambda_1, \dots, \lambda_N$ the eigenvalues of $\mathbf{M}$ in decreasing order;*
3. *finally, $\mathbf{C}' = \text{Diag}(\lambda_1^{\frac{1}{2}}, \dots, \lambda_m^{\frac{1}{2}}, 0, \dots, 0) \mathbf{Y}$*

Moreover, we observe that the minimum of $\|\mathbf{C} - \mathbf{C}'\|_F$ so attained is

$$\sqrt{\sum_{i=m+1}^N \lambda_i}$$

Therefore, this is (with $m = 3$) a good candidate as a penalization term. Since we would like it to be invariant to scale changes, we normalized it to obtain

$$k \frac{\sqrt{\sum_{i=m+1}^N \lambda_i}}{\sqrt{\sum_{i=1}^N \lambda_i}}$$

where k is a weighting factor to be described later on.

6.3 Genetic operators

The EA can be further specified with the definition of genetic operators. We defined and experimented with a number of different recombination operators and we studied the stability of schemata, the feasibility of solutions produced by each of them and their behavior w.r.t. the repair algorithm. The following definitions are a generalization of the ones in [Vos91].

Definition 4 A schema is a subset of the solution space.

Definition 5 A schema H is said to be stable under some genetic operator G if whenever G is applied to individuals all belonging to H then every generated offspring belongs to H .

Every operator is supposed to have as input one or two zero diagonal symmetric matrices (parents) with positive entries and outputs, as it is easy to check, a matrix with the same properties.

The first genetic operator is the customary *uniform crossover*, i.e. let $\mathbf{D}'$ and $\mathbf{D}''$ be parents of $\mathbf{D}$, then for each i, j independently

$$P(d_{ij} = d'_{ij}) = \frac{1}{2} \quad P(d_{ij} = d''_{ij}) = \frac{1}{2}$$

It is easy to show that every coordinate hyperplane in matrix space is stable under this operator. The resulting matrix is not guaranteed to have a positive semidefinite metric matrix $\mathbf{M}$, so that the new individual has to be rejected or penalized, depending on constraint handling strategy. Moreover, even when $\mathbf{M}$ is positive semidefinite, it can have full rank also when parents have rank 3, so that the upper bound on distortion in theorem 3 can be rewritten as

$$\left| \frac{d_{ij}^2 - d_{ij}''^2}{d_{ij}^2} \right| \leq O(\sqrt{N \log N})$$

This can be better evaluated considering that, in general, in a feasible structure $\mathbf{D}$ $1 \leq d_{ij} \leq N$ and in a maximally compact structure (and low energy structures are usually very compact) $1 \leq d_{ij} \leq O(N^{\frac{1}{3}})$. We found experimentally that using a similar operator but with probability biased toward one of the parents (19/20) positive semidefiniteness is often preserved and distortion is lower. We call this biased operator *graft crossover*. See table 1 for an experimental analysis of this operator.

The third operator (called *arithmetic crossover*) is a convex combination of element-wise squared matrices i.e.

$$d_{ij}^2 = \alpha d_{ij}'^2 + (1 - \alpha) d_{ij}''^2$$

with α uniformly distributed between 0 and 1. It is straightforward to show that every convex set in matrix space is stable under this operator, so that it has a very rich set of stable schemata, that includes, for example, coordinate hyperplanes and general hyperplanes. As to what concerns feasibility, we have the following

Theorem 5. [HKC83] *The set of all matrices of squared distances is convex in zero-diagonal matrix space. Furthermore, the convex combination of two matrices of squared distances embeddable in k and l dimensions respectively is embeddable in at most $k + l$ dimensions.*

This means that when adopting the repair strategy we are guaranteed that the generated individual has a positive semidefinite metric matrix of rank at most 6, since parents are guaranteed to be feasible. This turns out in a much lower distortion, as we can see specializing theorem 3 to this case as follows

$$\left| \frac{d_{ij}^2 - d_{ij}^{\prime 2}}{d_{ij}^2} \right| \leq O(\sqrt{\log N})$$

A fourth operator (*block crossover*) tries to emulate the effects of multi-point crossover applied to internal coordinates, as in previous approaches cited before. It is akin to uniform crossover with submatrices instead of elements as basic entities. It works as follows

1. The set of aminoacids is partitioned in intervals $I_1, \dots, I_k$, according to their order in the aminoacidic sequence;
2. Both parents' distance matrices ($\mathbf{D}'$ and $\mathbf{D}''$) are partitioned in submatrices

$$\mathbf{D}'_{ij} = \{d'_{hk}\}_{h \in I_i, k \in I_j} \quad \mathbf{D}''_{ij} = \{d''_{hk}\}_{h \in I_i, k \in I_j}$$

3. For each i, j choose a parent $p(i, j)$ with probability $1/2$;
4. Define a new solution as

$$d_{hk} = \{d_{hk}^{p(ij)}\}_{h \in I_i, k \in I_j}$$

The analysis of this operator is basically the same as the one for uniform crossover.

In table 1 the four crossover operators are analyzed experimentally on a sample population of 27 residue long individuals and size 1000, generated according to the strategy explained in 6.3. For 1000 individuals generated by crossover we report the ratio of individuals with positive semidefinite metric matrix, the ratio of individuals with energy lower, intermediate or higher than the two parents.

crossover	pos. sem.	lower	interm.	higher
arithmetic	100.0	23.3	49.5	27.2
block	45.4	7.3	19.2	73.5
graft	65.6	6.3	17.6	76.1
uniform	33.1	1.2	5.3	93.5

Table 1. Statistical analysis of crossover operators

The mutation operator is a classic creep operator that randomly chooses a variable and modifies it with the addition of a small gaussianly distributed

amount ε , preserving distance symmetry. The positivity of entries is preserved by suitable truncation of the probability distribution. Since the corresponding creep in Gram matrix eigenvalues is $O(\varepsilon)$ [Van92], mutation can be tuned so that feasibility is almost preserved. Small negative eigenvalues are rounded to zero.

Further details of the EA can be summarized as follows:

- we use a steady-state algorithm (offsprings are inserted in the population as soon as they are created);
- reproduction and replacement candidates are selected out of a small pool (*tournament selection*), according to fitness and diversity criteria (individual less fit and more similar to a new offspring are more likely to be replaced).
- each individual in the initial population is generated in the following way: the first aminoacid is at the origin; the position of the i th aminoacid is chosen uniformly at random on the unit sphere centered around the $(i-1)$ th aminoacid.

7 Implementation issues

The simulation program has been coded in C++, using the mathematical library Lapack++ and the genetic algorithm library GAlib. A graphical front end for simulation control and basic data analysis has been written in Perl and Tcl/Tk and using the visualization program Rasmol. Simulation have been run on a IBM-compatible with Pentium processor under Linux operating system, but a porting to a four processor Silicon Graphics Origin 2000 server is under way.

8 Simulation results

We compare our results to those found in [PPG95], but the reader should be warned that the comparison is somewhat unfair, since in that paper a lattice model is used, so that a residue neighborhood can accommodate only 6 residues, instead of 12 [CS92] as in our continuous model. Moreover, in the same paper the energy is the opposite of the number of hydrophobic residue pairs at unit distance, so we tried to evaluate our structures using the same criterion, but introducing a little tolerance.

In table 2 we summarize our results on three test cases taken from [PPG95] using the *repair strategy*, comparing different crossover operators and tournament sizes. Column C1 reports the number of hydrophobic pairs at distance 1 ± 0.2 , C2 at distance 1 ± 0.1 . The corresponding results in [PPG95] are 15 for P27-4, 11 for P27-6 and 13 for P27-7.

In figure 1 the best structure obtained for P27-4 is shown. The compact hydrophobic core is very protein-like. The simulations ran with a population of 1000 individuals and the algorithm stopped after the generation of 200,000

		Tournament size 10			Tournament size 4		
sequence	crossover	C1	C2	energy	C1	C2	energy
P27-4	arithmetic	20	20	24.211	18	15	19.655
	block	35	28	-23.438	31	27	-19.484
	graft	35	31	-23.051	36	30	-28.068
	uniform	22	19	22.071	25	22	5.267
P27-6	arithmetic	10	7	16.592	10	8	15.730
	block	17	15	-4.172	17	15	-4.511
	graft	13	10	0.961	16	11	0.459
	uniform	12	8	6.580	13	9	8.270
P27-7	arithmetic	20	15	-2.190	16	12	12.719
	block	25	21	-5.368	21	19	-4.231
	graft	23	19	-1.343	21	17	2.959
	uniform	22	19	-4.640	15	12	13.631

Table 2. Simulation results for “repair” strategy

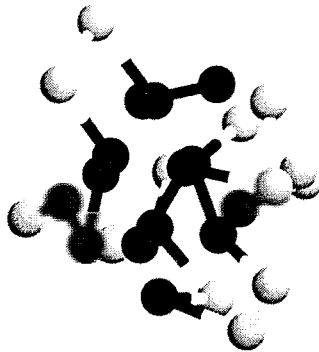


Fig. 1. Best structure for P27-4

offsprings, taking about 21 minutes. The probability of applying the crossover and mutation operators were set to 0.9 and 0.5 respectively.

Simulation with the *penalize strategy* gave unsatisfactory results. We tried different penalization strategies, with fixed or time varying weight factor. No run stopped producing a feasible solution. This can be explained noting that all genetic operators described tend to increase the rank of the metric matrix of new individuals.

9 Discussion and further work

The ultimate test for any protein folding prediction algorithm is the prediction of experimental structures of biological sequences. Of course the EA described in the present work is not ready for such a task, at least because the very simple model prevents accurate predictions. Anyway it compares favorably with previous approaches and can be straightforwardly extended to more complex and realistic models, something that lattice model oriented algorithms can't. The experimental analysis of this algorithm is clearly in its preliminary stage and we plan to test longer sequences as soon as possible. Moreover we would like to clarify the relative strengths and weaknesses of different crossover operators. A first distinction can be made between *arithmetic crossover* and the others, since this operator has some nice property w.r.t rank constraints but doesn't explore thoroughly the search space. This observation should be made more precise and supported with analytic or experimental evidence.

Acknowledgments

This work has been partly supported by MURST project "Efficienza di Algoritmi e Progetto di Sistemi Informativi" and by CNR grant 97.02399.CT12.

References

- [AHSJ61] C. B. Anfinsen, E. Haber, M. Sela, and F. H. White Jr. The kinetics of formation of native ribonuclease during oxidation of the reduced polypeptide chain. In *Proceedings of the National Academy of Sciences of the U. S. A.*, volume 47, pages 1309–1314, 1961.
- [BBM93] David Beasley, David R. Bull, and Ralph R. Martin. An overview of genetic algorithms: Part 2, research topics. *University Computing*, 15(4):170–181, 1993.
- [BBM94] D. Beasley, D. R. Bull, and R. R. Martin. Complexity reduction using expansive coding. In T. C. Fogarty, editor, *Evolutionary computing: aish workshop: selected papers*, page 304, Leeds, UK, april 1994.
- [BS93] Th. Bäck and Hans-Paul Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1–23, 1993.
- [CS92] J. H. Conway and N. J. A. Sloane. *Sphere packings, lattices and groups*. Number 290 in Grundlehren der mathematischen Wissenschaften. A series of comprehensive studies in mathematics. Springer-Verlag, New York, second edition, 1992.
- [DA94] T. Dandekar and P. Argos. Folding the main chain of small proteins with the genetic algorithm. *Journal of Molecular Biology*, 236:844–861, 1994.
- [De 75] Kenneth De Jong. *An analysis of the behaviour of a class of genetic adaptive systems*. PhD thesis, University of Michigan, 1975.
- [Dil85] Ken A. Dill. Dominant forces in protein folding. *Biochemistry*, 24:1501, 1985.

- [FOW66] Lawrence J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial intelligence through simulated evolution*. Wiley, New York, 1966.
- [Gol90] D. E. Goldberg, editor. *Genetic Algorithm in search, optimization and machine learning*. Addison-Wesley, 1990.
- [HKC83] T. F. Havel, I. D. Kuntz, and G. M. Crippen. The theory and practice of distance geometry. *Bulletin of Mathematical Biology*, 45(5):665–720, 1983.
- [Hoe63] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, (58):13–30, 1963.
- [Hol75] John H. Holland, editor. *Adaptation in Natural and Artificial Systems*. MIT Press, 1975.
- [HS95] Frank Herrmann and Sándor Suhai. Energy minimization of peptide analogues using genetic algorithm. *Journal of Computational Chemistry*, 16(11):1434–1444, 1995.
- [LLR95] Nathan Linial, Eran London, and Yuri Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995.
- [MS96] Zbigniew Michalewicz and Marc Schoenauer. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary computation*, 4(1):1–32, 1996.
- [Neu93] Arnold Neumaier. Molecular modeling of proteins and mathematical prediction of protein structure. *SIAM Rev.*, pages 407–460., 1993.
- [NS77] G. Nemethy and H. Scheraga. Protein folding. *Quarterly reviews in Biophysics*, 10:239–352, 1977.
- [PPG95] A. Patton, W. Punch, and E. Goodman. A standard ga approach to native protein conformation prediction. In L. Eshelman, editor, *Proc. Sixth Int. Conf. Gen. Algo.*, volume 574. Morgan Kaufmann, 1995.
- [SK93] Steffen Schulze-Kremer. Genetic algorithms for protein tertiary structure prediction. In P. B. Brazdil, editor, *Machine Learning: ECML-93*. Springer-Verlag, 1993.
- [SK95] Steffen Schulze-Kremer. Genetic algorithms and protein folding. <http://www.techfak.uni-bielefeld.de/bcd/Curric/ProtEn/contents.html>, June 1995.
- [Tre96] Luca Trevisan. When hamming meets euclid: The approximability of geometric TSP and MST. In *Proceedings of the 28th Symposium on the Theory of Computing*, 1996.
- [UM93] R. Unger and J. Moulton. Genetic algorithms for protein folding simulations. *Journal of Molecular Biology*, 231:75–81, 1993.
- [Van92] Charles Van Loan. A survey of matrix computations. In A. H. Rinnooy Kan, J. K. Lenstra, and E. G. Coffman, editors, *Handbooks in operations research and management science*, volume 3: computing, chapter 6, pages 247–321. Elsevier Science Publisher, 1992.
- [Vos91] Michael D. Vose. Generalizing the notion of schema in genetic algorithms. *Artificial Intelligence*, 50:385–396, 1991.

Telephone Network Traffic Overloading Diagnosis and Evolutionary Computation Techniques

Isabelle Servet¹, Louise Travé-Massuyès¹ and Daniel Stern²

¹ LAAS/CNRS 7,avenue Colonel Roche F-31077 Toulouse Cedex France

² CNET 38-40 ,rue du Général Leclerc F-92131 Issy-les-Moulineaux France

Abstract. Traffic supervision in telephone networks is a task which needs to determine streams responsible for call losses in a network by comparing their traffic values to nominal values. However, stream traffic values are not measured by the on-line data acquisition system and, hence, have to be computed. We perform this computation by inverting an approximate knowledge based model of stream propagation in circuit-switched networks. This inversion is computed thanks to three evolutionary computation techniques (multiple restart hill-climbing, population-based incremental learning and genetic algorithms) for which both a binary version and a real variant have been experimented with several fitness measures. The final results first point out how the fitness measure choice can impact on their quality. They also show that, in this case, real variants of the algorithms give significantly better results than binary ones.

1 Introduction

Due to the increasing size of telephone networks and to their high connectivity (several organs are shared by several streams), the task of networks managers becomes more and more complex. Determining which streams are responsible for an overloaded situation is difficult, especially as the disturbances may propagate within the network in a very short time frame.

Real-time network supervision does not require accurate values (approximate values are sufficient enough to incriminate the streams responsible for overloads) but it needs to compute them rapidly.

Thus, the stream propagation computation cannot be done using a call by call simulation program because of the extremely long computer running time it requires. This is the reason why macroscopic methods are interesting [STMS96b]. The simplest approach for network analysis is the one-moment method where offered traffic is only characterized by its mean. The model briefly presented here uses this method and has been developed under classical assumptions of single moment traffic modeling methods [Pas84]. It is dedicated to compute the whole traffic losses and offered traffic for each communication organ, for each stream and the lost and offered traffic for each stream at each network element (i.e. each communication organ), all this given the network topology and the offered traffic to each stream.

Then, as our goal is to know the values of the offered traffic to each stream, given the traffic losses and the offered traffic to each network element which are the only measures available, we have to *invert* the previous model thanks to optimization methods.

Our model is indeed an highly non linear and non differentiable function so that neither a direct inversion nor classical optimization methods are appropriate. However, evolutionary computation techniques are the subject of many researches, given their high potential for solving complex optimization problems. In the third part of this paper, three of these methods are presented :

- *multiple restart hill-climbing*, an heuristic and, then, very simple method,
- *population based incremental learning* which can be seen as an abstraction of genetic algorithms,
- *a standard genetic algorithm*, a well-known method to solve complex optimization problems.

These methods have been chosen because of their simplicity and, then, rapidity. They initially have been developed with binary variables but a real variant of these methods is proposed too.

Comparative results of the model inversion are given in the fourth part of this paper.

2 Stream propagation model

This model is based on the concept of **blocking** network elements and the search of such elements in the network.

Definition 1 *A network element (NE): exchange or circuit group is said to be blocking when, according to the network structure, it is likely to experiment traffic loss.*

It can be divided in 3 main stages [STMS96a]:

- *Blocking organs search* which is performed thanks to 2 qualitative rules: the overdimensioned capacities rule (independent of the offered traffic values) and the loss rate rule which is performed using **Erlang's formula** given in the definition below.

Definition 2 *Consider an organ whose capacity is N and which routes a traffic stream of intensity A . Then, the probability that the N circuits are busy is: $E[N, A] = \frac{A^N/N!}{1 + \dots + A^N/N!}$*

- *Blocking NEs ordering*. This is especially important when overflow from a circuit group to an other is allowed, the offered traffic on the latter depending on the amount of lost traffic on the former.
- *Traffic loss computation* for each blocking NE, which is computed thanks to a *resource availability backpropagation* algorithm [STMS96a]

3 Evolutionary algorithms

Evolutionary computation is an umbrella term used to describe computer-based problem solving systems which use models of evolutionary processes as key elements in their design and implementation [HB94].

Their common base is the simulation of the evolution of *individuals* via processes of *selection*, *mutation* or *reproduction* which depend on the performance, computed thanks to a *fitness measure*, of the individuals.

Hence, before an evolutionary algorithm can be run, a suitable representation for each structure must be determined. It is assumed [BBM93a] that a potential solution to a problem may be represented as a set of parameters, known as *genes* that are joined together to form a string of values, often called a *chromosome*. Besides, these algorithms are often used for function optimization [Bal95], their effectiveness strongly depending on the particularities of the function.

3.1 Multiple restart hill-climbing (MRH)

Binary version This iterative method, although very simple, sometimes leads to better results than genetic algorithms [Bal94]. Its simplest algorithm is:

```

 $V \leftarrow$  randomly generated chromosome
 $BEST \leftarrow V$ 
Loop NB.ITERATIONS
   $MUT\_POSITION \leftarrow$  random bit position
   $N \leftarrow$  mutation of  $V$  in  $MUT\_POSITION$ 
  If  $N$  is better than  $BEST$  Then  $BEST, V \leftarrow N$ 

```

Real-coded variant First, the chromosome V , which is randomly generated, is a real vector. Hence, when optimizing a function, each component of this chromosome represents the value of the corresponding variable.

Then, $MUT\_POSITION$ corresponds to the range of the variable which has to undergo a mutation.

Therefore, a mutation of the i^{th} component $V[i]$ of the chromosome V consists, like in genetic algorithms real-coded variants [BBM93b], in adding to $V[i]$ a small random real (for instance a real in $[-V[i]/10, V[i]/10]$).

3.2 Population-based incremental learning (PBIL)

Binary version Contrary to the previous method, this one does not consider a single chromosome but a set of nb chromosomes called a *generation*.

The object of this method is to create a real-valued *probability vector* P which is dedicated to reveal high quality chromosomes [Bal95]. Hence, P is updated at each generation and the chromosomes of each generation are created according

to its values: each component $V[i]$ of each chromosome V is determined as follow :

Given $random_i$ a random value in $[0,1]$,

If $random_i > P[i]$ then $V[i] \leftarrow 0$ else $V[i] \leftarrow 1$

Therefore, $P[i]$ represents the probability to have '1' at this position in the best chromosome ; hence, $P[i]$ is, first, taken equal to 0.5 and P evolution, from a generation to an other, is computed thanks to competitive learning mechanisms[Bal94].

Real-coded variant First, we assume that each gene (i.e. each variable in the case of function optimization) has a continuous and bounded definition set. Hence, we consider that each component $V[i]$ of a chromosome V belongs to an interval $[Low_i, Up_i]$.

Then, each component $P[i]$ of P gives the probability of the i^{th} variable to be greater than $\frac{Low_i + Up_i}{2}$.

The probability vector updating is similar to P updating in the binary version but a step for intervals $[Low_i, Up_i]$ updating is added: when, for a given iteration, the i^{th} component of P is such that:

- $P[i] \geq 0.9$, the i^{th} component of the best chromosome is assumed to be greater than $(Low_i + Up_i)/2$. Then, Low_i is updated to $(Low_i + Up_i)/2$ and $P[i]$ is reinitialized at 0.5.
- $P[i] \leq 0.1$, the i^{th} component of the best chromosome is assumed to be lower than $(Low_i + Up_i)/2$. Then, Up_i is updated to $(Low_i + Up_i)/2$ and $P[i]$ is reinitialized at 0.5.

P reinitialization is due to the fact that the change of a bound of the interval $[Low_i, Up_i]$ comes to start again the algorithm with a new definition set.

3.3 A genetic algorithm (GA)

Binary version In this method, an offspring can be the result of gene mutation, as in the 2 previous methods, but also the result of 2 chromosomes crossover. We have chosen to apply 3 crossover techniques :

- *1-point crossover* that takes two individuals, "cuts" their chromosome strings at some random position to produce two "head" and two "tail" segments and, then, swaps over the tails to produce two full length chromosomes,
- *2-point crossover* that regards chromosome strings as loops by joining the ends together and, then, exchanges a randomly segment from one loop with that from another loop [BBM93b];
- *uniform crossover* that generates a random crossover mask: where there is a 1 in this mask, the gene is copied from the first parent, and where there is a 0, the gene is copied from the second parent [BBM93b].

The genetic algorithm description [HB94, BBM93a, Bal94] is:

```

Generate POPULATION_SIZE random chromosomes
Compute fitness of each individual of the population
Loop GENERATIONS
  Loop POPULATION_SIZE/2
    • Select probabilistically 2 chromosomes  $V_1$  and  $V_2$ 
    • Recombine (thanks to crossover)  $V_1$  and  $V_2$  to give  $C_1$  and  $C_2$ 
    • Perform mutation according to MUT_RATE of  $C_1$  and  $C_2$ 
    • Compute fitness of  $C_1$  and  $C_2$  and insert them in the new generation
   $WORST \leftarrow$  worst vector of the new generation
   $BEST \leftarrow$  best vector of the old generation
  Replace  $WORST$  by  $BEST$  in the new generation

```

The choice of the parent vectors V_1 and V_2 is done thanks to tournament selection. Indeed, it is often used in conjunction with noisy or imperfect fitness functions [MG95] which comes close to our problem which is to inverse a model that gives *approached* values.

Real-coded variant As in the real-coded variant of MRH, the only changes in real-coded GAs concern the way the chromosomes operators are applied. The mutation operator has already been described in the section concerning the real-coded variant of MRH. Moreover, many real-coded crossover operators can be envisaged. However, to keep the idea of some random choice, we apply a *blend-crossover* (BLX- α) [ES92] which is able to create 2 offsprings. Let us consider 2 parents p_1 and p_2 and their 2 offsprings c_1 and c_2 . Then, for $i=1,2$, each component $c_i[j]$ of c_i is a random value that belongs, if we assume $p_2[j] > p_1[j]$, to the interval

$$[p_1[j] - \alpha(p_2[j] - p_1[j]), p_2[j] + \alpha(p_2[j] - p_1[j])]$$

where α is a user specified GA parameter.

4 Model inversion

To supervise the French long distance telephone network, we propose to determine streams responsible for call losses by comparing their traffic values to nominal values. However, stream traffic values are not measured by the on-line data acquisition system and, hence, need to be computed in real time. This is done by inverting the model of Part 2 thanks to the techniques presented in the previous part, where genes values are given by the corresponding stream traffic values ; in binary versions of the presented evolutionary algorithms, Gray code is used to translate real traffic values into binary ones : using Gray code representations lead to better results than using "classical" binary representation [Hol71].

The method consists in calling iteratively the stream propagation model with stream traffic values updated by one of the previous techniques and in using a

fitness measure to compute the distance between the observed values and computed values. First, the observed (measured) quantities can be either the number of calls offered to each organ or the stream carried traffic value, which are the only on-line available measures in the French long distance telephone network.

Then, the distance between observed and computed values can be chosen among:

- the Euclidean Distance (ED), defined by

$$\sqrt{\sum_{i=1}^n (\text{observed\_value}_i - \text{computed\_value}_i)^2}$$

- the χ^2 distance (χ^2), defined by

$$\sqrt{\sum_{i=1}^n \frac{(\text{observed\_value}_i - \text{computed\_value}_i)^2}{\text{observed\_value}_i}}$$

- the Infinite Norm (IN), defined by

$$\max_{i=1 \dots n} |\text{observed\_value}_i - \text{computed\_value}_i|$$

where i is the number of considered quantities.

These distances have been chosen for their simplicity.

The 8 methods described in this paper have been tried successively with 6 fitness measures (the 3 distances being applied to the number of offered calls (OC) as well as the stream carried traffic (CT)).

In a perspective of real-time traffic supervision, each of these methods has been computed with only 500 iterations and they last about 10 minutes on a Sparc Station 5.

For the genetic algorithms, we have chosen :

POPULATION_SIZE = 50, MUT_RATE = 0.01 and $\alpha = 0.5$.

Then, we have tested each of the inversion methods on a set of 34 configurations that regroup all the particularities of the stream propagation model. For each of these 34 tests, we have computed the maximum of the

$$\frac{|\text{real stream traffic value} - \text{computed stream traffic value}|}{\text{real stream traffic value}}$$

Table 1 gives the average of these results.

First, we can notice that, in the case of our application, the choice of the crossover operator do not influence at all the results of binary GA.

Then, Table 1 clearly shows that, in most of the cases, the fittest techniques are the real ones.

	(IN)		(ED)		(χ^2)	
	(CT)	(OC)	(CT)	(OC)	(CT)	(OC)
Bin. MRH	58%	61%	63%	58%	64%	55%
Real MRH	22%	26%	19 %	12%	19%	10%
Bin. PBIL	59%	61%	59%	59%	59%	59%
Real PBIL	282%	39%	255%	37%	236%	19%
1 pt GA	62%	63%	61%	64%	61%	62%
2 pt GA	61%	64%	62%	63%	62%	63%
Uni. GA	60%	63%	62%	62%	62%	60%
Real GA	13%	22%	14%	15%	13%	11%

Table 1. Mean Results

Besides, if the choice of the fitness measure does not really impact on the binary versions of the studied evolutionary computation techniques, it is obviously very important for their real variants. Moreover, it is the fitness measure defined by the χ^2 distance applied to the number of offered calls that gives the better results and, this, whatever the considered evolutionary algorithm. Finally, concerning the choice of the model inversion method, the most promising results are given by either the *real variant of MRH* or the *real variant of GA*.

5 Conclusion

First of all, although the inverted model gives only approximate values and despite its complexity, in term of non-linearities and non differentiability (due to the blocking NEs ordering and to the resource availability backpropagation phenomenon), the results computed thanks to evolutionary computation techniques are quite good.

Then, an important conclusion of our study is the obvious influence of distance choice on evolutionary computation techniques performance. Unfortunately, there is no methodology so far, excepted empirical methods of course, to say a priori whether a distance is able to give better results than an other one or not.

Besides, our application clearly shows that, real-coded variants, that are closer to human reasoning, can outperform binary evolutionary computation techniques.

However, in a perspective of real-time traffic supervision, we cannot afford a larger population because of its long running time. To solve this problem, we envisage to use an explicit parallelism to improve the speed of both population-based incremental learning and genetic algorithms.

6 Acknowledgements

This research was funded by the Centre National d'Études des Télécommunications, contract 93 1B 142, project N°513.

References

- [Bal94] S. Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical Report CMU-CS-94-163, School of computer Science, Carnegie Mellon University, Pittsburgh, 1994. Disponible par ftp anonyme sur le serveur *reports.adm.cs.edu*.
- [Bal95] S. Baluja. An empirical comparison of seven iterative and evolutionary function optimization heuristics. Technical Report CMU-CS-95-193, School of computer Science, Carnegie Mellon University, Pittsburgh, 1995. Disponible par ftp anonyme sur le serveur *reports.adm.cs.edu*.
- [BBM93a] D. Beasley, D.R. Bull, and R.R. Martin. An overview of genetic algorithms : Part 1, fundamentals. *University Computing*, 15(2):58–69, 1993.
- [BBM93b] D. Beasley, D.R. Bull, and R.R. Martin. An overview of genetic algorithms : Part 2, research topics. *University Computing*, 15(4):170–181, 1993.
- [ES92] L.J. Eshelman and J.D. Schaffer. Real-coded genetic algorithms and interval schemata. In *Foundations of Genetic Algorithms 2*, 1992.
- [HB94] J. Heiktocker and D. Beasley. The hitch-hiker's guide to evolutionary computation: A list of frequently asked question. available by anonymous ftp at *rtfm.mit.edu*, 1994.
- [Hol71] R.B. Hollstein. *Artificial genetic adaptation in computer control systems*. PhD thesis, University of Michigan, 1971.
- [MG95] B.L. Miller and D.E. Goldberg. Genetic algorithms, tournament selection and the effects of noise. Technical Report 95006, IlliGAL, Juillet 1995.
- [Pas84] A. Passeron. Notions élémentaires sur le trafic téléphonique. Technical Report DE/ATR/57.84, CNET, 1984.
- [STMS96a] I. Servet, L. Travé-Massuyès, and D. Stern. Traffic supervision based on a one-moment model of telephone networks built from qualitative knowledge. In *IMACS/IEEE CESA '96*, 1996.
- [STMS96b] I. Servet, L. Travé-Massuyès, and D. Stern. Traffic supervision in telephone networks and qualitative modelling. *Annals of telecommunications*, 51(9-10):483–492, 1996.

Genetic Algorithms for Genetic Mapping

Christine Gaspin and Thomas Schiex
{gaspin,tschiex}@toulouse.inra.fr

Institut National de la Recherche Agronomique, Biometry and AI Dept.
Chemin de Borde Rouge BP 27, Castanet-Tolosan 31326 Cedex, France

Abstract. Constructing genetic maps is a prerequisite for most in-depth genetic studies of an organism. The problem of constructing reliable genetic maps for any organism can be considered as a complex optimization problem with both discrete and continuous parameters. This paper shows how genetic algorithms can be used to tackle this problem on simple pedigree. The approach is embodied in an hybrid algorithm that relies on the statistical optimization algorithm EM to handle the continuous variables while genetic algorithms handle the discrete side. The efficiency of the approach lies critically in the introduction of greedy local search in the fitness evaluation of the genetic algorithm, using a neighborhood structure which has been inspired by an analogy between the marker ordering problem and a variant of the famous traveling salesman problem. This shows how genetic algorithms can easily benefit from existing efficient neighborhood structures developed for local search algorithms. The resulting program, called CARTAGENE, has been applied both to real data, from a small parasitoid wasp, and simulated data. In both cases, it compares quite favorably to existing packages.

1 Introduction to Genetic Mapping

The aim of genetic mapping is to locate genes, and more generally genetic markers at *loci* (positions) on the chromosomes. The specific content of a locus in a given chromosome is termed the *allele* (in a computer analogy, a locus is a memory location and the allele is the content of the memory location). Given a locus on a chromosome, individuals of *diploid species* have two alleles, one on each of the corresponding chromosomes contributed by the parents. An individual is said to be *homozygous* at a locus if the two alleles are identical at this locus, else the individual is said to be *heterozygous*.

Each chromosome contributed by each parent is built, during the meiosis, by using sections of either member of each pair of chromosomes of the parent. Changes in the chromosome used are called *crossovers*. At a given locus, there is a 50% chance of having either one of the parental allele. However, the “closer” two loci are on the chromosome, the higher the probability that the two alleles on this chromosome will appear together on the contributed chromosome. Two loci (or genetic markers) are thus said to be *linked* if the parental allele combinations are preserved more often than would be expected by random choice. When a parental allele combination between two loci is *not* preserved, a recombination

event is said to have occurred (this corresponds to an odd number of crossovers between the two loci).

The degree of linkage, or genetic distance, between two loci is a function of the frequency of recombinations. The measurement of genetic distance is expressed in Morgan (or more usually cM for centiMorgan) and is defined as the expected number of crossovers between the two loci on the chromosome.

The idea used to build the genetic map of an organism is to observe the alleles of several loci of interest in a large number of meiosis and to build a map that, in a precise sense, best explains what is observed. In the simplest case, this is done using the so-called *backcross* pedigree (see Fig. 1) which consists in breeding an individual α which is homozygous at all loci of interest with another individual ω which is heterozygous at all loci of interest. After the meiosis, α will contribute a chromosome where all loci have the same allele and therefore are not informative on the recombinations. But ω will give a chromosome where each recombination event changes the type of the allele used in the chromosome. In the figure, if we suppose that the order ABCDE is the correct order, the string 01100 indicates two recombinations, one between A and B, another between C and D. This type of measure is done on a large number of descendants of α and ω .

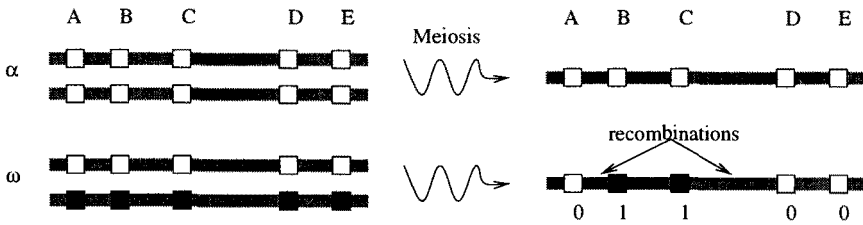


Fig. 1. An example of backcross for one individual and 5 markers (noted A,B,C,D,E).

We consider N markers, and K descendants. For a given loci ℓ and a given descendant i , the allele contributed by the heterozygous parent, which can be either 0 or 1 will be denoted X_{ℓ}^i . This defines the observations and we end up with a data set which is defined by an array of bits X_{ℓ}^i . A map is defined by a linear order of the loci *i.e.*, a one-to-one mapping ϕ from the set $\{1, \dots, N\}$ to $\{1, \dots, N\}$ and a distance between adjacent loci or more precisely, a probability of recombination between two adjacent loci $\phi(\ell)$ and $\phi(\ell + 1)$, denoted $\hat{\theta}_{\ell, \ell+1}$. See fig. 2, for an example of a map.

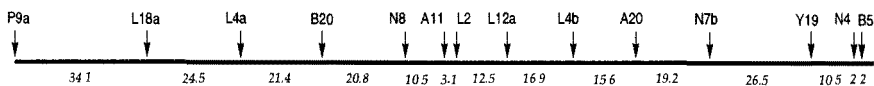


Fig. 2. A genetic map example (parasitoid wasp *Trichogramma brassicae*).

The genetic mapping problem is therefore to find an order of the markers and probabilities of recombination between pairs of markers that best explain the available data set. The task is difficult in several aspects: it is not always obvious to say when a map best explains the data set (criteria to optimize), the space of all maps is defined both by a very large number of orders (rapidly tremendous since $\frac{N!}{2}$ different orders exist) and by $N - 1$ continuous parameters (the distances).

The best criteria one can probably use to quantify how well a map explains the data set is the so-called *multipoint likelihood* of the map *i.e.*, the probability of observing the data set. It is a powerful criteria, which uses all the available evidence, but which may be in itself very expensive to compute. It is used in some packages such as MAPMAKER [Lander et al.87] or LINKAGE [Lathrop et al.85].

Using the previous notations, one can say that a recombination (resp. non-recombination) event has been observed for two adjacent markers when $|X_{\phi(\ell)}^i - X_{\phi(\ell+1)}^i|$ is equal to 1 (resp. 0). If we assume, as it is traditional, that recombination events occur independently in each interval, the probability of the observations given the map is simply obtained by multiplying the probabilities of all the events observed which yields the following formula:

$$\prod_{\ell=1}^{\ell=N-1} \prod_{i=1}^{i=K} \left[(1 - \hat{\theta}_{\ell,\ell+1}) \cdot (1 - |X_{\phi(\ell)}^i - X_{\phi(\ell+1)}^i|) + \hat{\theta}_{\ell,\ell+1} \cdot |X_{\phi(\ell)}^i - X_{\phi(\ell+1)}^i| \right]$$

The optimization problem is now well-defined: we have to find an order of the markers (defined by ϕ) and probabilities of recombinations (the $\hat{\theta}$ s) that maximize the likelihood. For a given order of the loci, we can easily compute the probabilities $\hat{\theta}^*$ that maximize the likelihood. Looking at first and second-order derivatives of the logarithm of previous formula, the $\hat{\theta}^*$ that maximize the likelihood can be easily obtained¹:

$$\hat{\theta}_{\ell,\ell+1}^* = \frac{\sum_{i=1}^{i=K} |X_{\phi(\ell)}^i - X_{\phi(\ell+1)}^i|}{K}$$

So, in this case, when all X_{ℓ}^i are known, and as far as recombination fractions are considered, the logarithm of the likelihood for optimal recombination fractions can be rewritten:

$$\sum_{\ell=1}^{\ell=N-1} K \cdot \underbrace{\left[\hat{\theta}_{\ell,\ell+1}^* \log(\hat{\theta}_{\ell,\ell+1}^*) + (1 - \hat{\theta}_{\ell,\ell+1}^*) \log(1 - \hat{\theta}_{\ell,\ell+1}^*) \right]}_{\text{elementary contribution to log-likelihood}}$$

Thus, the maximum log-likelihood of an order is equal to a sum of elementary contributions which depend only on two loci. One can therefore precompute all

¹ This is obtained by solving the simple equations stating that first-order derivatives are equal to 0. At this point, one can further notice that the matrix of second order derivatives is diagonal negative on the domain of optimization, which shows that this point is a maximum.

these contributions for all pairs of loci and the problem of finding an order that maximizes the maximum loglikelihood is in essence identical to the *symmetric wandering salesman problem* (a variant of the famous *symmetric traveling salesman problem*): given n cities and the distances between each pair of cities, find a path that goes once through each city and that minimizes the overall distance. The choice of the first and last cities in the path is free. One can simply associate one imaginary city to each marker, and define as the distance between two cities the opposite of the elementary contribution to the loglikelihood defined by the corresponding pair of markers².

This connection is interesting in several aspects: the WSP is known to be an NP-hard problem and this shows that the marker ordering problem may be difficult in some cases (computational complexity theory tells us that the tremendous number of existing orders is not sufficient to conclude this). More interestingly, all the techniques which have been developed for the TSP, and which can easily be adapted to the WSP, can also be applied here.

However, *real* data usually contain missing measures *i.e.*, some of the X_i^j are unknown. In fact, there can be a lot of extra missing data introduced if the data set has been obtained by pooling several data sets on several different families, each family being informative on a different set of markers. In order to be able to handle data sets with missing measures, the quality of a map will not be simply obtained by summing elementary contributions as above, but using a dedicated statistical optimization algorithm : the EM algorithm [Dempster et al.77]. The algorithm given an order, computes the maximum likelihood of this order. It is a relatively expensive iterative procedure, each iteration being in $O(NK)$. Naturally, the pure theoretical connection with the WSP is lost when missing data exist, but our assumption is that the structure of the problem will still be close to the structure of the WSP and therefore, that techniques which proved to be efficient for the TSP will be efficient on the marker ordering problem.

2 Genetic mapping solving

When missing measures appear, the connection with the TSP is theoretically lost and the best available techniques, like Branch and Cut [Applegate et al.95], cannot be used. In that case, heuristic approaches offer an alternative to solve difficult optimization problems. Because heuristic approaches do not offer any guaranty of optimality, we have used two types of algorithms to solve the genetic mapping problem: tabu search (TS) and genetic algorithm (GA). This allows to give more confidence in results when similar optima are found.

Genetic mapping software should not only give the most likely map, but also be able to indicate how strongly the best map is supported by the data (if there is another map whose likelihood is very close to the optimal map, then there is no reason to choose one or the other). To offer this service, our software, CAR_TAGENE, maintains a set of fixed size containing the k best different

² We would like to acknowledge the fact that the similarity between TSP and simpler two-points criteria is mentioned in [Liu95].

maps encountered during the search (k fixed, actually equal to 31). A hash table [Cormen et al.90] is used to efficiently test if the map is already in the set, a heap structure is used to efficiently manage insertions/deletions [Cormen et al.90]. At the end of the search, the user can browse this set and check for the existence of other maps whose likelihood is close to the best map's likelihood in order to get an idea of how strongly the best map is supported. This is incorporated in all the algorithms presented in the sequel.

2.1 Tabu search algorithm (TS)

Tabu search [Glover89, Glover90] repeatedly scans the current neighborhood, selecting the best neighbor (the neighbor with the best likelihood) to be the new solution. The neighborhood we have chosen is the *2-change* neighborhood of the individual, a successful well-known neighborhood structure introduced in [Lin et al.73] to tackle the TSP. Adapted to the WSP, the *2-change* neighborhood of a map is the set of all maps obtained by an inversion of a subsection of the map. Thus, for N markers, the neighborhood has a size of $\frac{N \cdot (N-1)}{2} - 1$.

To avoid being stucked in local optima, the content of the neighborhood of the current solution is influenced by a memory mechanism which may forbid some moves (which are said to be tabu) in the neighborhood. In CAR_HAGENE, the tabu moves are the recent moves (*i.e.*, subsections of the map which have been recently inverted). The precise definition of “being recent or not” varies stochastically during search as advocated by [Taillard91]. A tabu move may eventually be chosen if it leads to a map which improves the best likelihood known (this is called “aspiration” in tabu terminology).

We observed that tabu moves, while breaking cycles, weren't able to avoid being stuck in large locally optimum plateaus. We therefore used the hash table that memorizes the best orders encountered to also memorize the number of times each of them is reached. When this count exceeds a fixed number, a random jump is performed. This largely enhanced performances. However, since only the best solutions are memorised in the hash table, it is still useless if the algorithm is stucked in a plateau of poor quality (defined by orders whose likelihood is far from the best likelihoods memorized in the hash table). Further improvement should be possible by adding a new hash table that memorizes *recent* positions rather than *best* positions.

2.2 Basic genetic algorithm (BGA)

Genetic algorithms are general adaptative heuristic search algorithms based on an analogy with the genetic structure and behavior of chromosomes within a population of individuals. Individuals represent potential solutions to a given problem. A fitness score is associated to each individual and represents its adaptation ability. The algorithm makes the population of individuals evolve maintaining both diversity and favoring the existence of best individuals. Starting from an initial population, a new generation is created by randomly applying

mutation and by crossing pairs of individuals (favoring crosses of good individuals). The hope is that the population will evolve towards one which contains optimal individuals with respect to the fitness.

Representation: In CARTAGENE, each individual represents one genetic map (an ordering of markers). The crossover operator used is the order crossover [Telfar94] which computes two offspring individuals, I_1 and I_2 from two parents P_1 and P_2 (figure 3). The parents are cut into three sections by selecting randomly two markers. The middle section of P_1 is copied into the corresponding position of I_1 , the rest of I_1 being filled in with values taken in order from the third, first and second section of P_2 , skipping values that have already been copied from the first parent. I_2 is computed in the same way by reversing the parents.

P1 :	2 3 7	4 1 6 9	8 5 10
P2 :	5 9 1	6 7 3 2	10 4 8
I1 :	7 3 2	4 1 6 9	10 8 5
I2 :	4 1 9	6 7 3 2	8 5 10

Fig. 3. Example of the order crossover operator. I_1 is computed from its parents P_1 and P_2 . The middle section of P_1 is copied into the middle section of I_1 . The rest of I_1 is filled in with respectively 10 and 8 from section 3 of P_2 , 5 from section 1 of P_2 and 7, 3 and 2 from section 2. I_2 is computed in the same way by reversing the parents.

The mutation operator selects two markers and simply exchanges them. The selection relies on a biased roulette wheel [Goldberg89]. For markers ordering, a solution is an order of the markers, and its fitness score is the maximum likelihood of the map. Thus a simple fitness function is given by the evaluation of the map using the EM algorithm.

A comparison of the basic genetic algorithm with tabu search was performed on randomly generated problems³. On such problems, the original map used to generate the data, and its likelihood are naturally available. The likelihood of the original map is usually among the highest and gives a generally good lower bound of the optimal likelihood.

	GO	BL
<i>BGA</i>	59 %	70 %
<i>TS</i>	81 %	100 %

Fig. 4. Results

In our test, we measured the number of times each algorithm was able to found the original order (noted GO, aka Good Order) and the number of times each algorithm was able to found a likelihood larger than or equal to the likelihood of the original map (noted BL, aka Better Likelihood). The results show that *TS* gives better results than *BGA*. We explain these results by the power of the greedy optimization which is performed in *TS*. This suggested us to embed such a greedy optimization in *BGA*.

³ Simulated data consist of $N = 25$ markers per individual and the number of individuals in each data set is 100. The probability p for an allele to be missing is 20%. One hundred problems were solved.

2.3 Incorporating greedy optimization in genetic algorithms

There are several different possible ways to use the idea of *2-change* neighborhood in a genetic algorithm:

- A first question is whether a full greedy optimization should be performed (until a local optima is reached) or if a limited and cheaper optimization (a fixed number of greedy steps) would suffice.
- A second question is when the actual greedy optimization should take place. It could be incorporated in the mutation operator, or applied systematically (eg. inside the fitness evaluation function).

After several trials, it appeared that performing a full greedy optimization (until a local optimum is reached) on each new individual yields the best performances. In practice, the greedy optimization is performed during fitness evaluation: when a local optima is reached, the individual is replaced by this local optimum and its likelihood is used as the fitness. This function usually modifies the individual under evaluation. It is expensive, but it yields the best results.

With this approach, the genetic algorithm does not perform an optimization in the full space of all orders, but only in the space of local optima, skipping from a local optimum to another throughout the crossover and mutation operators. In the sequel of the paper, the resulting algorithm will be denoted *GA*. On the previous data sets, it performed exactly as *TS*.

3 Experimental results

CAR_HTAGENE has been applied both on simulated and real backcross-like data. In each case, an empirical analysis is used to compare genetic algorithm (*GA*) and tabu search.

Results now compare the performances of the algorithms when running with their own stopping criteria. In each case, the stopping criteria depends on the number of markers and performances are evaluated with regard to the quality of solutions and the number of EM calls.

3.1 Simulated data

In Figure 5 we investigate the relation between the percentage of missing data and the number of individuals needed to find optimal orders that are original orders. In each individual data set, the number of markers is 10. The number of individuals in each data set is either 25, 50, 100 or 250. The probability for an allele to be missing varies from 0% to 20% by 5% step. One hundred problems are solved for each combination of these parameters. Each curve is associated with a number of individuals and gives the percentage of original orders found by CAR_HTAGENE. It appears that 250 or even 100 individuals seem to be largely sufficient to find the good order even with missing data. These curves are similar for both *TS* and *GA*.

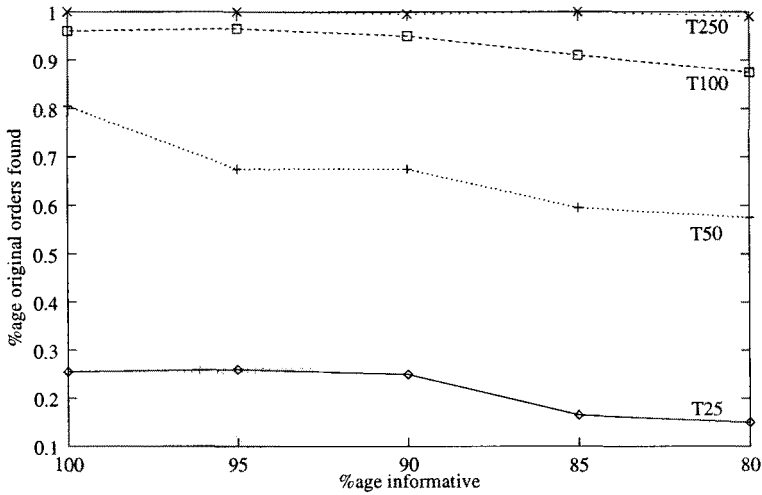


Fig. 5. Percentage of original orders found.

We do not report here the curves giving the percentage of maps found with a loglikelihood equal or better than the loglikelihood of the original map. It was always equal to 100%.

Table 1 shows results obtained by running genetic algorithm and tabu search with their own stopping criteria on backcross-like data involving two populations.

	2 markers in common					5 markers in common					10 markers in common				
	Failures			EM calls		Failures			EM calls		Failures			EM calls	
size	TS	GA	both	TS	GA	TS	GA	both	TS	GA	TS	GA	both	TS	GA
25	4	0	0	29998	38433	4	0	0	20833	25879	7	4	1	9672	10093
50	18	3	0	31174	39880	6	2	0	21460	28305	9	7	2	9546	9259
100	13	3	0	32750	41120	2	0	0	21959	29688	7	6	0	9338	9218
250	10	0	0	33903	42961	4	0	0	22554	30129	25	8	3	9430	8427
Mean	11.2	1.5	0	31956	40598	4	0.5	0	21702	28500	12	6.25	1.5	9496	9249

Table 1. Comparison of GA and TS with an increasing number of common markers and different sizes of sample. A failure occurs when the best likelihood found is worse than the likelihood of the original map.

Data have been generated as described in [Schiex et al.97]. In each individual data set, the number of markers is 15. The number of common markers is respectively 2, 5 and 10. The size of each data set is either 25, 50, 100 or 250, leading to a joined map built from data sets of 50, 100, 200 and 500 individuals. The probability p for an allele to be missing is 20%. Additional blocks of missing

data are introduced which come from markers available in one individual data set and not in the other. This results in a percentage of missing data always higher than p depending on the number of common markers. The number of markers varies from 20 (15 markers in each population and 10 common markers) to 28 (15 markers in each population and 2 common markers). One hundred problems are solved for each combination of these parameters.

Stopping criteria have been tuned so that “good quality” solutions are produced in a “reasonable” amount of time. Tabu search stopping criteria considers the number of iterations which have not improved the likelihood of the best solution. When this number is equal to twice the number of markers in the data set, tabu search stops. Genetic algorithm stops as soon as two generations of individuals produce the same best individual or the maximum likelihood has not been improved.

In table 1, it appears that the number of EM calls decreases with the number of common markers for both approaches. This can be explained by the fact that the total number of markers decreases with the increasing number of common markers and the stopping criteria depends on the total number of markers.

In all cases *GA* appears better than *TS* in finding best maps and this remains true for other values of p ($p = 10\%$ and $p = 0\%$, results not given there). Nevertheless, one can note that *TS* and *GA* generally do not fail on the same instances. One can exploit this complementarity to obtain a lower failing rate. Nevertheless, in all cases, except for 10 markers in common and sample sizes 50, 100 and 250, *TS* stops before *GA*. In spite of the fact that the case of 10 common markers indicates *GA* as the most powerful approach as well for the number of EM calls as the quality of solutions, these results are certainly strongly linked to the running protocols that have been used and therefore have to be considered with caution.

3.2 Real backcross-like data

The real data consist of backcross-like data for the *Trichogramma brassicae* genome. Full results are described in [Laurent et al.] where *TS* and *GA* give similar solutions. One can note there that *TS* was always less expensive in EM calls than *GA*. We also compared CAR_HAGENE with existing software on these data. For all the maps, the orders obtained using CAR_HAGENE were more likely (up to 10^{16} more likely) than JOINMAP, an existing commonly used software [Schiex et al.97].

Conclusion

CAR_HAGENE shows how existing efficient neighborhood structures can be exploited to boost the performances of genetic algorithms. It is likely that the idea of using greedy local search in the fitness function could be used for other problems as well.

Experiments show how difficult the measure and comparison of performances between different approaches is. On simulated and real backcross-like data tabu search and genetic algorithms give similar results when the number of markers is around 15. In those cases, *TS* is always less consuming in EM calls than *GA*. The case of simulated data with a higher number of markers and 20% of missing data gives *GA* as the most robust approach. This result is to be considered with caution and more experiments remain to do to confirm it. More interestingly, the analysis of the failures suggests to combine the two approaches in order to increase the reliability.

In genetic mapping, *CARTAGENE* extends the scope of application of multi-point maximum likelihood criterion to data sets with larger number of markers and larger amount of missing, a situation frequently encountered when the data set is build from several families. In this case, it compares quite favorably with other dedicated packages [Stam93, Schiex et al.97].

References

- [Applegate et al.95] Applegate (D.), Bixby (R.), Chvátal (V.) et Cook (W.). – *Finding cuts in the TSP (a preliminary report)*. – Technical Report 95-05, DIMACS, March 1995.
- [Cormen et al.90] Cormen (Thomas H.), Leiserson (Charles E.) et Rivest (Ronald L.). – *Introduction to algorithms*. – MIT Press, 1990. ISBN : 0-262-03141-8.
- [Dempster et al.77] Dempster (A.P.), Laird (N.M.) et Rubin (D.B.). – Maximum likelihood from incomplete data via the EM algorithm. *J. R. Statist. Soc. Ser.*, vol. 39, 1977, pp. 1–38.
- [Glover89] Glover (F.). – Tabu search – part I. *ORSA Journal on Computing*, vol. 1 (3), Summer 1989, pp. 190–206.
- [Glover90] Glover (F.). – Tabu search – part II. *ORSA Journal on Computing*, vol. 2 (1), Winter 1990, pp. 4–31.
- [Goldberg89] Goldberg (D. E.). – *Genetic algorithms in Search, Optimization and Machine Learning*. – Addison-Wesley Publishing Company, 1989.
- [Lander et al.87] Lander (E.S.), Green (P.), Abrahamson (J.), Barlow (A.), Daly (M. J.), Lincoln (S. E.) et Newburg (L.). – MAPMAKER: An interactive computer package for constructing primary genetic linkage maps of experimental and natural populations. *Genomics*, vol. 1, 1987, pp. 174–181.
- [Lathrop et al.85] Lathrop (G.M.), Lalouel (J.M.), Julier (C.) et Ott (J.). – Multilocus linkage analysis in humans: detection of linkage and estimation of recombination. *Am. J. Hum. Genet.*, vol. 37, 1985, pp. 482–488.
- [Laurent et al.] Laurent (V.), Vanlerberghe-Masutti (F.), Wajnberg (E.), Mangin (B.), Schiex (T.) et Gaspin (C.). – Construction of a composite genetic map of the parasitoid wasp *trichogramma brassicae* using RAPD informations from three populations. *Submitted*.
- [Lin et al.73] Lin (S.) et Kernighan (B. W.). – An effective heuristic algorithm for the traveling salesman problem. *Operation Research*, vol. 21, 1973, pp. 498–516.

- [Liu95] Liu (B. H.). – The gene ordering problem, an analog of the traveling salesman problem. *In: Plant Genome 95*.
- [Schiex et al.97] Schiex (T.) et Gaspin (C.). – Cartagene: Constructing and joining maximum likelihood genetic maps. *In: Proceedings of the fifth international conference on Intelligent Systems for Molecular Biology*. – Porto Carras, Halkidiki, Greece, 1997. Available at <http://www-bia.inra.fr/T/schiex>.
- [Stam93] Stam (P.). – Constructing of integrated genetic linkage maps by means of a new computer package:JOINMAP. *The Plant Journal*, vol. 3 (5), 1993, pp. 739–744.
- [Taillard91] Taillard (E.). – Robust taboo search for the quadratic assignment problem. *Parallel computing*, vol. 17, 1991, pp. 443–455.
- [Telfar94] Telfar (G.). – *Generally Applicable Heuristics for Global Optimization: An Investigation of Algorithm Performance for the Euclidean TSP*. – Master's thesis, Victoria University of Wellington, 1994.

Inverse Problems for Finite Automata: A Solution Based on Genetic Algorithms

B. Leblanc¹, E. Lutton¹ and J.-P. Allouche²

¹ INRIA - Rocquencourt, B.P. 105, F-78153 LE CHESNAY Cedex, France
Tel: +33 (0)1 39 63 55 23 - Fax: +33 (0)1 39 63 59 95
e-mail: Benoit.Leblanc@inria.fr, Evelyne.Lutton@inria.fr
<http://www-rocq.inria.fr/fractales/>

² CNRS, LRI, Bât. 490, Université Paris-Sud, F-91405 Orsay Cedex, France
Tel: 33 (0)1 69 15 64 54
e-mail: Jean-Paul.Allouche@lri.fr

Abstract. The use of heuristics such as Genetic Algorithm optimisation methods is appealing in a large range of inverse problems. The problem presented here deals with the mathematical analysis of sequences generated by finite automata. There is no known general exact method for solving the associated inverse problem. GA optimisation techniques can provide useful results, even in the very particular area of mathematical analysis. This paper presents the results we have obtained on the inverse problem for fixed point automata. Software implementation has been developed with the help of “ALGON”, our home-made Genetic Algorithm software.

1 Introduction

A *finite automaton* is defined as a symbolic substitution σ acting on strings of symbols. More precisely σ is a map from a finite set of symbols S to S^* , the set of strings of symbols in S . The elements of S^* are called *words* and the images by σ of elements of S are called *words of the automaton*. The map σ is extended to S^* by concatenation (the image of a word is obtained by concatenating the images of its symbols), making σ a morphism of the free monoid S^* .

A sequence of words can be produced by successive applications of σ to an initial word s_0 . If we denote by $s_n = s_{n_1}s_{n_2}\dots s_{n_p}$ the word at step n , the word obtained at step $n+1$ is then:

$$s_{n+1} = \sigma(s_n) = \sigma(s_{n_1})\sigma(s_{n_2})\dots\sigma(s_{n_p}).$$

Note that the words $(s_n)_{n \in \mathbb{N}}$ are concatenations of the words of the automaton (in Example 1 this fact is highlighted by the alternation of bold and standard fonts).

Example 1. $S = \{1, 2, 3\}$

$$\sigma \begin{cases} 1 \rightarrow 211 \\ 2 \rightarrow 13 \\ 3 \rightarrow 123 \end{cases}$$

Iteration	Word
0	1
1	211
2	13211211
3	2111231321121113...

Of course, it is clear that the sequence of words $(s_n)_{n \in \mathbb{N}}$ is determined by the automaton σ and the initial word s_0 .

An interesting property of such words concerns the frequency of occurrences of symbols of S . Let σ be an automaton acting on $S = \{\alpha_1, \alpha_2, \dots, \alpha_m\}$, let s_0 be an initial word. The number of occurrences of any of the symbols observed in the word s_k (for any k) can be computed. Let us denote by

$$O_k = \begin{pmatrix} o_k^1 \\ o_k^2 \\ \dots \\ o_k^m \end{pmatrix}$$

the occurrence vector of s_k (o_k^i being the number of occurrences of the symbol α_i observed in s_k). Then $O_{k+1} = A * O_k$ with $A = (a_{ij})_{(i,j) \in \{1, \dots, m\}^2}$ the “growth” matrix, a_{ij} being the number of symbols α_i in the word $\sigma(\alpha_j)$.

We thus obtain $O_k = A^k * O_0$ with O_0 the occurrence vector of s_0 . For Example 1, we have:

$$A = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}.$$

Let us also define the square (and in the same way any power) of an automaton: $\forall i \in \{1, \dots, m\}$, $\sigma^2(\alpha_i) = \sigma(\sigma(\alpha_i))$. The associated matrix is then A^2 . For more information about substitutions, see [4].

2 The inverse problem for finite automata

2.1 Motivations and formulation

To know whether a given sequence is generated by a finite automaton and to know explicitly one such automaton, can be useful in many situations. We list but three of them.

- In combinatorics on words: the third author, J. Currie and J. Shallit proved recently that the lexicographically least overlap-free³ sequence on a two-letter alphabet, that begins with a given word (if it exists), must end with a

³ An *overlap* is a string of the form $axaxa$ where a is a letter and x a (finite) word. A (finite or infinite) word is called *overlap-free* if it does not contain any overlap.

tail of the Thue-Morse sequence $10010110\dots$, hence is the pointwise image (i.e. image under a morphism that sends each letter to a letter) of a fixed point of a morphism of constant length [8]. It is not known whether the least square-free sequence on a three-letter alphabet has the same property.

- In number theory: let $(u_n)_{n \in \mathbb{N}}$ be a sequence with values in the finite field $\mathbb{F}_q$. Then the formal power series $\sum u_n X^n$ is algebraic over the field of rational functions $\mathbb{F}_q(X)$ if and only if the sequence $(u_n)_{n \in \mathbb{N}}$ is the pointwise image of a fixed point of a morphism of length q over a finite alphabet, [10, 11]. For example the transcendence of values of Carlitz functions can be proved by showing the non-automaticity of the corresponding formal power series (see for example [9, 7, 12]). A hint that a sequence is not a fixed point of a morphism (it is more complicated for the pointwise image of a fixed point) is that, when solving the inverse problem with longer and longer prefixes of the sequence, the automaton we obtain keeps growing. That means almost certainly that there is no automaton that generates the infinite sequence.
- In physics: quasi-crystals are the 3-D analogue of the Penrose tiling. A one-dimensional description of this tiling involves the Fibonacci sequence, i.e., the fixed point of the morphism $0 \rightarrow 01, 1 \rightarrow 0$.

Another occurrence of this “inverse” question for finite automata, in music, is recalled below.

Suppose we have a string of symbols and we want to know whether it has been produced by iterating an automaton. Of course if the string is finite, there always exists a trivial solution: the substitution sending the first letter of the string to the string itself. But we are interested in non-trivial solutions if any. Apart from the mathematical aspects of this “inverse” problem, it might be interesting to note that this work began as a composer, T. Johnson, produced a sequence of notes using a finite automaton, kept only the sequence of notes, and wanted to remember the automaton he used. (For the use of finite automata in a piece of T. Johnson, see [1].)

If the word s_0 happens to be a prefix of the word $s_1 = \sigma(s_0)$, it is not hard to see that the sequence of words $(s_n)_{n \in \mathbb{N}}$ converges to an infinite word. And this infinite word is clearly a *fixed point* of the substitution σ (extended to infinite words by concatenation). Now suppose an infinite word is given, is this word the fixed point of a substitution? Or is it the pointwise image of a fixed point of a substitution? No general answer to these questions is known either: a theoretical answer to the second question is known if the substitution has constant length [3] (i.e., if all words of the automaton have the same length), and also if the substitution is *primitive* (i.e., is such that there exists an m with the property that σ^m of any symbol contains at least one occurrence of each symbol of the set S) as proved recently by Durand [5]. Looking at finite prefixes of the given infinite sequence that have well-chosen lengths, we see that we can first restrict to finite words.

Hence, ideally, we would like to solve in the general case (i.e., not only for fixed point automata) what we called the inverse problem, that is:

Given a finite word s , find an automaton σ and an initial word s_0 such that $\sigma^n(s_0) = s$ for some n .

Of course, in its generality, this problem is extremely complex and can have many solutions or no non-trivial solution at all⁴.

It can be reformulated as an optimisation problem on the search space of all possible σ , n and s_0 that minimizes a distance between $\sigma^n(s_0)$ and s . In order to reduce the complexity of this problem one also has to give some restrictions to the search space. We suppose in the following that the length of the words of σ is limited to l_{max} and that s_0 is a single symbol.

2.2 A bruteforce GA implementation

The first GA implementation that comes to mind is to perform a search over the space of all automata having word lengths smaller than or equal to l_{max} . Each individual of the GA thus represents an automaton with the following characteristics:

- m chromosomes for an individual: one per word of the automaton;
- variable length chromosomes: their lengths may vary between 1 and l_{max} ;
- an m -ary coding: the allele set is S .

These particular characteristics imply of course some modified GA operators, that are implemented in ALGON [6].

Thus, setting $l_{max} = 4$, the automaton of Example 1 would have the coding shown in Figure 1.

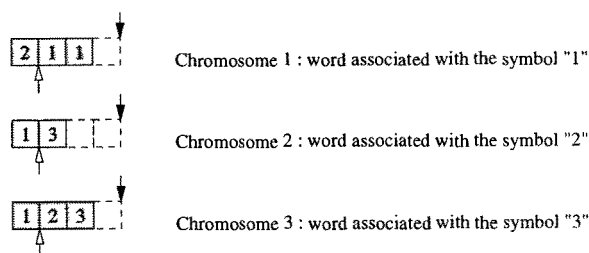


Fig. 1. Direct coding of example 1 with $l_{max} = 4$.

The fitness function that must reflect the “resemblance to the target”, is based on a comparison between words (Hamming distance, frequencies of occurrences of symbols, of couples of symbols, etc. . .).

⁴ In fact, for a given solution couple (σ, s_0) and for any divisor p of n , the couple (σ^p, s_0) is also a solution.

This approach did not lead to interesting results, due to the size of the search space with respect to m and l_{max} : $|S| = \left(\sum_{i=1}^{l_{max}} m^i\right)^m$. The size would not be a problem if the resulting fitness landscape was smooth enough⁵, but in our case one can easily check that a single change in the genetic code leads to important changes in the observed words.

Though this direct approach is not appropriate, its benefit is to highlight the difficulty of solving this problem. In fact, it is obvious that the coding should use more efficiently the information contained in the target word.

3 The fixed point hypothesis

If we restrict the inverse problem to the search of automata with fixed points, the complexity of the problem is reduced.

3.1 Definition and properties

A finite automaton has a fixed point if there exists a symbol α such that the first letter of $\sigma(\alpha)$ is α itself.

The sequence of words s_n produced by such an automaton starting with the initial symbol $s_0 = \alpha$ converge to a fixed point of σ : the beginning of the word at iteration $n + 1$ is exactly the word at iteration n . In fact each iteration adds symbols to the end of the previous word.

Example 2. $S = \{1, 2, 3\}$

$$\sigma \begin{cases} 1 \rightarrow 21 \\ 2 \rightarrow 231 \\ 3 \rightarrow 13 \end{cases}$$

Iteration	Word
0	2
1	231
2	2311321
3	2311321211323121...

The inverse problem for a fixed point automaton is then much easier to solve than in the general case. Indeed, the information contained in the target word can be efficiently exploited, taking advantage of the fact that a fixed point is a succession of words of the automaton as well as the succession of symbols which generated them. Of course, it is necessary to know the lengths of the words in order to identify the connection between the two successions. A simple assumption on the lengths of words of the automaton permits then to identify it with a mechanism of simultaneous identification and reconstruction.

Checking an hypothesis is then a direct process of “reconstruction-comparison”. As previously outlined, the first symbol of the fixed point is associated to the first word, its size is given by assumption, so it can be identified. The second

⁵ With a few secondary optima.

Incorrect hypothesis (2,2,2).		Correct hypothesis (2,3,2).	
$\sigma(2) = 23$	23 11321211323121	$\sigma(2) = 231$	231 1321211323121
$\sigma(3) = 11$	23 11 321211323121	$\sigma(3) = 13$	231 13 21211323121
$\sigma(1) = 32$	2311 32 1211323121	$\sigma(1) = 21$	23113 21 211323121
$\sigma(1) = 12$	231132 12 11323121	$\sigma(1) = 21$	2311321 21 1323121
Contradiction on "1".		$\sigma(3) = 13$	231132121 13 23121
		$\sigma(2) = 231$	23113212113 231 21
		$\sigma(1) = 21$	23113212113231 21

Fig. 2. Hypothesis propagation.

word, associated to the second letter, start right after the first, and knowing its size by hypothesis it can be identified too, and so on all along the fixed point. If the hypothesis is not correct, then the case will arise when the same symbol will be associated to two different words, discarding it. If it is correct the whole fixed point will be "reconstructed" without such contradiction.

Let us consider Example 2 and take $s = \sigma^3(2)$ as the target word. Assume each word of σ has two symbols:

- The initial symbol s_0 is simply the first symbol of s , i.e., $s_0 = 2$.
- The word associated with this symbol is a prefix of the target word, and it is assumed to be composed of two symbols, then we directly identify $\sigma(2) = 23$.
- The identification process is continued until a contradiction appears or the end of the target word is reached, as shown in Figure 2: in the incorrect hypothesis case we get $\sigma(1) = 32$ at step 2 and $\sigma(1) = 12$ at step 3 then the hypothesis is infirmed. Conversely, in the correct case, for the same symbol, the same word is always recognized, so the whole fixed point word is "reconstructed".

3.2 A GA to search the space of word lengths

Coding the individuals :

An individual of the GA population may just represent an assumption on the words lengths, the corresponding automaton being reachable through the identification mechanism previously exposed.

If we set an upper limit l_{max} for the possible lengths of the words, the genetic coding is the following:

- A set of alleles of cardinality l_{max} .
- A single chromosome per individual containing as many genes as elements in the symbol set S . The gene k codes the length of the word associated with the symbol α_k . The coding of a right assumption for Example 2 is:

$$|2|3|2| \rightarrow \sigma \begin{cases} 1 \rightarrow ?? \\ 2 \rightarrow ??? \\ 3 \rightarrow ?? \end{cases}$$

Compared to the “brute-force” implementation, a substantial improvement is the reduction of the search space which size is now $|S| = (l_{max})^m$.

Fitness function :

The evaluation of an individual relies on the validation process of the assumption it encodes. If the assumption appears to be valid, it is assigned a maximal fitness value. Note that any power of an automaton that is a solution to the problem is also a solution. Hence there is a potentially infinite number of solutions, as soon as one solution is found. But practically the number of solutions is limited by the length of the target word and the l_{max} value. The minimal solutions (in terms of lengths) are obviously the most interesting ones.

Invalid assumptions are given an intermediate value, and it is also desirable to differentiate these non-valid assumptions in order to drive the search towards a solution.

If a contradiction arises, two cases are considered:

- The contradiction arises before the identification of all the words of the automaton:

$$f(i) = \epsilon * \text{Number of identified words}$$

with ϵ a very small positive value.

- If a contradiction arises after the identification of all words of the automaton:

$$f(i) = \left(\frac{\text{Length of the “checked” sequence}}{\text{Length of the target sequence}} \right)$$

The “checked” word denotes the part of the target word checked before the contradiction occurs.

The maximum of f is then 1, corresponding to the case where the target word has been entirely checked. In order to give a best fitness value to any assumption leading to a complete identification of the words of the automaton than to any other that doesn't, the number ϵ simply has to fulfill the following condition:

$$\epsilon < \left(\frac{m + 1}{\text{Length of the target word}} \right).$$

3.3 Results and discussions

We present here results obtained with ALGON [6], on two target words which are prefixes of fixed points of two different automata using 6 symbols. The maximal lengths of words being $l_{max} = 6$, the size of the search space is then: $|S| = 6^6 = 45656$

The general parameters of the GA are:

- A population of 100 individuals. Each individual being unique.
- A mutation probability $p_m = 0.125$.
- One point crossover with probability $p_c = 0.85$.

- An elitist population replacement with a ratio $r_s = 0.4$ of surviving individuals, i.e., 60 new individuals are created at each generation replacing the 60 worst individuals of the previous population.
- Selection performed with *Stochastic Universal Sampling* (see [2]).

Automaton 1:

$$\sigma \left\{ \begin{array}{l} 1 \rightarrow 61 \\ 2 \rightarrow \mathbf{234} \\ 3 \rightarrow 52 \\ 4 \rightarrow 234 \\ 5 \rightarrow 6551 \\ 6 \rightarrow 433 \end{array} \right. \quad (1)$$

The target word s is then obtained by iterating 5 times the automaton starting from the initial seed “2”, that is a word of length 196. We present in table 1 some statistics obtained over 20 runs. The following quantities are computed:

N_1 : Number of generations to obtain an assumption leading to a complete automaton (before a contradiction arises in the identification process).

N_2 : Number of generations to find a solution.

$N_3 = N_2 - N_1$: number of generations to find a solution when at least one individual has lead to a complete automaton.

The results are summarized in table 3.3.

Table 1. Results for automaton 1.

	Mean	Std.
N_1	4.2	2.38
N_2	18.95	24
N_3	14.75	23.3

About 1000 fitness evaluations are necessary to find a solution, which is to be compared to the search space size.

Automaton 2:

$$\sigma \left\{ \begin{array}{l} 1 \rightarrow 11116 \\ 2 \rightarrow \mathbf{24} \\ 3 \rightarrow 5 \\ 4 \rightarrow 35 \\ 5 \rightarrow 23341 \\ 6 \rightarrow 666 \end{array} \right. \quad (2)$$

The target word s is again obtained by iterating 5 times the automaton starting from the initial seed “2”. It has been designed to slow down the automaton

identification process: the symbol “6” first appears quite far in s (26th position), so a contradiction has a greater chance to arise before all words have been identified.

Table 2. Results for automaton 2.

	Mean	Std.
N_1	8.65	2.38
N_2	13.2	7.76
N_3	4.55	3.76

We can see that, for this apparently more tricky automaton, the performances of the GA are better. But it can certainly be explained by the fact that the frequency of the hypothesis leading to a complete automaton identification (before a contradiction arises) is lower than previously, but other points of the search space seem to lead quite easily to those interesting regions.

4 Conclusion and further works

The results we obtained on fixed points automata suggest a coding of the general problem based on a set of possible words observed in the target word to be analysed. Such an approach, by considerably reducing the search space of possible automata, allows to obtain interesting results in the general case. This will be studied in a forthcoming paper.

References

1. J.-P. Allouche, T. Johnson (1995): *Finite automata and morphisms in assisted musical composition*. Journal of New Music Research **24**, 97–108.
2. J. E. Baker (1987): *Reducing bias and inefficiency in the selection algorithm*. Genetic Algorithms and their application: Proceedings of the Second International Conference on Genetic Algorithms, p. 14–21.
3. A. Cobham (1972): *Uniform tag sequences*. Math. Systems Theory **6**, 164–192.
4. S. Eilenberg (1974): *Automata, Languages, and Machines*. Vol. A, Academic Press.
5. F. Durand (1997): *A characterization of substitutive sequences using return words*. Disc. Math., to appear.
6. B. Leblanc, E. Lutton (1997): *ALGON: A Genetic Algorithm software package*, <http://www-rocq.inria.fr/fractales/>
7. J.-P. Allouche (1996): *Transcendence of the Carlitz-Goss Gamma function at rational arguments*. J. Number Theory **60**, 318–328.
8. J.-P. Allouche, J. Currie, J. Shallit (1997): *Extremal infinite overlap-free binary words*. Preprint.

9. V. Berthé (1994): *Automates et valeurs de transcendance du logarithme de Carlitz*. Acta Arith. **66**, 369–390.
10. G. Christol (1979): *Ensembles presque périodiques k -reconnaissables*. Theoret. Comput. Sci. **9**, 141–145.
11. G. Christol, T. Kamae, M. Mendès France, G. Rauzy (1980): *Suites algébriques, automates et substitutions*. Bull. Soc. Math. France **108**, 401–419.
12. M. Mendès France, J.-y. Yao (1997): *Transcendence and the Carlitz-Goss gamma function*. J. Number Theory **63**, 396–402.

Evolving Turing Machines from Examples

Julio Tanomaru

Faculty of Engineering, The University of Tokushima,
Tokushima 770 Japan

Abstract. The aim of this paper is to investigate the application of evolutionary approaches to the automatic design of automata in general, and Turing machines, in particular. Here, each automaton is represented directly by its state transition table and the number of states is allowed to change dynamically as evolution takes place. This approach contrasts with less natural representation methods such as trees of genetic programming, and allows for easier visualization and hardware implementation of the obtained automata. Two methods are proposed, namely, a straightforward, genetic-algorithm-like one, and a more sophisticated approach involving several operators and the 1/5 rule of evolution strategy. Experiments were carried out for the automatic generation of Turing machines from examples of input and output tapes for problems of sorting, unary arithmetic, and language acceptance, and the results indicate the feasibility of the evolutionary approach. Since Turing machines can be viewed as general representations of computer programs, the proposed approach can be thought of as a step towards the generation of programs and algorithms by evolution.

1 Introduction

Recently, paradigms of *Evolutionary Computing* [3], a relatively novel area of Computer Science which employs principles of natural evolution to solve engineering problems, have been proposed as alternative methods for optimization and machine learning. Focusing on the automatic generation of automata, the literature registers approaches based on *Genetic Algorithms* (GAs) [7], *Evolutionary Programming* (EP) [5], and *Genetic Programming* (GP) [9], mainly the last two fields.

When evolving automata, application of a GA demands the design of a coding scheme to represent each automaton as a chromosome. This process often results in complex representations and in the need of customized operations to avoid the generation of chromosomes that cannot be translated into meaningful automata. Furthermore, with a few notable exceptions [6, 8], GAs operate on fixed-length chromosomes. This often poses a problem when one does not know *a priori* the optimal size of the automaton necessary to solve a given task.

On the other hand, while the core of EP methods seems to be suitable to the evolution of automata, EP has only been tested on specific problems of evolution of small-scale automata for Artificial Intelligence (AI) problems [4]. Finally, concerning GP, this method requires that the automata be mapped into

convenient tree structures, which are then evolved. Later, after evolution finishes or is terminated, it is necessary to map the obtained solutions back to the space of automata. Generally, this last operation is not straightforward.

This paper examines an alternative form of evolutionary generation of automata, in which each automaton in a population is represented by the corresponding state transition table (STT), and the number of states is allowed to change dynamically. Overall, the system resembles a customized GA in its population model, selection method and operators, but instead of chromosome representations, having each automaton denoted by its STT facilitates the interpretation of the evolution results. This is an essential property when the automata are to be analyzed or implemented in hardware. Genetic operators were devised and applied to the particular case in which each automaton is a Turing Machine (TM), chosen because of its generality as a computer model.

2 Related Research

Perhaps the first application of evolutionary techniques to generate automata was done by Fogel, Owens, and Walsh [5], who developed EP to evolve small automata to solve an environment (represented by a sequence of symbols) prediction problem. Recently, there have been new applications of the technique to evolve automata for specific problems as, for example, to solve the classic prisoner's dilemma and to model human behavior observations [1]. Studies involving the application of EP to produce more general automata have not been done so far.

The need for a chromosome representation as a string of symbols from a finite alphabet, together with the fact that virtually all GAs operate on chromosomes of fixed length, make it troublesome to evolve automata by GAs. One of the few reported applications was for the task of finding an automaton to navigate an artificial ant so as to find all the food lying along an irregular trail defined on a toroidal grid of 32×32 cells [2]. Collins and Jefferson coded each possible state and action as a binary substring and concatenated such substrings in a fixed order, in such a way that any finite state automaton could be represented by a binary string. Although it was able to find high-quality solutions for the ant problem, that approach assumes that at least a good estimate of the necessary number of states is known beforehand.

Finally, GP has been extensively applied to automatic generation of programs and automata in recent years [9]. However, GP deals with the evolution of tree-like structures, usually LISP S-expressions, combining primitive functions and operators. This creates a representation problem when trying to evolve automata, since GP is not directly applicable to automata represented by their STTs. As a consequence, it becomes difficult to translate the programs obtained by evolution into automata, which is the original problem.

We argue that an ideal evolutionary method for generation of automata should have intuitive representation, allow for easy mapping of solutions to STTs

(to facilitate analysis and hardware implementation by flip-flops and logic gates), allow for variation of the number of states, and be robust.

3 Proposed Approaches

3.1 Automata Representation and Population

Generally, an automaton M is a sequential machine specified as

$$M = (Q, \Sigma, \Delta, \delta, \lambda, q_0) \quad (1)$$

where Q is the finite set of all states, Σ and Δ are the finite sets of input and output symbols, respectively, δ is the state transition function (a function specifying the next state for each pair of current state $p \in Q$ and input symbol $a \in \Sigma$, that is, $\delta(p, a) = q \in Q$, λ is the output function defining the output symbol in the form $\lambda(p, a) = b \in \Delta$, and q_0 is the initial state.

Since automata are typically represented by their state transition tables, this seems to be the most natural representation to be adopted. In the proposed approach, each automaton is represented by an array in which each row denotes a possible current state, each column an input symbol, and each entry is a pair of the next state and the corresponding output symbol. Evolution takes place in a population Pop of pop_size machines μ . Formally, for each generation t we have

$$Pop(t) = [\mu^1(t), \mu^2(t), \mu^3, \dots, \mu^{pop\_size}(t)]. \quad (2)$$

Each automaton μ^i is represented by a matrix (table) in which each entry is a pair of the form “next state|output”, that is

$$\mu^i = [a_{jk}^i \in Q | b_{jk}^i \in \Delta] \quad (3)$$

for $i = 1, 2, \dots, pop\_size$, $j = 1, 2, \dots, n^i$, and $k = 1, 2, \dots, \dim(\Sigma)$, where n^i stands for the number of states of the i -th automaton and $\dim(\Sigma)$ gives the number of different input symbols. For consistency, it is assumed that all the states and input symbols are labeled in a fixed order, with the first row of each STT corresponding to the current state being the initial state q_0 . A population of automata of various sizes is illustrated in Fig. 1.

3.2 Population Initialization and Consistency

In the beginning, a population $Pop(0)$ of automata is generated randomly, as it is usual with evolutionary procedures. However, due to the limited size of the population, it is important that one impose limits on the size of each automaton, that is, make

$$n_{\min} \leq n^i \leq n_{\max} \quad \text{for } i = 1, 2, \dots, pop\_size. \quad (4)$$

The limits should be wide enough to allow for a global search, but small enough for the sake of efficiency. To avoid operating on unmeaningful automata,

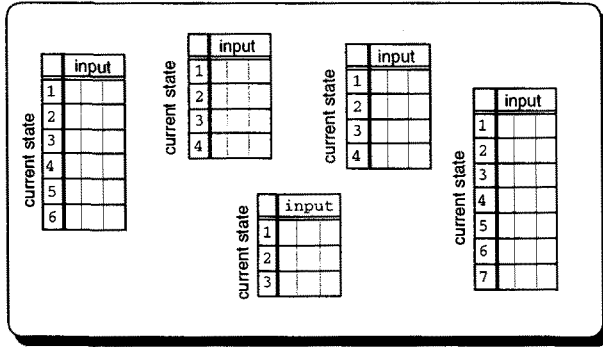


Fig. 1. Example of a population of automata with different sizes.

it is also crucial to make sure that the automata are consistent, that is, that they not include transitions to non-existing states. This is trivial when generating the initial population, since it suffices to choose the number of states n^i first, and then limiting the choice of the next states a_{jk}^i in the range $[1, n^i]$.

3.3 Evaluation Function

Careful selection of the evaluation and fitness functions are essential for attaining good performance. Such a choice is, of course, problem-dependent, but a few points should be emphasized. First, common sense and economical reasoning favors the well-known *Occam's razor* principle, by which "the simpler, the better". Accordingly, the literature registers applications of different measures to favor small, simple individuals, including the minimum description length [8] and the minimum message length [1]. This paper uses a simpler approach based on the summation of individual costs.

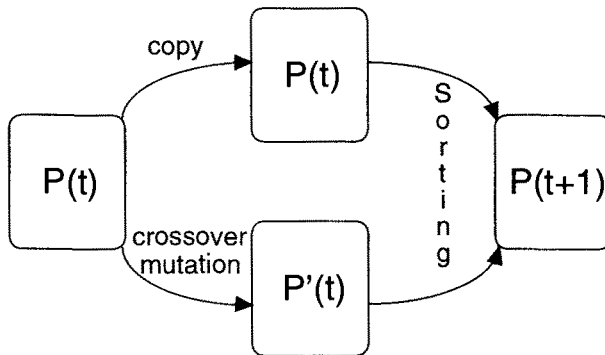


Fig. 2. Generation model.

3.4 Generation Model and Genetic Operators

A variation of the continuous generation model was employed. At the t -th generation, the population $Pop(t)$ is initially duplicated, and then an auxiliary population $Pop'(t)$ of the same size is produced by either crossover or mutation. Denoting by $0 \leq \rho \leq 1$ the ratio of crossover, in $Pop'(t)$

$$n_{\text{crossover}} = \text{int}\left(\frac{\rho \times \text{pop\_size}}{2} + 1\right) \times 2 \quad (5)$$

children automata are produced by crossover, whereas the remaining individuals result from mutation. Finally, the next population $Pop(t+1)$ results from selecting the best automata from the pool of $2 \times \text{pop\_size}$. This generation model is shown in Fig. 2.

Crossover Operator A 2-point crossover operator was applied as follows. First, a pair of automata in the population is chosen with probability proportional to their fitness, using the well-known *roulette-wheel method*. Crossover then takes place by exchanging groups of contiguous rows in the automata's STTs. Both crossover points in each automaton are chosen randomly. This is repeated until the number of children reaches the value specified in Eq. (5).

The procedure allows for dynamic variation of the number of states of the automata. However, as in GP, in practice most of the times the crossover operation results in bad individuals, and may even produce inconsistent automata which refer to non-existing states or have states which are never reached, for example. This suggests that the crossover ratio ρ should be kept small for the sake of efficient search.

Mutation Operator A simple mutation operator can be defined by selecting cells to undergo mutation and then changing some of the corresponding entry values within a certain range. Using fitness-proportional selection, at first $(\text{pop\_size} - n_{\text{crossover}})$ automata are chosen to undergo mutation. Next, a percentage p_{mut} of the STT entries in each selected automaton are chosen randomly. Finally, in each selected entry (a table cell) a single symbol is changed randomly.

4 Evolution of Turing Machines

4.1 Problem Relevance

It is a well-known fact that Turing machines (TMs) are universal models of computer programs, in the sense that any computer program can be emulated by a convenient TM. The opposite is also obviously true, since one can easily translate any given TM into a more conventional computer program.

Assume now that one wants to generate a computer program or an algorithm to solve a certain processing task, and that all that is given is a limited collection of possible inputs to the program and desired (corresponding) outputs. The

approach considered in this paper is an indirect one: 1) First, a TM tha can solve the task, learning from the example patterns, is evolved; 2) Later, by observing the structure and behavior of successfully evolved TMs while they process the data, the final computer programs or algorithms are produced. Only the first part is considered in this paper, since the latter is usually straightforward.

Therefore, generating TMs from examples is actually an approach for the automatic generation of programs and algorithms. Furthermore, since TMs are very simple automata consisting of extremely simple elements, this approach seems more natural than the one of GP, where elemental functions are assumed to be known beforehand.

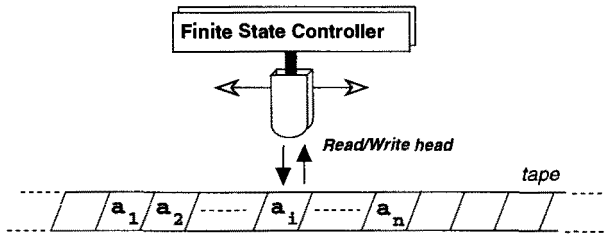


Fig. 3. Turing machine.

4.2 Turing Machines

Consider now the special case in which each automaton in the population represents a Turing Machine. A TM is a general automaton able to read and write in arbitrary positions of a tape, as shown in Fig. 3. In the basic model, there is only one unidimensional tape divided into cells of same size, each one able to contain a symbol in a specified tape symbol set Γ . The TM features a head capable of reading the symbol immediately below it, changing it, and moving to the right or left by one cell. At first, the head is positioned on the beginning (leftmost symbol) of an input string of symbols belonging to the input symbol set $\Sigma \subset \Gamma$. The set $\Gamma - \Sigma$ contains the symbol $\mathcal{B}$, the blank character. Generally the input string is surrounded by blank characters on both sides, and the tape can grow without limit in both directions. In practice, however, the tape must be limited. The head is controlled by a finite state machine of states in the set Q , initialized to the state q_0 . For each current state $p \in Q$ and input symbol $a \in \Sigma$, the “next move function” δ defines the next state q , the symbol b to replace a and the head movement, that is

$$\delta(p, a) = (q, b, D) \quad \text{where } D \in L, R. \quad (6)$$

The function δ does not necessarily has to be defined for all the combinations (p, a) , and it may also specify any or none of the terms on the right-hand side

of Eq. (6). If the direction is not specified, for example, it means that the head simply should not move. Similarly to Eq. (1), a TM can be denoted as

$$TM = (Q, \Gamma, \Sigma, \delta, q_0, \mathcal{B}, F) \quad (7)$$

where F is the set of final states. At any instant, the state of a TM can be given by its instantaneous description, as follows:

$$TM : (p, a_1 a_2 \dots \overset{\downarrow}{a_i} \dots a_n)$$

where p is the current state and the next symbol to be read is indicated by the arrow.

Table 1. Training tapes for the two-symbol sorting problem.

Pair #	Input	Target
1	bbaabaaaa	aaaaaaabbb
2	aabbbabaaa	aaaaaabbbs
3	aabaabaaaa	aaaaaaaabb
4	aaaababaaa	aaaaaaaabb
5	abaababbba	aaaaabbbbs
6	aababaaabb	aaaaaabbbs
7	bbaabaaaa	aaaaaaabbs
8	abaabaabba	aaaaaabbbs
9	aababbaaba	aaaaaabbbs
10	abaababbba	aaaaabbbbs

4.3 Experimental Results

Two cases were investigated, namely, a two-symbol sorting problem and a unary proper subtraction problem. In both cases, starting from a population of randomly-initialized single-tape TMs, the objective was to generate general TMs for the tasks. For training, each problem used a few pairs of input tapes and respective desired output tapes. It was assumed that all the non-blank input symbols were gathered contiguously on the center of tapes of length equal to 30 characters, with blank characters on both sides.

For each problem, a population of 100 TMs was generated randomly with maximum number of states equal to 10, and was set to evolve for 1,000 generations. In the temporary population, 10 TMs were generated by crossover, whereas the remaining 90 TMs were produced by mutation (that is, crossover ratio $\rho = 0.1$). The maximum number of states exchanged by crossover per time was fixed at 3, and children TMs with more than 20 states were not allowed ($n_{\max} = 20$).

The execution of each TM terminates when one of the following conditions is reached: 1) the head advances beyond the tape's limits; 2) the TM fails to stop within a maximum number of steps; 3) the TM stops acting; or 4) the TM refers to a non-existing state.

It is also important to have a high mutation rate early in the run to allow good exploration of the search space, and reduce the rate as evolution advances. Accordingly, the mutation rate p_{mut} was scheduled in such a way that a symbol in each entry of each STT to undergo mutation was mutated during the first 500 generations, while only 10% of the cells had a symbol mutated during the last 100 generations.

To evaluate each TM, first a cost function was defined as a weighted sum considering the number of wrong positions with respect to the learning tapes, the difference between the number of symbols between the input and desired output tapes, the complexity of each TM (number of states, number of symbols in the STT), etc., for all input training tapes and their corresponding target output tapes. Finally, the obtained cost values were linearly coded into fitness values between 1 and 10.

Table 2. Best results for 10 runs for both the two-symbol sorting. The obtained TMs were successfully tested against a set of tapes not used for learning. The best TMs were shown to be general solutions.

Run	States	Cost	No. of Gen.	Time (s)
1	4	310	916	143
2	6	321	947	132
3	4	311	860	127
4	5	315	856	126
5	3	310	871	135
6	3	310	801	119
7	6	312	937	144
8	3	313	740	116
9	3	310	802	123
10	4	310	850	123

Two-Symbol Sorting The objective of the sorting problem is to generate an optimal TM able to sort random tapes containing the symbols a and b , that is, to learn the relationship $a < b$ from only 10 pairs of training tapes, as shown in Table 1. It is important to note the enormous dimensions of the space of TMs. Even if only 4-state TMs with 3 input-output symbols (if blanks are also included) are considered, there are $36^{12} \approx 4.7 \times 10^{18}$ machines.

Results for 10 runs are shown in Table 2. For the sorting problem, despite the different number of states and/or complexity, the best TMs in all runs were able

to sort correctly not only the 10 tapes used for learning, but also the 20 others used for testing, suggesting that general TMs were obtained. We can, therefore, conclude that, more than only generating specific TMs for a particular problem, the proposed approach succeeded in synthesizing algorithms. In fact, in this experiment the best TMs implemented the well-known “bubble sort” algorithm.

Table 3. Training tapes for the unary proper subtraction problem.

Pair #	Input	Target	Meaning
1	1111111101	1111111	$8 - 1 = 7$
2	1111111011	11111	$7 - 2 = 5$
3	1111110111	111	$6 - 3 = 3$
4	1111101111	1	$5 - 4 = 1$
5	1111011111		$4 - 5 = 0$
6	1110111111		$3 - 6 = 0$
7	1101111111		$2 - 7 = 0$
8	1011111111		$1 - 8 = 0$

Proper Unary Subtraction The second problem deals with a TM containing only 0s, 1s and blank characters. There are two strings of contiguous 1s denoting two unary numbers, in such a way that a natural number n is represented by a string of n 1s. It is assumed that there are two numbers separated by a single 0 character, and the objective is to generate a TM able to perform proper subtraction, resulting in a tape containing only a single string of 1s, corresponding to the difference between the numbers. Only 8 pairs of tapes are assumed to be available for training, as shown in Table 3. By “proper” subtraction it is meant that there should be no 1s remaining in the tape when the first operand is smaller than or equal to the second one.

The results of 10 runs are shown in Table 4. Once again the generated TMs succeeded in producing correct results for tapes other than the ones used for training, indicating that general solutions were generated.

5 Enhanced Evolutionary Approach

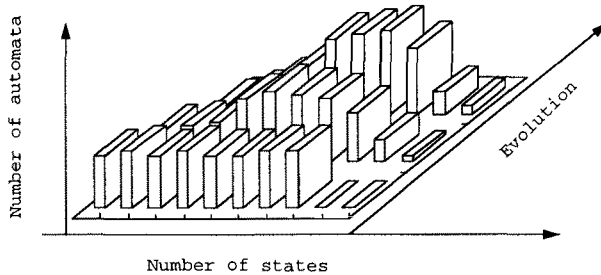
Although the approach described above seems to suffice for simple problems, the procedure does not scale up well for more complex tasks. In fact, such a non-satisfactory performance was somewhat predictable, since the proposed crossover and mutation operators described above were devised without taking into account particular characteristics of automata generation.

As described, the mutation operator performs a type of local search by changing one or a few symbols of the STT representing an automaton, while crossover

Table 4. Best results for 10 runs for the unary proper subtraction problems.

Run	States	Cost	No. of Gen.	Time (s)
1	5	3850	932	101
2	5	3851	842	115
3	3	3850	810	86
4	5	3850	837	102
5	6	3857	936	94
6	6	3850	917	97
7	10	3861	976	120
8	8	3862	995	110
9	6	3851	918	113
10	8	3859	993	124

is the only way to change the number of states, carrying out a more radical change in the search space. However, since random crossover is likely to result in unmeaningful automata, it is not an effective procedure. As a consequence, the number of states of each automaton remains constant in most cases and, therefore, valuable processing time is spent on local evolution of hopeless automata. For example, suppose that the initial population contains 10 automata for each size from 5 to 14 states, in a total of 100 automata, and that the number of states of the optimal automaton for the problem at hand is, say, 12. If the number of states does not change while evolution takes place, then only 10% of the automata in the population will ever have any chance of ever reaching the optimal configuration.

**Fig. 4.** Population shift approach.

5.1 Population Shifting Approach

In this approach, performance statistics are collected at each generation, and operators are devised in such a way to favor the appearance of automata of sizes

close to the sizes of the best performing automata in the previous generation. A pictorial description of this idea is shown in Fig. 4. Beginning from a situation in which the sizes of the automata in the population are approximately uniformly distributed in a given range, the number of automata of each size changes in such a way to concentrate the search on the region of more likely improvement. This approach is implemented by using a mutation operator, as described below.

5.2 New Mutation Operators

First of all, crossover was dropped out due to its inefficiency. The idea of crossover is based on the “building block hypothesis”, which is very unlikely to hold in the case of automata generation. Instead, three mutation operators were developed.

- Mutation¹: This is the same mutation operator described above, with the exception that now multiple changes on the same cell are allowed.
- Mutation²: This operator allows dynamic changes to the number of states of a given automaton. After an automaton is selected to undergo this type of mutation, a state is deleted from or added to it in such a way to make its size closer to the size of the best automata of the previous generation. In the case of deletion, the state to be deleted is chosen as the least visited one, all references to the deleted state are changed randomly inside the new state range, and the corresponding STT becomes one row shorter. In the case of addition of a state, the new state is appended to the end of the STT, the corresponding cells are filled with random values, and one of the cells corresponding to other states is changed to refer to the newly added state.
- Mutation³: This operator kills a given automaton and generates another one with number of states determined from the performance statistics of the previous generation. This determination is done by using a roulette-wheel with slots corresponding to possible number of states, with areas proportional to the average performance of the automata in the previous generation.

5.3 1/5 Heuristics

In the first implementation of the evolutionary approach, the rate of execution of the operators were kept constant for the sake of simplicity. A better approach (at least, in principle) is to set initial rates and have them adapted as evolution takes place, in such a way to optimize performance. Here, this is done by borrowing Rechenberg’s idea of “evolution window” [10, 11].

This idea, developed in the field of “evolution strategy”, states that successful evolution only takes place when the mutation rate falls within a narrow band named evolution window. The “1/5 rule” provides one heuristical procedure to attempt to keep the mutation rate within the evolution window.

In concrete terms, the history of the application of each mutation operator is kept up to date in terms of success rate, that is, the percentage of the total number of applications in which the resulting automaton was fitter than its parent prior to mutation. Whenever this value tends to go above 1/5, the corresponding

mutation rate is changed in such a way to make the search process more global; conversely, success rate below $1/5$ is used as an indication that search should be made more local. This $1/5$ value is close to theoretical optimal values for a few specific problems, but has been widely applied to a number of general tasks.

In our implementation, making the search more local was interpreted as increasing the probability of mutation¹ while decreasing the probabilities of both mutation² and mutation³. These changes were implemented by addition or subtraction of a fixed value. The enhanced evolutionary approach is summarized in Fig. 5.

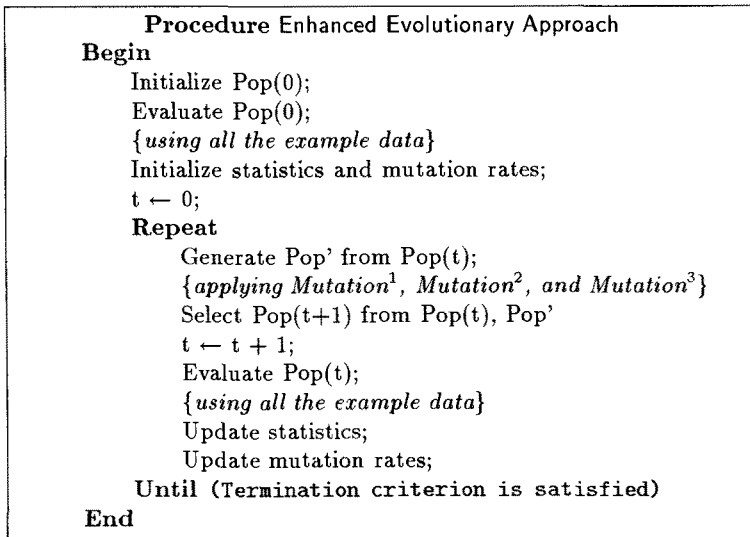


Fig. 5. Summary of the enhanced evolutionary approach.

5.4 Experimental Results

Language acceptance problems were used to evaluate the enhanced evolutionary approach, since such problems often require automata more complex than those needed for the simple problems used in the previous section. Three simple problems, namely, the recognition of a regular, a context-free, and a context-sensitive languages, were set up for experiments.

In all cases, populations of 100 automata were allowed to evolve for 1,000 generations. At the outset, the probabilities of the mutation operators were set to 70%, 25%, and 5% for mutation¹, mutation², and mutation³, respectively, and up to 50% of the cells of a given automaton were allowed to be changed. These values were later adapted by the $1/5$ heuristics.

For the three language acceptance problems, the input alphabet was set as $\{a, b, c, \mathcal{B}\}$, where $\mathcal{B}$ represents the blank space. A TM capable of recognizing a language should append a c after the input string if it belongs to the language. Otherwise, it should stop leaving the tape unaltered. In all experiments, 40 tapes were used for search, only half of them containing strings belonging to the languages to be recognized. The other tapes were generated randomly.

Target Languages The first problem was to generate TMs to recognize the regular language awb , where $w \in \{a, b\}^*$. That is, a string should be accepted if it begins with an a , ends with a b , and does not contain any blank space or c . Input strings of length 20 were employed.

The second problem used the context-free language $a^n b^n$, where $n \geq 1$ was employed, and input strings with 40 symbols each were used. Finally, the language $a^n b^n a^n$, for $n \geq 1$, was used as a simple context-sensitive language, and the input strings had 60 symbols each.

Comparative Results Experiments were performed using the simple and the enhanced evolutionary approaches, and results for 100 runs of each approach are shown in Table 5. The results indicate the success ratio in 100 runs and the average generation in which the problems were solved. As expected, the extended approach outperformed the simple one, but the results were still far from the ideal success ratio of 100%. It is reasonable to expect to improve the results by starting from TMs generated by some simple heuristical method and/or by post-processing the obtained TMs using a conventional technique.

Table 5. Comparative results for language acceptance problems.

Problem	Evolutionary Approach			
	Simple		Enhanced	
	succ. (%)	gen.	succ. (%)	gen.
awb	9	660	82	500
$a^n b^n$	41	387	62	429
$a^n b^n a^n$	1	259	38	616

6 Conclusion

In this paper, an approach to evolve automata represented by their state transition tables was proposed. By operating directly on STTs, the method allows for easy analysis and conversion of the evolution results into hardware implementable logic. Two approaches were investigated, namely, a simple GA approach and another implementation employing heuristics of known evolutionary

computing methods. Experimental results for several problems of generation of TMs from examples indicated the feasibility of the proposed methods, particularly the second one.

It may be argued that the second proposed approach can be thought of as an EP method, since there is not crossover and only sophisticated mutations. However, the proposed approach employs neither tournament selection nor Gaussian mutations which are typical of EP methods. Furthermore, the population shift approach plays a significant role in concentrating the automata search in the regions of more likely improvement.

While the results indicate it is possible to evolve TMs from examples, the proposed idea is still inefficient and needs to be enhanced. Many runs failed to solve the target problems in the number of generations allocated. This was caused in part because the number of training examples provided was very low in comparison with the size of the search space. It was necessary to limit the number of examples due to the rapid increase in the processing time, since all examples had to be processed to calculate the fitness of a given TM. One possible via for improvement is devising an estimate fitness measure that does not require the machine to process each training example until it stops or runs out of time. Other avenues for research include the use of more sophisticated initialization procedures, better operators, and post-processing using conventional techniques.

References

1. Clelland, C. H., Newlands, D. A.: PFSA modelling of behavioural sequences by evolutionary programming. In R. J. Stonier and X. H. Yu, *Complex Systems: Mechanism for Adaptation*, IEEE Press (1994) 165–172
2. Collins, R., Jefferson, D.: Ant farm: toward simulated evolution. In C. G. Langton et al., *Artificial Life II*, Addison Wesley (1991)
3. Fogel, D. B.: An introduction to simulated evolutionary optimization. *IEEE Trans. Neural Networks* 5 (1994) 3–14
4. Fogel, D. B.: Evolving behaviors in the iterated prisoner's dilemma. *Evolutionary Computation* 1 (1993) 77–97
5. Fogel, L. J., Owens, A. J., Walsh, M. J.: *Artificial Intelligence through Simulated Evolution*, John Wiley (1966)
6. Goldberg, D. E., Deb, K., Korb, B.: Don't worry, be messy. *Proc. Fourth Int. Conf. Genetic Algorithms* (1991) 24–30
7. Holland, J. H.: *Adaptation in Natural and Artificial Systems*, Univ. of Michigan Press (1975)
8. Iba, H., Kurita, T., deGaris, H., Sato, T.: System identification using structured genetic algorithms. *Proc. 5th Int. Conf. Genetic Algorithms* (1993)
9. Koza, J. R.: *Genetic Programming—On The Programming of Computers by Means of Natural Selection*, MIT Press (1992)
10. Rechenberg, I.: *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*, Frommann-Holzboog Verlag (1973)
11. Rechenberg, I.: Evolution strategy. In J. M. Zurada and R. J. Marks II and C. J. Robinson, *Computational Intelligence: Imitating Life*, IEEE Press (1994) 147–159

Theory

Genetic Algorithms: Minimal Conditions for Convergence

Alexandru Agapie

Computational Intelligence Lab., National Institute of Microtechnology
PO Box 38-160, 72225, Bucharest, Romania
E-mail: agapie@imt.pub.ro

Abstract. This paper is concerning the finite, homogenous Markov chain modeling of the binary, elitist^{*} genetic algorithm (EGA) and provides a set of minimal sufficient conditions for convergence to the global optimum. The case of a GA where each population would be allow to mutate only a small number of bits has not been covered yet by the GA's literature, although it commonly appears in practice. The main result presented here shows that the condition of the one-step transition probability by mutation between two arbitrary strings being larger than zero can be relaxed in the sense that it is also sufficient to achieve the transition by a chain of small mutations. Consequently, even one-bit mutations would be sufficient to make the GA globally convergent, because they can be chained to achieve a multi-bit mutation. All this study is performed with respect to the theory of non-negative matrices and their relationship to Markov chains.

1. Motivations for a new approach

Genetic algorithms (GAs) - as introduced by Holland in the seventies [4] - are probabilistic techniques for optimization, operating on populations of strings (called *chromosomes*) coded to represent some underlying parameter set. Some operators, called *selection*, *crossover* and *mutation* are applied to successive generations of chromosomes to create new, better valued populations.

When the schema theorem - established in [4], and consecutively revised [13] - proved to be insufficient for analyzing the GAs, most of the theoretical approaches moved onto the convergence theorems of stochastic processes, and especially to the ergodic theorems from Markov chain theory. The first attempts were concerning the theory of finite, homogenous Markov chain only [2, 3, 5, 14], but they evolved also to the study of infinite [10], or inhomogenous models [1]. However, the finite, homogenous algorithm is most commonly used in practice, so the improvement of its theoretical fundamentals is strongly required.

This study deals with the model of the binary, elitist genetic algorithm (EGA) and provides some sufficient conditions for convergence to the global optimum, regardless of the initialization. It provides a generalization - based on some realistic assumptions - of the similar results (*Theorems 6, 7*) presented in [9].

^{*} All over this paper the term *elitist* is associated to a canonical GA maintaining the best solution found over time, without using it to generate new individuals.

Against the well known hypothesis “ M (the mutation matrix) is positive” , herein is proposed a weaker assumption on the mutation operator, namely: “ M is irreducible with positive diagonal elements”. For example, the irreducibility is satisfied by assuming that mutation can invert only a small number of bits in a generation - thus not all the bits, as required by the positive M condition. Roughly speaking, this paper covers the case of a GA where each population of chromosomes would be allow to mutate at max one bit, which previous papers do not cover. One must admit that this representation is much closer to the practical GAs. Additionally, the crossover matrix will be required to have positive diagonal elements, also. It is shown that, even with this weaker assumptions, the convergence result for EGA still holds.

In fact, the convergence path from [9]:

$EGA \rightarrow M \text{ positive} \rightarrow M \text{ primitive} \rightarrow \text{Ergodic theorem for reducible EGA} \rightarrow \text{Convergence of EGA}$

is replaced by:

$EGA \rightarrow M \text{ irreducible} \rightarrow \text{Sub-optimal states are inessential} \rightarrow \text{Zero limit theorem for the sub-matrix of inessential states} \rightarrow \text{Convergence of EGA}$.

The major tool used in this study is the *Zero limit theorem for the sub-matrix of inessential states* - from [11]. As one can notice, the ergodic behavior of the EGA model is *not* available any more. However, this fact does not affect the convergence of the algorithm to the optimal states - as a sufficient condition for convergence is: “For each sub-optimal state i , the sum of transition probabilities to all the sub-optimal states tends to zero”. One will see how this statement ensures the desired convergence - using the triangular form of the transition matrix, which is still available for this approach.

A new approach in Markov chain analysis of EGA convergence must first explain its utility, as the convergence for the elitist algorithm was already proved, under the positive mutation matrix assumption [9]. At a closer look, this statement of positivity proves to be not so realistic: consider the extreme case of two states* - say i and j - placed at maximal *Hamming distance* one from the other. Recall that the Hamming distance - denoted H - is defined as the number of different bits between two binary strings. Therefore, $H_{ij}=n\ell$, where n denotes the GA's population size and ℓ - the chromosome's string length. For example let $i=(111\dots 1)$ and $j=(000\dots 0)$, both of length ℓ . One can easily compute (see e.g. [1], [9]) the transition probability from state i to state j , due to mutation only, as $m_{ij}=p_m^{n\ell}$ (p_m denotes the usual mutation probability). If the parameters are set at $p_m \sim 0.1$, $n \sim 20$ and $\ell \sim 20$ (which are very permissible assumptions) one will obtain $m_{ij} \sim 10^{-400}$. From a practical point of view, the positivity of this value is non-realistic: actually, a value of 10^{-400} is rather zero than positive. This statement is supported by statistical considerations - for a detailed discussion on this subject see [6] or [7]. Thus, the problem of relaxing the positivity

* All over this paper a state of the Markov chain will be associated to a population of chromosomes, not to a single chromosome.

assumption on M is a worthwhile challenge, and the objective of this paper is to show that this assumption is a sufficient condition for convergence, but not a necessary one.

Section 2 describes the Markov chain formalism used in the paper, and the GA model - as introduced in [9]. This formalization, together with some classical results from the Markov chain theory, are used in Section 3 for deriving the minimal conditions for convergence. A critique of this attempt completes the paper.

2 The Markov Chain Model

There are several GA approaches based on the ergodic theorem for *irreducible* transition matrix [1, 8, 12] but the idea of applying the ergodic theorem for *reducible* transition matrix is due to Rudolph [9], yielding a convergence result for the elitist model. Going deeper in the Markov chain theory, this paper states that the ergodicity (and thus the reducible form of the transition matrix) is not a necessary condition for the EGA's convergence. Anyway, it remains a sufficient condition for convergence.

This paper entirely adopts the formalism from [9], namely:

1. The optimization task is assumed as: $\max \{f(x) \mid x \in \{0, 1\}^{\ell}\}$, where f , the *fitness* function, satisfies $f(x) > 0 \forall x$;
2. The simple (canonical) GA consists of an n -tuple (called *population*) of binary strings x_i , each of them of length ℓ ;
 - the transition between successive populations is performed by applying the genetic operators of *crossover*, *mutation* and *selection* - in this order; crossover and mutation are applied with some probabilities - p_c , res. p_m - which are considered fixed all over the algorithm, in the homogenous case;
 - the canonical GA's transition matrix, denoted P , can be described as a product of the matrices corresponding to crossover, mutation and selection: $P = CMS$;
3. The EGA is modeled by a Markov chain in the following manner:
 - the state space is of cardinality $2^{(n+1)\ell}$ - a state is a possible population of the EGA, this is, the GA population supplemented by a *super individual* which does not take part in the evolutionary process. The super individual is artificially maintained from a generation to another, being subject of change only when the algorithm produces a better chromosome. The k chromosome from the current population at time t (let this population be denoted by i) can be accessed by the projection function, $\pi_k^t(i)$. For notational convenience the super individual is placed on the leftmost position in the $(n+1)$ -tuple (position 0, thus accessed by $\pi_0^t(i)$, from the population i , at time t). The transition probabilities of those states containing the same super individual are assumed to be listed one below the other in the transition matrix associated to the EGA (say P^*), and the better the super individual's fitness the higher the position of the corresponding state in P^* ;
 - since the super individual is not affected by crossover, selection and mutation, P^* (the transition matrix associated to the EGA) can be computed as:

$$P^+ = \begin{pmatrix} C & & \\ & C & \\ & & \ddots \\ & & & C \end{pmatrix} \begin{pmatrix} M & & \\ & M & \\ & & \ddots \\ & & & M \end{pmatrix} \begin{pmatrix} S & & \\ & S & \\ & & \ddots \\ & & & S \end{pmatrix} U =$$

$$= \begin{pmatrix} CMS & & \\ & CMS & \\ & & \ddots \\ & & & CMS \end{pmatrix} U,$$

where C , M and S are $2^{n'} \times 2^{n'}$ size matrices, P^+ is of size $2^{(n+1)'} \times 2^{(n+1)'}$ and U is the so called *upgrade* matrix defined as follows: let $b = \arg \max \{f(\pi_k(i)) \mid k=1, \dots, n\} \in \{0, 1\}^\ell$ denote the best individual of the population at any state i (regardless of time), excluding the super individual. Then, by definition, $u_{ij}=1$ if $f(\pi_0(i)) < f(b)$ where $j = (b, \pi_1(i), \pi_2(i), \dots, \pi_n(i))$, otherwise $u_{ij}=1$ (i and j are two possible states for the Markov chain, this is, two populations of $(n+1)$ chromosomes). Actually, the upgrade matrix is acting like a first position modifier, by copying the new best individual - in case of improvement/generation, and resting inactive otherwise (see [9] for details). Therefore, its structure is:

$$U = \begin{pmatrix} U_{11} = I & & & \\ U_{21} & U_{22} & & \\ \vdots & \vdots & \ddots & \\ U_{2^\ell,1} & U_{2^\ell,2} & \dots & U_{2^\ell,2^\ell} \end{pmatrix}.$$

U is composed of 2^ℓ lines (and columns) of $2^{n'} \times 2^{n'}$ -size blocks.

Assuming the existence of only one global optimum for the problem to be solved, it yields $U_{11}=I$ - the $2^{n'} \times 2^{n'}$ unit matrix - whereas all U_{aa} with $a \geq 2$ are unit matrices with some zero diagonal entries. Therefore, the transition matrix for the EGA can be derived as:

$$P^+ = \begin{pmatrix} P & & \\ & P & \\ & & \ddots \\ & & & P \end{pmatrix} \begin{pmatrix} I & & \\ U_{21} & U_{22} & \\ \vdots & \vdots & \ddots \\ U_{2^\ell,1} & U_{2^\ell,2} & \dots & U_{2^\ell,2^\ell} \end{pmatrix} =$$

$$= \begin{pmatrix} P & & & \\ PU_{21} & PU_{22} & & \\ \vdots & \vdots & \ddots & \\ PU_{2^\ell,1} & PU_{2^\ell,2} & \cdots & PU_{2^\ell,2^\ell} \end{pmatrix}.$$

4. The *convergence definition*: let $Z_t = \max \{f(\pi_k^t(i)), k=1,..n\}$ be a sequence of random variables representing the best fitness within a population represented by state i at time t . A GA converges to the global optimum if and only if $\lim_{t \rightarrow \infty} P\{Z_t = f^*\} = 1$, where $f^* = \max \{f(x) \mid x \in \{0, 1\}^\ell\}$ is the global optimum of the problem.

Note 1. It was proved [8], that the maximal number of different populations is actually less than 2^{n^ℓ} , namely:

$$n_{\text{pop}} = \binom{n + 2^\ell - 1}{2^\ell - 1}. \text{ But for the sake of convenience the cardinality of the state}$$

space will be maintained at the value 2^{n^ℓ} , all over this article. This assumption is equivalent to considering ordered populations of chromosomes in the GA (instead of non-ordered ones) and does not affect the convergence behavior of the EGA model, depicted below.

Definition 1.

- Let i and j be two states from a Markov chain (with transition matrix P). We say that i *leads to* j (denote $i \rightarrow j$) if there is a chain of states $(i, i_1, i_2, \dots, i_k, j)$ s.t. $p_{i,i_1} p_{i_1,i_2} \dots p_{i_k,j} > 0$. (Equivalently: it exists an integer m s.t. the (i,j) element from P^m is positive.)
- A state i from a Markov chain is said to be *essential* if $i \rightarrow j$ implies $j \rightarrow i$, for all j . Otherwise, (if there exist some j s.t. $i \rightarrow j$ but $j \not\rightarrow i$) i is said to be *inessential*.
- A non-negative matrix $A=(a_{ij})_{i,j=1,\dots,n}$ is said to be *stochastic* if $\sum_{j=1,\dots,n} a_{ij} = 1$, for each $i=1,\dots,n$.
- A square matrix $A=(a_{ij})_{i,j=1,\dots,n}$ is said to be *irreducible* if for each pair of states $i, j \leq n$ there is an integer m s.t. the (i,j) element from A^m is positive. (Equivalently, A is irreducible if $i \rightarrow j$ for each pair (i, j) .) Otherwise A is said to be *reducible* (see [11, p. 12, 18] for *definition 1. a, b, c, d*).
- A square matrix is said to be *diagonal-positive* if all its main diagonal elements are positive.
- A state i is said to be *optimal* (for the EGA problem) if the corresponding population contains the globally optimal individual on its first position - $\pi_0(i)$ (the globally optimal *super individual* from [9]). Otherwise, i is said to be *sub-optimal* (for the EGA problem).

3 Convergence results

The following lemmas set up the foundations of the EGA convergence upon the irreducible mutation matrix assumption. The first one is a simple result involving only non-negative matrices, while the second makes the connection between non-negative matrices and the GA's model - identifying the sub-optimal populations from a GA with the inessential states in the Markov chain.

Lemma 1. Let C , M and S be stochastic matrices, where M is irreducible and S , C are diagonal. Then the product CMS is irreducible. Supplementary, if M is diagonal-positive then CMS is diagonal-positive.

Proof: Let us consider first the product MS . M is irreducible implies that for each pair i, j there is a chain of states $(i, i_1, i_2, \dots, i_k, j)$ s.t. $m_{i,i_1} m_{i_1,i_2} \dots m_{i_k,j} > 0$. Since S is diagonal positive, one gets $m_{i,i_1} s_{i_1,i_1} m_{i_1,i_2} s_{i_2,i_2} \dots m_{i_k,j} s_{j,j} > 0$, too. As $m_{i',j'} s_{j',j'} \leq (ms)_{i',j'}$ for every pair i', j' , it yields $(ms)_{i,i_1} (ms)_{i_1,i_2} \dots (ms)_{i_k,j} > 0$, hence MS is irreducible. The left product between a diagonal-positive matrix (C) and an irreducible one (MS) is still irreducible (similar proof), thus CMS yields irreducible.

Supplementary, if M is diagonal-positive, (and S , C are diagonal-positive by assumption) it is obvious that CMS is diagonal-positive, too.

Lemma 2. Let P^+ be the transition matrix of the EGA. If $P=CMS$ is irreducible and diagonal-positive, then a state i is sub-optimal for the EGA problem if and only if it is inessential.

Proof: One will treat separately the two implications:

a) $\square \square i$ is sub-optimal $\Rightarrow i$ is inessential;

b) $\square \square i$ is optimal $\Rightarrow i$ is essential;

In order to prove (a), let $i=(\pi_0(i), \pi_1(i), \dots, \pi_n(i))$ be a sub-optimal state, and let $max_i = \max\{f(\pi_k(i)), k=1, \dots, n\}$. Two different situations may occur:

a.1) $f(\pi_0(i)) \geq max_i$,

a.2) $f(\pi_0(i)) < max_i$.

Case (a.1) allows two possibilities as well:

a.1.1) there is a state k s.t. $f(\pi_0(k)) < max_k$ and $p_{ik} > 0$ (note that the last inequality implies $\pi_0(k) = \pi_0(i)$). Thus, according to the definition of the upgrade matrix U , one has $u_{kj} = 1$, where $j = (b, \pi_1(i), \dots, \pi_n(i))$ and $b = \arg \max\{f(\pi_s(i)), s=1, \dots, n\}$. Hence $p_{ij}^+ \geq p_{ik} u_{kj} = p_{ik} > 0$, thus $i \rightarrow j$.

Next, we prove that $j \not\rightarrow i$: as j is placed in a class of states (by size 2^M) superior to that of i (because $f(\pi_0(j)) = max_k > f(\pi_0(i))$), and - by definition - matrix P^+ is zero above the block diagonal, there will be no possibility for $j \rightarrow i$.

a.1.2) there is a chain of states $(i, i_1, i_2, \dots, i_s, k)$ s.t. the product $p_{i,i_1} p_{i_1,i_2} \dots p_{i_s,k} > 0$, where $i_1, i_2, \dots, i_s$ satisfy $max_{i_l} \leq f(\pi_0(i))$, $\dots$, $max_{i_s} \leq f(\pi_0(i))$, and $\pi_0(i) = \pi_0(i_1) = \dots = \pi_0(i_s)$, and k satisfies $f(\pi_0(k)) < max_k$ and $p_{ik} > 0$ (as in (a.1.1)).

Consequently $u_{i1, i1}=1, u_{i2, i2}=1, \dots, u_{is, is}=1$, thus $p_{i, i1} u_{i1, i1} p_{i1, i2} u_{i2, i2} \dots u_{is, is} p_{is, k} = p_{i, i1} p_{i1, i2} \dots p_{is, k} > 0$, and this case is reduced to (a.1.1). Therefore (a.1) is fulfilled.

Case a.2) If $f(i_0) < \max_i$, let $j = (b, i_1, \dots, i_N)$ (as in a.1.1). One will see that $i \rightarrow j$, yet $j \not\rightarrow i$. Obviously $u_{ij}=1$ and, as $p_{ii} > 0$, it yields $p_{ij}^+ \geq p_{ii} u_{ij} > 0$, so $i \rightarrow j$. As i and j belong to different classes of states, the inverse does not hold, thus i is inessential.

Case (b). First, recall that at least one essential state exists (any Markov chain contains at least one essential state - see [11, p. 16]). From part (a), it yields that this state is optimal. Next, as the class of optimal states is closed (consequence of the zero- above diagonal form of P^+), one can see that any optimal state can **not** lead to an sub-optimal state. Additionally, the behavior of the Markov chain inside the class of optimal states follows the irreducible matrix CMS - thus $i \rightarrow j$ for every pair of optimal states i and j .

Note 2. The hypothesis “ P is diagonal-positive” was used in the proof of case (a.2) only.

Before moving further, let us remind a very important result from the theory of Markov chain and non-negative matrices. This theorem is presented in detail in [11], but an other form (for the so called “*non-recurrent*” states - one can prove the equivalence *non-recurrent* = *inessential*) may be found also in [6, p. 97], [7] as a corollary of *Doebelin's Formula*.

Let A be the transition matrix of a reducible Markov chain in canonical form, let Q be the square sub-matrix of A associated to the (transitions between the) inessential states and let T be the square sub-matrix associated to the essential states:

$A = \begin{pmatrix} T & 0 \\ R & Q \end{pmatrix}$. Then the following theorem holds (quoted all over the paper as the:

Zero limit theorem for the sub-matrix of inessential states).

Theorem 1. [11, p. 120] $Q^k \rightarrow 0$ elementwise as $k \rightarrow \infty$, geometrically fast.

Note 3. This result is also a major tool in the proof of the *Ergodic theorem for reducible matrices* ([7, p. 126], [9, Theorem 2]).

In the following, one will see how this result only is sufficient for the EGA convergence, ensuring the absorption of the Markov chain in the set of optimal points (which is a closed class).

The idea behind these convergence results is that even one-bit mutations would be sufficient to make the EGA globally convergent, because they can be chained to achieve a multi-bit mutation. Thus, the condition “ M is strictly positive” (i.e. the one-step transition probability by mutation between two populations being larger than zero) can be relaxed to the condition “ M is irreducible” (i.e. it is sufficient to achieve the transition by a chain of small mutations).

Theorem 2. The EGA modeled by the transition matrix P^+ , with matrices C , M , S stochastic and diagonal-positive, and M irreducible, converges to the global optimum, regardless of initialization.

Proof: According to *Lemma 1*, CMS is irreducible and diagonal-positive; by *Lemma 2*, a state from the EGA model is sub-optimal if and only if it is inessential.

Thus $P^+ = \begin{pmatrix} P & 0 \\ R & Q \end{pmatrix}$ with $P=CMS$ a square, irreducible matrix corresponding to the set of optimal states, and Q a square matrix corresponding to the set of sub-optimal (thus inessential) states.

Then $(P^+)^k = \begin{pmatrix} P^k & 0 \\ R_k & Q^k \end{pmatrix}$ and even if R_k and P^k do *not* converge to steady

state matrices as $k \rightarrow \infty$ (like in the ergodic case), the zero limit convergence for Q^k - guaranteed by *Theorem 1* - is sufficient for the Markov chain's absorption in the set of optimal states (i.e. the indices of sub-matrix P). Thus $P\{Z_k \neq f^*\} \rightarrow 0$ as $k \rightarrow \infty$, which is equivalent to $\lim_{k \rightarrow \infty} P\{Z_k = f^*\} = 1$. As in [9], the convergence is regardless of initialization (i.e. the choice of the EGA's initial population does not affect the limit behavior). This fact yields from straightforward matrix multiplication:

$p' \begin{pmatrix} A & 0 \\ & I_{n(n-m)} \end{pmatrix} = \begin{pmatrix} p' A & 0 \\ & I_{n(n-m)} \end{pmatrix}$, where p' is an n -length vector and A is an $n \times m$ matrix.

The following corollary makes the connection to the case of a more realistic GA model - as outlined in Section 1 - this is, to a GA where each population of chromosomes would be allowed to mutate at max a fixed number of bits (say T), less than the string's length.

Recall that H_{ij} stands for the Hamming distance between populations i and j .

Corollary 1. Let the S and C matrices from the Canonical GA model be stochastic and diagonal-positive. Let T , $1 \leq T \leq 2^{n'}$, be an integer. Let the M matrix be stochastic and satisfying $m_{ij}=0$ if $H_{ij} > T$ and $m_{ij} > 0$ elsewhere. Then the corresponding EGA with transition matrix P^+ converges to the global optimum.

Proof: We must prove that if M satisfies $m_{ij} > 0$ for each pair (i, j) s.t. $H_{ij} \leq T$, then:
a) M is diagonal-positive and
b) M is irreducible.

Part (a) is obvious, as long as $H_{ii} = 0$ for each i (notice that $T \geq 1$).

For (b): $m_{ij} > 0$ for each pair (i, j) satisfying $H_{ij} \leq T$. Next, let i and j be two populations with $H_{ij} = k$, $1 \leq k \leq 2^{n'}$. One can easily see that it can be found a chain $(i, i_1, i_2, \dots, i_{k-1}, k)$ s.t. $H_{i,i_1} = H_{i_1,i_2} = \dots, H_{i_{k-1},k} = 1$ and $m_{i,i_1} m_{i_1,i_2} \dots m_{i_{k-1},k} > 0$. Thus M is irreducible, so all the hypothesis from *Theorem 2* are fulfilled and the EGA will converge.

Note 3. The hypothesis on M from this corollary is a generalization of the *classical* form (i.e. positive) of the mutation matrix: for $T=2^{n_\ell}$ one will get the positivity of all the entrances in M .

Note 4. The condition C is *diagonal-positive* is not guaranteed by the assumptions in [9] - where only little attention is paid to the crossover operator. Nevertheless, this requirement is not very restrictive: for example a crossover probability $p_c < 1$ (as it is commonly used in practice) provides the diagonal-positivity of C . (This statement can be easily proved using the transition probabilities form derived in [1] for the *Three operator GA* (i.e. a GA with reproduction, mutation and crossover).)

As in [9, *Theorem 7*], the convergence result from *Theorem 2* can be adapted for the EGA copying the better solution directly after the evaluation phase (before selection). It follows straightforwardly:

Theorem 3. The EGA as in *Theorem 2*, but maintaining the best solution found over time *before* selection, converges to the global optimum, regardless of initialization.

Proof : Let $T=CM$. The transition matrix becomes:

$$P^+ = \begin{pmatrix} T & & & \\ & T & & \\ & & \ddots & \\ & & & T \end{pmatrix} U \begin{pmatrix} S & & & \\ & S & & \\ & & \ddots & \\ & & & S \end{pmatrix} =$$

$$= \begin{pmatrix} CMS & & & \\ TU_{21}S & TU_{22}S & & \\ \vdots & \vdots & \ddots & \\ TU_{2^\ell,1}S & TU_{2^\ell,2}S & \dots & TU_{2^\ell,2^\ell}S \end{pmatrix}$$

According to the conclusion of *Lemma 1*, CMS is irreducible and diagonal-positive.

Next, one can easily prove that an analogous of *Lemma 2* (but with the assumptions “CM irreducible and diagonal-positive, C, M and S diagonal-positive” - instead of “CMS is irreducible and diagonal-positive”) still holds for the new matrix model of the EGA. (The proof is quite similar to that of *Lemma 2*, so it will be omitted.) The conclusion is the same: a state from the EGA model is sub-optimal if and only if it is inessential, thus:

$$P^+ = \begin{pmatrix} P & 0 \\ R & Q \end{pmatrix} \quad \text{where } P=CMS \text{ is the square, irreducible matrix of optimal states,}$$

and Q - the square matrix of inessential states. From this point, the proof of *Theorem 2* can be reproduced identically, providing the globally optimal convergence of the algorithm.

4 Conclusions

The theoretical analysis of the binary GA relies strongly on the theory of Markov chains and non-negative matrices.

The previous papers in this field tackled the convergence behaviour of different models of GAs, yet the models proposed were not so close to a *real* GA. This study is an attempt of surpassing the gap between the theoretical properties and the *real* GA, stating that even relaxing one of the existing theoretical models (to a more realistic one) the convergence result still holds.

Against the well known hypothesis "*M (the mutation matrix) is positive*", herein is proposed a weaker assumption on the mutation operator, namely: "*M is irreducible with positive diagonal elements*". For instance, the irreducibility is satisfied by assuming that mutation can invert only a small number of bits in a generation - thus not all the bits, as required by the positive M condition. Additionally, the crossover matrix will be required to have positive diagonal elements.

Actually, this study proves that everything one needs for the elitist GA's convergence is the irreducibility of the transition matrix. So, one has to look for conditions on matrices C, M, S such that their product is irreducible. The approach given above may be seen either as:

- a different proof for the globally optimal convergence from [9], more direct, as we avoid the *Ergodic theorem* - notice that a primitive matrix is irreducible but not inversely, or as
- a generalization of the usual GA Markov chain model - regarding the non-(strictly)-positive form of the mutation matrix M,

The Markov chain modeling of the other elitist GA - the one *using* the best solution found over time to generate new individuals, not only *maintaining* it - is still in work, but one can anticipate that there will appear no major differences (both in modeling and convergence behavior) versus the above study.

A more important problem is the comparison between different types of convergence and, for the same convergence type, between different convergence rates. This objective requires the use of some constructive convergence results from the theory of probabilities and stochastic processes. The *Zero limit theorem for the sub-matrix of inessential states from a Markov chain* - hardly used in this paper - is such a powerful tool.

Acknowledgements

The author would like to thank G. Rudolph of Dortmund University for his support and for making the things clear in Markov chain modeling of GAs. The author is also grateful to the referees for their helpful comments and suggestions during the revision of the paper.

References

1. T.E. Davis and J.C. Principe, A Markov Chain Framework for the Simple Genetic Algorithm, *Evol. Comp. 1, Vol. 3* (1993) 269-288.
2. A.E. Eiben, E.H.L. Aarts and K.M. Van Hee, Global Convergence of Genetic Algorithms: a Markov Chain Analysis, in: *Parallel Problem Solving from Nature* (Springer, Berlin, 1990) 4-12.
3. D.E. Goldberg and P. Segrest, Finite Markov chain analysis of genetic algorithms, in: *Proc. ICGA '87 (Int. Conf. On Genetic Algorithms)* (Hillsdale, NJ: Lawrence Erlbaum, 1987) 1-8.
4. J.H. Holland, Adaptation in natural and artificial systems, *Ann. Arbor*, (The University of Michigan Press, 1975).
5. J. Horn, Finite Markov Chain Analysis of Genetic Algorithms with Niching, in: *Proc. ICGA '93 (Int. Conf. on Genetic Algorithms)* (1993) 110-117.
6. M. Iosifescu, *Finite Markov chains and applications* (Techn. Publ., Bucharest, 1977 - in Romanian).
7. M. Iosifescu, *Finite Markov Processes and Their Applications* (Chichester: Wiley, 1980).
8. A.E. Nix and M.D. Vose, Modeling genetic algorithms with Markov chains, *Ann. Math. and Artificial Intelligence 5* (1992) 79-88.
9. G. Rudolph, Convergence Analysis of Canonical Genetic Algorithms, *IEEE Trans. on NN. 1, Vol. 5* (1994) 98-101.
10. X. Qi and F. Palmieri, Theoretical Analysis of Evolutionary Algorithms, parts I-II, *IEEE Trans. on NN. 1, Vol. 5* (1994) 102-129.
11. E. Seneta, *Non-negative Matrices and Markov Chains*, 2nd edition, (Springer, New York, 1981).
12. J. Suzuki, A Markov Chain Analysis on A Genetic Algorithm, in: *Proc. ICGA '93 (Int. Conf. on Genetic Algorithm, Urbana Illinois)* (Morgan Kaufmann, 1993) 46-153.
13. Y. Uesaka, Convergence of algorithm and the schema theorem in genetic algorithms, in: *Proc. ICANGA 1995* 210-213.
14. M.D. Vose, Modeling Simple Genetic Algorithms, in: D. Whitley, ed., *Foundations of Genetic Algorithms II* (San Mateo, CA: Morgan Kaufmann, 1993) 63-73.

A Template For Scatter Search And Path Relinking

Fred Glover

School of Business, CB 419
University of Colorado
Boulder, CO 80309-0419, USA

fred.glover@colorado.edu

Abstract. Scatter search and its generalized form called path relinking are evolutionary methods that have recently been shown to yield promising outcomes for solving combinatorial and nonlinear optimization problems. Based on formulations originally proposed in the 1960s for combining decision rules and problem constraints, these methods use strategies for combining solution vectors that have proved effective for scheduling, routing, financial product design, neural network training, optimizing simulation and a variety of other problem areas. These approaches can be implemented in multiple ways, and offer numerous alternatives for exploiting their basic ideas. We identify a template for scatter search and path relinking methods that provides a convenient and "user friendly" basis for their implementation. The overall design can be summarized by a small number of key steps, leading to versions of scatter search and path relinking that are fully specified upon providing a handful of subroutines. Illustrative forms of these subroutines are described that make it possible to create methods for a wide range of optimization problems.

This research was supported in part by the Air Force Office of Scientific Research Grant #F49620-97-1-0271.

Table of Contents

1 Introduction	3
2 Foundations Of Scatter Search And Path Relinking	4
1.1 Scatter Search	4
1.2 Path Relinking	6
3 Outline Of The Scatter Search/Path Relinking Template	7
4 Diversification Generator	10
4.1 Diversification Generators for Zero-One Vectors	10
4.2 A Sequential Diversification Generator	12
4.3 Diversification Generator for Permutation Problems	14
4.4 Additional Role for the Diversification Generator	15
5 Maintaining And Updating The Reference Set	15
5.1 Notation and Initialization	16
6 Choosing Subsets Of The Reference Solutions	19
6.1 Generating the Subsets of Reference Solutions	22
6.2 Methods for a Dynamic RefSet	23
6.3 Arrays for the Subset Generation Method	24
6.4 Subset Generation Method	24
7 Improvement Method	30
7.1 Avoiding Duplications	33
7.2 Move Descriptions	34
7.3 Definitions of 1-moves and the Composition of M	36
7.4 Advanced Improvement Alternatives	37
8 Conclusions	40
APPENDIX 1 Construction-by-Objective: Mixed Integer and Nonlinear Optimization	44
APPENDIX 2: Checking for Duplicate Solutions	49

1 Introduction

Scatter search and path relinking have recently been investigated in a number of studies, disclosing the promise of these methods for solving difficult problems in discrete and nonlinear optimization. Recent applications of these methods (and of selected component strategies within these methods) include:¹

Vehicle Routing – Rochat and Taillard (1995); Taillard (1996)
 Quadratic Assignment – Cung *et al.* (1996)
 Financial Product Design – Consiglio and Zenios (1996)
 Neural Network Training – Kelly, Rangaswamy and Xu (1996)
 Job Shop Scheduling – Yamada and Nakano (1996)
 Flow Shop Scheduling – Yamada and Reeves (1997)
 Graph Drawing – Laguna and Marti (1997)
 Linear Ordering – Laguna, Marti and Campos (1997)
 Unconstrained Continuous Optimization – Fleurent *et al.* (1996)
 Bit Representation – Rana and Whitley (1997)
 Optimizing Simulation – Glover, Kelly and Laguna (1996)
 Complex System Optimization – Laguna (1997)

We propose a template for generating a broad class of scatter search and path relinking methods, with the goal of creating versions of these approaches that are convenient to implement. Our design is straightforward, and can be readily adapted to optimization problems of diverse structures. We offer specific comments relating to multidimensional knapsack problems, graph partitioning problems, linear and nonlinear zero-one problems, mixed integer programming problems and permutation problems.

From the standpoint of classification, scatter search and path relinking may be viewed as evolutionary algorithms that construct solutions by combining others, and derive their foundations from strategies originally proposed for combining decision rules and constraints. The goal of these procedures is to enable a solution procedure based on the combined elements to yield better solutions than one based only on the original elements.

Historically, the antecedent strategies for combining decision rules were introduced in the context of scheduling methods, to obtain improved local decision rules for job shop scheduling problems. New rules were generated by creating numerically weighted combinations of existing rules, suitably restructured so that their evaluations embodied a common metric. The approach was motivated by the supposition that information about the relative desirability of alternative choices is captured in different forms by different rules, and that this information can be exploited more

¹ The References Section contains website listings where abstracts and/or copies can be obtained for a number of the references cited in this paper.

effectively when integrated by means of a *combination mechanism* than when treated by the standard strategy of selecting different rules one at a time, in isolation from each other. The decision rules created from such combination strategies produced better empirical outcomes than standard applications of local decision rules, and also proved superior to a “probabilistic learning approach” that selected different rules probabilistically at different junctures, but without the integration effect provided by generating combined rules (Crowston, et al., 1963; Fisher and Thompson, 1963).

The associated procedures for combining constraints likewise employed a mechanism of generating weighted combinations, in this case applied in the setting of integer and nonlinear programming, by introducing nonnegative weights to create new constraint inequalities, called *surrogate constraints* (Glover, 1965, 1968). The approach isolated subsets of constraints that were gauged to be most critical, relative to trial solutions based on the surrogate constraints, and produced new weights that reflected the degree to which the component constraints were satisfied or violated.

A principal function of surrogate constraints, in common with the approaches for combining decision rules, was to provide ways to evaluate choices that could be used to generate and modify trial solutions. From this foundation, a variety of heuristic processes evolved that made use of surrogate constraints and their evaluations. Accordingly, these processes led to the complementary strategy of combining solutions, as a *primal* counterpart to the *dual* strategy of combining constraints², which became manifest in scatter search and its path relinking generalization.

2 Foundations Of Scatter Search And Path Relinking

1.1 Scatter Search

The scatter search process, building on the principles that underlie the surrogate constraint design, is organized to (1) capture information not contained separately in the original vectors, (2) take advantage of auxiliary heuristic solution methods to evaluate the combinations produced and to generate new vectors.

The original form of scatter search (Glover, 1977) may be sketched as follows.

Scatter Search Procedure

1. Generate a starting set of solution vectors by heuristic processes designed for the problem considered, and designate a subset of the best vectors to be *reference solutions*. (Subsequent iterations of this step, transferring from Step 3 below, incorporate advanced starting solutions and best solutions from previous history as candidates for the reference solutions.)

² Surrogate constraint methods give rise to a mathematical duality theory associated with their role as relaxation methods for optimization (e.g., see Greenberg and Pierskalla 1970, 1973; Glover, 1975; Karwan and Rardin, 1976, 1979; Freville and Plateau, 1986, 1993).

2. Create new points consisting of linear combinations of subsets of the current reference solutions. The linear combinations are:
 - (a) chosen to produce points both inside and outside the convex regions spanned by the reference solutions.
 - (b) modified by generalized rounding processes to yield integer values for integer-constrained vector components.
1. Extract a collection of the best solutions generated in Step 2 to be used as starting points for a new application of the heuristic processes of Step 1. Repeat these steps until reaching a specified iteration limit.

Three particular features of scatter search deserve mention. First, the linear combinations are structured according to the goal of generating weighted centers of selected subregions, allowing for nonconvex combinations that project these centers into regions external to the original reference solutions. The dispersion pattern created by such centers and their external projections is particularly useful for mixed integer optimization. (See Appendix 1 for specific procedures in this context.) Second, the strategies for selecting particular subsets of solutions to combine in Step 2 are designed to make use of clustering, which allows different types of strategic variation by generating new solutions “within clusters” and “across clusters”. Third, the method is organized to use supporting heuristics that are able to start from infeasible solutions, and hence which remove the restriction that solutions selected as starting points for re-applying the heuristic processes must be feasible.³

In sum, scatter search is founded on the following premises.

- (P1) Useful information about the form (or location) of optimal solutions is typically contained in a suitably diverse collection of elite solutions.
- (P2) When solutions are combined as a strategy for exploiting such information, it is important to provide for combinations that can extrapolate beyond the regions spanned by the solutions considered, and further to incorporate heuristic processes to map combined solutions into new points. (This serves to provide both diversity and quality.)
- (P3) Taking account of multiple solutions simultaneously, as a foundation for creating combinations, enhances the opportunity to exploit information contained in the union of elite solutions.

The fact that the heuristic processes of scatter search are not restricted to a single uniform design, but represent a varied collection of procedures, affords additional strategic possibilities.

³ An incidental feature – that has more than incidental implications – is the incorporation of general (mixed integer) solution vectors, in contrast to a reliance on binary representations. By contrast, the focus on binary representations that held sway in genetic algorithm proposals until the mid to late 1980s can be shown to create “information gaps” for combining solutions (Glover, 1994a). The problems of *distortion* in binary-based GAs are therefore not surprising.

1.2 Path Relinking

From a spatial orientation, the process of generating linear combinations of a set of reference solutions may be characterized as generating paths between and beyond these solutions (where solutions on such paths also serve as sources for generating additional paths). This leads to a broader conception of the meaning of creating *combinations* of solutions. By natural extension, such combinations may be conceived to arise by generating paths between and beyond selected solutions in neighborhood space, rather than in Euclidean space (Glover 1989, 1994b).

This conception is reinforced by the fact that a path between solutions in a neighborhood space will generally yield new solutions that share a significant subset of attributes contained in the parent solutions, in varying "mixes" according to the path selected and the location on the path that determines the solution currently considered. The character of such paths is easily specified by reference to solution attributes that are added, dropped or otherwise modified by the moves executed in neighborhood space. Examples of such attributes include edges and nodes of a graph, sequence positions in a schedule, vectors contained in extreme point basic solutions, and values of variables and functions of variables. To generate the desired paths, it is only necessary to select moves that perform the following role: upon starting from an *initiating solution*, the moves must progressively introduce attributes contributed by a *guiding solution* (or reduce the distance between attributes of the initiating and guiding solutions). The process invites variation by interchanging the roles of the initiating and guiding solutions, and also by inducing each to move simultaneously toward the other as a way of generating combinations.⁴

Such an incorporation of attributes from elite parents in partially or fully constructed solutions was foreshadowed by another aspect of scatter search, embodied in an accompanying proposal to assign preferred values to subsets of *strongly determined* and *consistent* variables. The theme is to isolate assignments that frequently (and influentially) occur in high quality solutions, and then to introduce compatible subsets of these assignments into other solutions that are generated or amended by heuristic procedures. (Such a process implicitly relies on a simple form of frequency based memory to identify and exploit variables that qualify as consistent, and thereby provides a bridge to associated tabu search ideas discussed in later sections.)

Multiparent path generation possibilities emerge in path relinking by considering the combined attributes provided by a set of guiding solutions, where these attributes are weighted to determine which moves are given higher priority. The generation of such paths in neighborhood space characteristically "relinks" previous points in ways not achieved in the previous search history, hence giving the approach its name.

⁴ Variants of path relinking that use constructive and destructive neighborhoods, called *vocabulary building* approaches, produce strategic combinations of partial solutions (or "solution fragments") as well as of complete solutions. The organization of vocabulary building permits the goal for combining the solution components to be expressed as an optimization model in a number of contexts, with the added advantage of allowing exact methods to be used to generate the moves (see, e.g., Glover and Laguna, 1997).

Neighborhoods for these processes may differ from those used in other phases of search. For example, they may be chosen to *tunnel through* infeasible regions that may be avoided by other neighborhoods. Such possibilities arise because feasible guiding points can be coordinated to assure that the process will re-enter the feasible region, with out danger of becoming "lost." The ability of neighborhood structures to capture contextual features additionally provides a foundation for incorporating domain-specific knowledge about different classes of problems, thus enabling path relinking to exploit such knowledge directly.⁵

3 Outline Of The Scatter Search/Path Relinking Template

The goal of this paper is to provide a template for implementing both scatter search and path relinking that takes advantage of their common characteristics and principles. Such a template is based upon specific subroutines of the following types:

- (1) A Diversification Generator: to generate a collection of diverse trial solutions, using an arbitrary trial solution (or seed solution) as an input.
- (2) An Improvement Method: to transform a trial solution into one or more enhanced trial solutions. (Neither the input nor output solutions are required to be feasible, though the output solutions will more usually be expected to be so. If no improvement of the input trial solution results, the "enhanced" solution is considered to be the same as the input solution.)
- (3) A Reference Set Update Method: to build and maintain a Reference Set consisting of the b best solutions found (where the value of b is typically small, e.g., between 20 and 40), organized to provide efficient accessing by other parts of the method.
- (4) A Subset Generation Method: to operate on the Reference Set, to produce a subset of its solutions as a basis for creating combined solutions.
- (5) A Solution Combination Method: to transform a given subset of solutions produced by the Subset Generation Method into one or more combined solution vectors.

We provide illustrative designs for each of these subroutines except the last, which will typically vary according to the context. The processes for combining solutions in scatter search and path relinking, as embodied in (5), have been discussed at some length in the references previously cited, and especially in Glover and Laguna (1997). Consequently, we focus instead on other features, which deserve greater attention than they have been accorded in the past. Our purpose in particular is to identify forms of

⁵ This may be contrasted with the GA crossover concept, which lacks a unifying design to handle applications that are not well served by the original proposals for exchanging components of binary vectors. The crossover model, which has significantly influenced the popular view of the meaning of combining vectors, compels one of two outcomes: (a) domain-specific knowledge must be disregarded (a property of GAs once regarded to be a virtue); (b) amended versions of crossover must repeatedly be devised, without a guiding principle for structuring the effort to accommodate new problems. The second outcome has led to frequent reliance on ad hoc constructions, as noted by Reeves (1997) and Muhlenbein (1997).

the subroutines (1) to (4), preceding, that can be usefully adapted to a variety of settings, and that can be integrated to produce an effective overall method.

We specify the general template in outline form as follows. This template reflects the type of design often used in scatter search and path relinking, but is not intended to represent the only way to organize and implement such methods. (E.g., see Glover (1994a, 1995), for other alternatives.)

SS/PR Template

Initial Phase

1. (*Seed Solution Step.*) Create one or more seed solutions, which are arbitrary trial solutions used to initiate the remainder of the method.
2. (*Diversification Generator.*) Use the Diversification Generator to generate diverse trial solutions from the seed solution(s).
3. (*Improvement and Reference Set Update Methods.*) For each trial solution produced in Step 2, use the Improvement Method to create one or more enhanced trial solutions. During successive applications of this step, maintain and update a Reference Set consisting of the b best solutions found.
4. (*Repeat.*) Execute Steps 2 and 3 until producing some designated total number of enhanced trial solutions as a source of candidates for the Reference Set.

Scatter Search/Path Relinking Phase

5. (*Subset Generation Method.*) Generate subsets of the Reference Set as a basis for creating combined solutions.
6. (*Solution Combination Method.*) For each subset X produced in Step 5, use the Solution Combination Method to produce a set $C(X)$ that consists of one or more combined solutions. Treat each member of $C(X)$ as a trial solution for the following step.
7. (*Improvement and Reference Set Update Methods.*) For each trial solution produced in Step 6, use the Improvement Method to create one or more enhanced trial solutions, while continuing to maintain and update the Reference Set.
8. (*Repeat.*) Execute Steps 5-7 in repeated sequence, until reaching a specified cutoff limit on the total number of iterations.

Table 1, following, summarizes the relationships between sources of input and output solutions in the preceding template.

<u>Source of Input Solutions</u>	<u>Source of Output Solutions</u>
Arbitrary seed solutions	Diversification Generator
Diversification Generator	Improvement Method
Improvement Method	Reference Set Update Method
Reference Set Update Method	Subset Generation Method
Subset Generation Method	Solution Combination Method
Solution Combination Method	Improvement Method

Table 1: Input/Output Links

Scatter search and path relinking are often implemented in connection with tabu search (TS), and their underlying ideas share a significant intersection with the TS perspective. A principal element of this perspective is its emphasis on establishing a strategic interplay between intensification and diversification. In the original scatter search design (Glover, 1977), which carries over to the present template, intensification is achieved by:

- (a) the repeated use of the Improvement Method as a basis for refining the solutions created (from combinations of others);
- (b) maintaining the Reference Set to consist of the highest quality solutions found; and
- (c) choosing subsets of the Reference Set and uniting their members by strategies that reinforce the goal of generating good solutions (as opposed to relying on mating and “crossover” schemes that are heavily based on randomization).

In the mid to late 1980s, a number of the elements proposed earlier in scatter search began to be introduced in hybrid variants of GA procedures. Consequently, some of the current descendants of these hybrid approaches appear to have a structure similar to the outline of the SS/PR Template. Nevertheless, significant differences remain, due to perspectives underlying scatter search and path relinking that have not become incorporated into the GA hybrids. These are particularly reflected in the concepts underlying intensification and diversification, which will be elaborated in subsequent discussions.

The remaining sections are devoted to providing illustrative forms of the subroutines that support the foregoing template. We first focus on the diversification generator, followed by the method for updating the reference set and then the method for choosing subsets of the reference solutions. Finally we examine the issue of specifying an improvement method, and identify an approach that likewise embodies ideas that have not been adequately considered in the literature.

4 Diversification Generator

We indicate two simple types of diversification generators, one for problems that can be formulated in a natural manner as optimizing a function of zero-one variables, and the other for problems that can more appropriately be formulated as optimizing a permutation of elements. In each of these instances we disregard the possible existence of complicating constraints, in order to provide a simple representation of the basic ideas. However, these methods can also be used in the presence of such constraints by using the solutions generated as targets for creating solutions that satisfy the additional requirements of feasibility. This can be done by applying neighborhood procedures (including those that use constructive or destructive neighborhoods) to insure the preservation or attainment of feasibility, while utilizing evaluations that give preference to moves which approach the targets. In addition, Appendix 1 shows how zero-one solution generators can be embedded in a method to create diversified collections of feasible points for mixed integer programming and nonlinear optimization problems.

These approaches embody the tabu search precept that diversification is not the same as randomization. In this respect, they differ from the randomized approaches for creating variation that are typically proposed in other types of evolutionary approaches. The goal of diversification is to produce solutions that differ from each other in significant ways, and that yield productive (or “interesting”) alternatives in the context of the problem considered. By contrast, the goal of randomization is to produce solutions that may differ from each other in any way (or to any degree) at all, as long as the differences are entirely “unsystematic”. From the tabu search viewpoint, a reliance on variation that is strategically generated can offer advantages over a reliance on variation that is distinguished only by its unpredictability.

4.1 Diversification Generators for Zero-One Vectors

We let x denote an n -vector each of whose components x_j receives the value 0 or 1. The first type of diversification generator we consider takes such a vector x as its seed solution, and generates a collection of solutions associated with an integer $h = 1, 2, \dots, h^*$, where $h^* \leq n - 1$. (Recommended is $h^* \leq n/5$.)

We generate two types of solutions, x' and x'' , for each value of h , by the following rule:

Type 1 Solution: Let the first component x'_1 of x' be $1 - x_1$, and let $x'_{1+kh} = 1 - x_{1+kh}$ for $k = 1, 2, 3, \dots, k^*$, where k^* is the largest integer satisfying $k^* \leq n/h$. Remaining components of x' equal 0.

To illustrate for $x = (0,0,\dots,0)$: The values $h = 1, 2$ and 3 respectively yield $x' = (1,1,\dots,1)$, $x' = (1,0,1,0,1 \dots)$ and $x' = (1,0,0,1,0,0,1,0,0,1,\dots)$. This progression

suggests the reason for preferring $h^* \leq n/5$. As h becomes larger, the solutions x' for two adjacent values of h differ from each other proportionately less than when h is smaller. An option to exploit this is to allow h to increase by an increasing increment for larger values of h .

Type 2 Solution: Let x'' be the complement of x' .

Again to illustrate for $x = (0,0,\dots,0)$: the values $h = 1, 2$ and 3 respectively yield $x'' = (0,0,\dots,0)$, $x'' = (0,1,0,1,\dots)$ and $x'' = (0,1,1,0,1,1,0,\dots)$. Since x'' duplicates x for $h = 1$, the value $h = 1$ can be skipped when generating x'' .

We extend the preceding design to generate additional solutions as follows. For values of $h \geq 3$ the solution vector is shifted so that the index 1 is instead represented as a variable index q , which can take the values $1, 2, 3, \dots, h$. Continuing the illustration for $x = (0,0,\dots,0)$, suppose $h = 3$. Then, in addition to $x' = (1,0,0,1,0,0,1,\dots)$, the method also generates the solutions given by $x' = (0,1,0,0,1,0,0,1,\dots)$ and $x' = (0,0,1,0,0,1,0,0,1,\dots)$, as q takes the values 2 and 3 .

The following pseudo-code indicates how the resulting diversification generator can be structured, where the parameter *MaxSolutions* indicates the maximum number of solutions desired to be generated. Comments within the code appear in *italics*, enclosed within parentheses.

First Diversification Generator for Zero-One Solutions.

NumSolutions = 0

For $h = 1$ to h^*

Let $q^* = 1$ if $h < 3$, and otherwise let $q^* = h$

(q^* denotes the value such that q will range from 1 to q^* . We set $q^* = 1$ instead of $q^* = h$ for $h < 3$ because otherwise the solutions produced for the special case of $h < 3$ will duplicate other solutions or their complements.)

For $q = 1$ to q^*

let $k^* = (n-q)/h$ <rounded down>

For $k = 1$ to k^*

$$x'_{q+kh} = 1 - x_{q+kh}$$

End k

If $h > 1$, generate x'' as the complement of x'

(x' and x'' are the current output solutions.)

NumSolutions = *NumSolutions* + 2 (or + 1 if $h = 1$)

If *NumSolutions* $\geq$ *MaxSolutions*, then stop generating solutions.

End q

End h

The number of solutions x' and x'' produced by the preceding generator is approximately $q^*(q^*+1)$. Thus if $n = 50$ and $h^* = n/5 = 10$, the method will generate

about 110 different output solutions, while if $n = 100$ and $h^* = n/5 = 20$, the method will generate about 420 different output solutions.

Since the number of output solutions grows fairly rapidly as n increases, this number can be limited, while creating a relatively diverse subset of solutions, by allowing q to skip over various values between 1 and q^* . The greater the number of values skipped, the less “similar” the successive solutions (for a given h) will be. Also, as previously noted, h itself can be incremented by a value that differs from 1.

For added variation:

If further variety is sought, the preceding approach can be augmented as follows. Let $h = 3, 4, \dots, h^*$, for $h \leq n - 2$ (preferably $h^* \leq n/3$). Then for each value of h , generate the following solutions.

Type 1A Solution: Let $x'_1 = 1 - x_1$ and $x'_2 = 1 - x_2$. Thereafter, let $x'_{1+kh} = 1 - x_{1+kh}$ and let $x'_{2+kh} = 1 - x_{2+kh}$, for $k = 1, 2, \dots, k^*$, where k^* is the largest integer such that $2 + kp \leq n$. All other components of x' are the same as in x .

Type 2A Solution: Create x'' as the complement of x' , as before.

Related variants are evident. The index 1 can also be shifted (using a parameter q) in a manner similar to that indicated for solutions of type 1 and 2.

4.2 A Sequential Diversification Generator

The concept of diversification invites a distinction between solutions that differ from a given solution (e.g., a seed solution) and those that differ from each other.⁶ Our preceding comments refer chiefly to the second type of diversification, by their concern with creating a collection of solutions whose members exhibit certain contrasting features.

Diversification of the first type can be emphasized in the foregoing design by restricting attention to the complemented solutions denoted by x'' when h becomes larger than 2. In general, diversification of the second type is supported by complementing larger numbers of variables in the seed solution. We stress that this type of diversification by itself is incomplete, and the relevance of diversification of the first type is important to heed in many situations.

Approaches that combine characteristics of both types of diversification can be founded on the ideas of *sequential diversification* (Glover and Laguna, 1997). A “diverse sequence” can be composed by a rule that selects each successive vector to

⁶ These distinctions have often been overlooked by the genetic algorithm community, in spite of the emergence of GA hybrids that take advantage of tabu search to obtain improved outcomes. The interplay between intensification and diversification is sometimes further obscured by confusing it with the control theory notion of “exploitation versus exploration,” which GAs have traditionally adopted. The nature and consequences of these differing ideas are discussed, for example, in Glover and Laguna (1997).

maximize the minimum distance from all vectors previously considered, according to a chosen distance metric. (There can sometimes be many vectors that qualify to be chosen as the next member of a partially constructed sequence by such a criterion. Ties can be broken by a perturbed distance function where distances to vectors that appear later in the partial sequence are considered smaller than distances to vectors that appear earlier in the sequence.) Sequential diversification makes it possible to generate solutions that differ from a *set* of solutions (such as a set of solutions examined during a search process), rather than those that differ from a single solution, by considering the members of the set to be first members of the sequence.

A sequential diversification generator for 0-1 vectors that follows the prescription to maximize the minimum distance from preceding vectors is embodied in the following procedure. We say that a solution y complements x over an index set J if $y_j = 1 - x_j$ for $j \in J$ and $y_j = x_j$ for $j \notin J$.

Sequential (Max/Min) Diversification Generator

1. Designate the seed solution x and its complement to be the first two solutions generated.
2. Partition the index set $N = \{1, \dots, n\}$ for x into two sets N' and N'' that, as nearly as possible, contain equal numbers of indexes. Create the two solutions x' and x'' so that x' complements x over N' and x'' complements x over N'' .
3. Define each subset of N that is created by the most recent partition of N to be a *key subset*. If no key subset of N contains more than 1 element, stop. Otherwise, partition each key subset S of N into two sets S' and S'' that contain, as nearly as possible, equal numbers of elements. Balance the overall collection of partitions so that the number with $|S'| > |S''|$ also equals the number with $|S'| < |S''|$ as nearly as possible. (This includes the case where S may contain only 1 element, and hence where S' or S'' is empty.)
4. Let N' be the union of all subsets S' and let N'' be the union of all subsets S'' . Create the complementary solutions x' and x'' relative to N' and N'' as in Step 2, and then return to Step 3.

The foregoing process generates approximately $2(1 + \log n)$ solutions. If n is a power of 2, every solution produced maximizes the minimum Hamming distance from all previous solutions generated. (This maxmin distance, measured as the number of elements by which the solutions differ, is $n/2$ for every iteration after the first two solutions are generated in Step 1. Such a maxmin value is also approximately achieved when n is not a power of 2.)

Useful variations result by partitioning N in different ways, but different partitions do not always produce different results. For example, if the initial rule is to partition each set so that the first member of the partition only contains indexes that are smaller than the minimum index contained in the second, then a corresponding rule that first

reindexes the last fourth of the elements of N to become instead the second fourth of these elements will produce exactly the same set of solutions as the initial rule.

The foregoing 0-1 Diversification Generator and the one first described can additionally be applied to generate solutions that differ from a set of solutions by using a TS frequency-based approach. An example occurs by weighting the vectors of the set according to their importance, and creating a combined vector of weighted frequencies by summing the weighted vectors. The resulting vector can then be transformed into a seed solution for the Diversification Generator by mapping each of its components into 0 or 1, according to whether the value of the component lies below or above a chosen threshold.

The next section offers another type of organization to assure that new solutions differ from the seed solution as well as from each other.

4.3 Diversification Generator for Permutation Problems

Although permutation problems can be formulated as 0-1 problems, they constitute a special class that preferably should be treated somewhat differently. Assume that a given trial permutation P used as a seed is represented by indexing its elements so they appear in consecutive order, to yield $P = (1, 2, \dots, n)$. Define the subsequence $P(h:s)$, where s is a positive integer between 1 and h , to be given by $P(h:s) = (s, s+h, s+2h, \dots, s+rh)$, where r is the largest nonnegative integer such that $s+rh \leq n$. Then define the permutation $P(h)$, for $h \leq n$, to be $P(h) = (P(h:h), P(h:h-1), \dots, P(h:1))$.

Illustration:

Suppose P is given by

$$P = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18)$$

If we choose $h = 5$, then $P(5:5) = (5, 10, 15)$, $P(5:4) = (4, 9, 14)$, $P(5:3) = (3, 8, 13, 18)$, $P(5:2) = (2, 7, 12, 17)$, $P(5:1) = (1, 6, 11, 16)$, to give:

$$P(5) = (5, 10, 15, 4, 9, 14, 3, 8, 13, 18, 2, 7, 12, 17, 1, 6, 11, 16)$$

Similarly, if we choose $h = 4$ then $P(4:4) = (4, 8, 12, 16)$, $P(4:3) = (3, 7, 11, 15)$, $P(4:2) = (2, 6, 10, 14, 18)$, $P(4:1) = (1, 5, 9, 13, 17)$ to give:

$$P(4) = (4, 8, 12, 16, 3, 7, 11, 15, 2, 6, 10, 14, 18, 1, 5, 9, 13, 17)$$

In this illustration we have allowed h to take the two values closest to the square root of n . These values are interesting based on the fact that, when h equals the square root of n , the minimum relative separation of each element from each other element in the new permutation is maximum, compared to the relative separation of exactly 1 in the permutation P . In addition, other useful types of separation result, and become more pronounced for larger values of n .

In general, for the goal of generating a diverse set of permutations, preferable values for h range from 1 to $n/2$. We also generate the reverse of the preceding permutations, denoted by $P^*(h)$, which we consider to be more interesting than $P(h)$. The preference of $P^*(h)$ to $P(h)$ is greater for smaller values of h . For example, when $h = 1$, $P(h) = P$ and $P^*(h)$ is the reverse of P . (Also, $P(n) = P^*(1)$.) In sum, we

propose a Diversification Generator for permutation problems to be one that generates a subset of the collection $P(h)$ and $P^*(h)$, for $h = 1$ to $n/2$ (excluding $P(1) = P$).

4.4 Additional Role for the Diversification Generator

A diversification generator that takes one of the forms indicated above can be used to modify the balance between intensification and diversification in the SS/PR Template. In particular, a stronger diversification emphasis can be effected by means of optional additions to Steps 6 and 7, which we denote by 6A and 7A, as follows.

6A. (*Pre-Improvement Diversification.*) Use selected trial solutions produced in Step 6 as seeds to produce additional trial solutions by the Diversification Generator.

7A. (*Post-Improvement Diversification.*) Similarly, use selected trial solutions produced in Step 7 to produce additional trial solutions by the Diversification Generator, where each resulting trial solution in this case is immediately subjected again to Step 7 to produce an enhanced solution.

The provision to apply Step 7 again in the Post-Improvement Diversification is not required in the Pre-Improvement Diversification because Step 6 (and hence 6A) is directly followed by Step 7 in the SS/PR Template. Typically the optional steps 6A and 7A will be used to generate a single additional trial solution for each trial solution selected to be a seed solution. The Diversification Generator of the next section can easily be restricted to generate such a solution that is “far from” the seed solution by reference to the observations previously indicated. The potential value of these optional steps for a given application can of course be conveniently determined by implementing a version of the SS/PR Template that excludes them, and then comparing the outcomes with those produced by incorporating Step 6A and/or Step 7A.

5 Maintaining And Updating The Reference Set

The Reference Set Update method accompanies each application of the Improvement Method, and we examine this updating function first because of its linking role, which also introduces structures that provide a foundation for the Subset Generation Method. The update operation consists of maintaining a record of the b best solutions found, where value of b is treated as a chosen constant, but may readily be allowed to vary. The underlying issues are conceptually straightforward, but their ramifications within the SS/PR Template are sufficiently extensive to motivate a concrete design for such a procedure.⁷

Hereafter, we will refer to the Reference Set by the abbreviation “RefSet”. We provide a pseudo-code description of the procedure for maintaining RefSet, which we

⁷ This section and the next are devoted explicitly to concerns of computer implementation. We include details that serve primarily for convenience as well as those that serve a more substantial function.

call the *RefSet Update Routine*, that is organized to handle vectors of 0-1 variables. Related forms to handle permutation vectors follow the same general format.

5.1 Notation and Initialization

Let $bNow$ denote the current number of solutions in the Reference Set. $bNow$ begins at 0 in an Initialization Step, and is increased each time a new solution is added to the Reference Set, until reaching the value $bMax$ (as where $bMax$ may take a value from 10 to 40, for example).

At each step, RefSet stores the best solutions in an array $x[i]$, $i = 1$ to $bNow$. The solution vector $x[i]$ may be viewed as the “ i th row” of the array. An associated location array $loc(i)$, $i = 1$ to $bNow$, indicates the ranking of the solutions; that is, $x[loc(1)]$ (the x vector stored in location “ $loc(1)$ ”) is the best solution, $x[loc(2)]$ is the next best solution, and so forth.

A solution $x = x'$ is not permitted to be recorded if it duplicates another already in RefSet.⁸ We speed the check to see if one solution duplicates another by keeping some simple auxiliary information about each solution. Specifically, in addition to keeping a value $E(loc(i))$ which identifies the evaluation (such as an objective function value) for $x[loc(i)]$, we may also keep a hash function value for this solution which we denote by $Hash(loc(i))$. Such a value can be determined by a single pass of the components of $x[loc(i)]$.

To test for adding x' to RefSet, let $E0$ and $Hash0$ be defined for x' in the same way the corresponding quantities $E(loc(i))$ and $Hash(loc(i))$ are defined for $x[loc(i)]$. In addition to storing x' properly, if it qualifies to be added to RefSet, the RefSet Update Subroutine will keep track of a value $NewRank$, which will be the rank of x' (as 1st best, 2nd best, etc.) if it is added to RefSet, and otherwise receives the value 0. (Thus, checking whether $NewRank$ is 0 or positive upon completion of the subroutine will automatically disclose whether x' succeeded or failed to qualify for inclusion.)

Finally, we let $RefSetCall$ count the number of times the RefSet Update subroutine is called (which thus tells how many solutions are generated and examined as potential additions to RefSet), and let $RefSetAdd$ count the number of times a solution is actually added. These values can be useful in the situation where it is desired to control the number of steps of executing the overall algorithm by limiting the maximum value of $RefSetCall$ or $RefSetAdd$, for example. The values may appropriately be segregated and re-initialized to give separate statistics for the Initial Phase and for the Scatter Search/Path Relinking Phase of the SS/PR Template. Auxiliary information that may be useful is recorded by three counters, $DupCheck$,

⁸ Unlike the development of scatter search and path relinking, where duplications have no relevance, the notion of avoiding duplicate population members was not embraced in GA methods for many years, and some GA researchers continue instead to argue in favor of duplications (see, e.g., Goldberg, 1989). However, the work of Whitley and Kauth (1988) and Whitley (1989) introduces a “steady state” design that avoids duplications, and the usefulness of this policy for such amended GAs has also been demonstrated by Davis (1991).

FullDupCheck and FullDupFound, which respectively count the number of partial duplication checks, full duplication checks, and the number of occurrences when duplications were found.

Pseudo-Code for the RefSet Update

Initialization Step

```

bNow = 0
RefSetCall = 0
RefSetAdd = 0
DupCheck = 0
FullDupCheck = 0
FullDupFound = 0

```

RefSet Update Subroutine (To Add x' to RefSet if it qualifies):

(This subroutine calls the Add Subroutine, specified below, which carries out the details of adding the solution x' to Refset.)

Begin Subroutine:

```

RefSetCall = RefSetCall + 1
NewRank = 0
If bNow = 0 then:
    NewRank = 1
    (this indicates that  $x'$  will be recorded as the solution with the first rank,
    i.e., the best, since no other solution is yet recorded.)
    Compute Hash0
    Call the Add Subroutine
    End the RefSet Update Subroutine
Elseif bNow > 0 then:
    (First check  $x'$  against the worst of the bNow best solutions, as follows:)
    If  $E0 \leq E(\text{loc}(\text{bNow}))$  and if  $\text{bNow} = \text{bMax}$  then:
        ( $x'$  is not better than the worst, and the list is full, so don't add  $x'$  to RefSet)
        End the RefSet Update Subroutine.
    Else
        Compute Hash0
    Endif
    If  $E0 > E(\text{loc}(1))$  then:
        ( $x'$  is the new best solution of all, so record the rank of  $x'$  as NewRank)
        NewRank = 1.
    Else
        (go through the solutions  $x[\text{loc}(1)]$  to  $x[\text{loc}(\text{bNow})]$  in reverse order, to
        test if  $x'$  duplicates a previous solution or is new)
        For i = bNow to 1 (decreasing index order)
            If  $E(\text{loc}(i)) = E0$  then:
                DupCheck = DupCheck + 1
                (this counts the duplication checks that are not trivially
                eliminated by the objective function value)

```

```

    If Hash(loc(i)) = Hash0 then:
        FullDupCheck = FullDupCheck + 1
        (check  $x'$  against  $x[loc(i)]$ )
        If  $x' = x[loc(i)]$  then:
            FullDupFound = FullDupFound + 1
            End the RefSet Update Subroutine
            ( $x'$  is not added to RefSet)
        Endif
    Endif
Elseif  $E(loc(i)) > E0$  then:
    (by the sequence of the loop, the current  $i$  is the largest index  $i$ 
    that satisfies  $E(loc(i)) > E0$  – i.e.,  $x[loc(i)]$  is the worst solution
    that is still better than  $x'$ , so  $x'$  will be ranked as the  $(i+1)$ -st
    best solution)
    NewRank =  $i+1$ 
    Call the Add Subroutine
    End the RefSet Update Subroutine
Endif
End i
(If the method reaches here, it has gone through the loop above without
finding a solution better than  $x'$  and without finding a duplicate for  $x'$ .
So  $x'$  qualifies as a new best solution, though its evaluation value must
implicitly tie for best.)
NewRank = 1
Endif
Call the Add Subroutine
Endif
End RefSet Update Subroutine

```

Now we indicate the subroutine called by the RefSet Update Subroutine. In addition to the arrays already noted, we include a LastTime(loc0) array (updated at the end of the following Add Subroutine) where loc0 ranges over the locations loc(i), $i = 1$ to bMax. The array LastTime(loc0), which records the last (“most recent”) time that the solution stored in location loc0 changes its identity, is important for linking with the routines of Section 4, which are designed to assure no duplicate subsets X of RefSet are ever generated).

Add Subroutine (to add x' to RefSet, given that it qualifies):

Begin Subroutine

RefSetAdd = RefSetAdd + 1

(x' will be recorded in the location occupied by the current worst solution. In case $bNow < bMax$, imagine the “worst” to be the empty solution in the location $loc(bNow + 1)$, which will become $loc(bNow)$ after incrementing $bNow$.)

If $bNow < bMax$ then:

$bNow = bNow + 1$

```

    loc(bNow) = bNow
Endif
(Next, the location pointers, loc(i), must be updated. First save the pointer to the solution that is currently worst, because this is the location where x' will be stored.)
    loc0 = loc(bNow)
(Now update the location pointers that change, as a result of making x' the solution that acquires the rank of NewRank. We only need to change pointers from NewRank + 1 to bNow, because loc(NewRank) will be updated for x', below. To avoid destroying proper values, the change must be made in reverse order.)
    If NewRank < bNow then:
        For i = bNow to NewRank + 1 (in decreasing index order)
            loc(i) = loc(i-1)
        End i
    Endif
    x[loc0] = x'
(thus x' is stored in the current location loc0)
    loc(NewRank) = loc0
(x' will now be accessed as the solution whose rank is NewRank, via the loc array)
    Hash(loc0) = Hash0.
(Finally, record the "time", given by NowTime, when the solution in location loc0 last changed its identity. NowTime is updated elsewhere, as shown later.)
    LastChange(loc0) = NowTime

```

End of Add Subroutine

A variation on ideas embodied in the preceding routines can be used to provide a method that checks for and eliminates duplications among solutions that are passed to the Improvement Method in Steps 2 and 7 of the SS/PR Template. By the philosophy of the approach developed here, such a weeding out of duplicates in the part of the overall approach can also be particularly useful, and a pseudo-code for the corresponding method is provided in Appendix 2.⁹

6 Choosing Subsets Of The Reference Solutions

We now introduce a special approach for creating different subsets X of RefSet, as a basis for implementing Step 5 of the SS/PR Template. It is important to note the

⁹ A simple refinement can also be introduced in the current routine by identifying the first and last indexes of nonzero components of x' during the pass of the components of x' that identifies Hash0. In case the hash value check does not disclose that x' differs from $x[\text{loc}(i)]$, the correspondence of these first and last indexes with those previously saved for $x[\text{loc}(i)]$ can also be checked. This allows a full check of x' to $x[\text{loc}(i)]$ to be restricted to components within the range of these indexes.

SS/PR Template prescribes that the set $C(X)$ of combined solutions (i.e., the set of all combined solutions that we intend to generate) is produced in its entirety at the point where X is created. Therefore, once a given subset X is created, there is no merit in creating it again. This creates a situation that differs noticeably from those considered in the context of genetic algorithms.

In some scatter search proposals, for example, the set $C(X)$ associated with X consists of a single (weighted) center of gravity of the elements of X . Once the center of gravity is created from X , it is preferable to avoid recreating the same subset X in the future. Other proposals similarly specify a particular set of combined points to be created from a given subset X . These points may be variable in number, as from a deterministic algorithm applied to X that terminates when the quality of points generated falls below a threshold. However, the total number of such points that are retained after an initial screening is usually small. Some path relinking proposals have a corresponding character (see, e.g., Glover, 1994b).

In such situations, we seek a procedure that generates subsets X of RefSet that have useful properties, while avoiding the duplication of subsets previously generated. Our approach for doing this is organized to generate four different collections of subsets of RefSet, which we refer to as SubSetType = 1, 2, 3 and 4. The principles we apply to generate these subsets can be applied to create additional subsets of a related character. We can also adapt the basic ideas to handle a policy that, by contrast, allows selected subsets of solutions to be generated a second time — as where it may be desired to create more than one “brood” of offspring from a given collection of parents, under conditions where the history of the method suggests that such a collection should be singled out for generating additional offspring.

A central consideration is that RefSet itself will not be static, but will be changing as new solutions are added to replace old ones (when these new solutions qualify to be among the current b best solutions found). The dynamic nature of RefSet requires a method that is more subtle than one that itemizes various subsets of an unchanging RefSet. In addition, we wish to restrict attention to a relatively small number of subsets with useful features, since there are massive numbers of subsets that may be generated in general. The types of subsets we consider are as follows.

- SubsetType = 1: all 2-element subsets.
- SubsetType = 2: 3-element subsets derived from the 2-element subsets by augmenting each 2-element subset to include the best solution not in this subset.
- SubsetType = 3: 4-element subsets derived from the 3-element subsets by augmenting each 3-element subset to include the best solutions not in this subset.
- SubsetType = 4: the subsets consisting of the best i elements, for $i = 5$ to $bNow$.

The total number of subsets that satisfy the preceding stipulations is usually quite manageable. For example, if $bMax = 10$ there are 45 different 2-elements subsets for SubsetType = 1, and the collections for SubsetType = 2 and 3 each contain a bit less

than 45 additional subsets. All together, SubsetType = 1 to 4 would generate approximately 130 distinct subsets. If bMax = 20, the total number of different subsets generated is a little less than 600. Depending on the number of solutions contained in $C(X)$, and on the amount of time required to generate a given combined solution and to enhance it by the Improvement Method, the value of bMax can be increased or decreased, or the types of subsets produced can similarly be changed (to produce variants or subsets of the four types generated by the process subsequently indicated).

Since the method will continue to add new solutions to RefSet — until no new solutions can be found that are better than the bMax best — the number of subsets generated will typically be larger than the preceding figures. Consequently, a limit is placed on the number of solutions generated in total, in case the best solutions keep changing. (A limit may also be placed on the number of iterations that elapse after one of the top 2 or 3 solutions has changed.) An appropriate cutoff can be selected by initial testing that gives the cutoff a large value, and by saving statistics to determine what smaller value is sufficient to generate good solutions.

Rationale

The reason for choosing the four indicated types of subsets of RefSet is as follows. First, 2-element subsets are the foundation of the first “provably optimal” procedures for generating constraint vector combinations in the surrogate constraint setting, whose ideas are the precursors of the ideas that became embodied in scatter search (see, e.g., Glover, 1965; Greenberg and Pierskalla, 1970). Also, conspicuously, 2-element combinations have for many years dominated the genetic algorithm literature (in “2-parent” combinations for crossover).

The generation of 2-element subsets also automatically generates (b-2)-element subsets, as complements of the 2-element subsets. We find the collection of (b-2)-element subsets less interesting, in the sense that the relative composition of such subsets tends to be much less varied than that of the 2-element subsets. (An exception occurs where the excluded elements may significantly affect the nature of the set $C(X)$.)

Our extension of the 2-element subsets to 3-element subsets is motivated by the fact that the values 2 and 3 are (proportionally) somewhat different from each other, and hence we anticipate the 3-element subsets will have an influence that likewise is somewhat different than that of the 2-element subsets. However, since the 3-element subsets are much more numerous than the 2-element subsets, we restrict consideration to those that always contains the best current solution in each such subset, which therefore creates approximately the same number of subsets as the 2-element subsets. A variation would be to replace this best solution with one of the top several best solutions, chosen pseudo randomly. Likewise, we extend the 3-element subsets to 4-element subsets for the same reason, and similarly restrict attention to a subcollection of these that always includes the two best solutions in each such subset.

As the subsets become larger, the proportional difference in their successive sizes becomes smaller. The computational effort of handling such subsets also tends to

grow. Hence we have chosen to limit the numbers of elements in these subsets (in this case to 4).

Nevertheless, to obtain a limited sampling of subsets that contain larger numbers of solutions we include the special subsets designated as `SubsetType = 4`, which include the b best solutions as b ranges from 5 to $bNow$. (Recall that $bNow$ increases as each new solution is added, until reaching $bMax$.) Since such subsets increasingly resemble each other for adjacent values of b as b grows, an option is to increment b by some fraction of $bMax$, e.g., $bMax/5$, instead of by 1.

6.1 Generating the Subsets of Reference Solutions

Each of the following algorithms embodies within it a step that consists of generating the set of combined solutions $C(X)$ and executing the Improvement Method (e.g., as proposed in Section 4). Regardless of the form of the Improvement Method used, we will understand that the Reference Set Update method of the previous section is automatically applied with it.

To introduce the general approach to create the four types of subsets, we first briefly sketch a set of four corresponding simple algorithms that could be used in the situation where `RefSet` is entirely static (i.e., where the set of $bMax$ best solutions never changes). These algorithms have the deficiency of potentially generating massive numbers of duplications if applied in the dynamic setting (where they must be re-initiated when `RefSet` becomes modified). However, their simple nature gives a basis for understanding the issues addressed by the more advanced approach.

In the following, when we specify that a particular solution is to become the second solution in X , we understand that the current first solution in X is unchanged, and similarly when we specify that a solution is to become the third (fourth) solution in X , we understand that the previous first and second (and third) solutions are unchanged.

Simple Algorithm for Subset Type 1

```

For i = 1 to bNow - 1
    Let x[loc(i)] be the first solution in X
    For j = i+1 to bNow
        Let x[loc(j)] be the second solution in X
        Create C(X) and execute the Improvement Method
    End j
End i

```

Simple Algorithm for Subset Type 2

```

Let x[loc(1)] be the first solution in X
For i = 2 to bNow - 1
    Let x[loc(i)] be the second solution in X
    For j = i+1 to bNow
        Let x[loc(j)] be the third solution in X
        Create C(X) and execute the Improvement Method
    End j
End i

```

Simple Algorithm For Subset Type 3

```

Let x[loc(1)] and x[loc(2)] be the first two solutions in X
For i = 3 to bNow - 1
    Let x[loc(i)] be the third solution in X
    For j = i+1 to bNow
        Let x[loc(j)] be the fourth solution in X
        Create C(X) and execute the Improvement Method
    End j
End i

```

Simple Algorithm for Subset Type 4

```

For i = 1 to bNow
    Let x[loc(i)] be the ith solution in X
    If i ≥ 5:
        Create C(X) and execute the Improvement Method
End i

```

6.2 Methods for a Dynamic RefSet

We must create somewhat more elaborate processes than the preceding to handle a dynamically changing reference set. We first indicate initializations that will be used to facilitate these processes.

Initialization Step:

```

For SubsetType = 1 to 4
    LastRunTime(SubsetType) = 4
End i
NowTime = 0
StopCondition = 0
SubsetType = 0

```

LastRunTime (SubsetType) is set to 4 above, since the approach cycles through 4 component methods. This initialization is a device to allow the approach to execute

all 4 methods on the first pass. Afterward, the procedure will terminate based on updated values of LastRunTime compared to LastChange, as subsequently explained.

In the iterative application of the steps of the SS/PR Template, we start from SubsetType = 0, as in the final statement of the Initialization Step, and repeatedly increase this index in a circular pattern that returns to SubsetType = 1 after reaching SubsetType = 4. However, the design that follows can be readily adapted to cycle through the subsets in any other order.

6.3 Arrays for the Subset Generation Method

There are two key arrays for the subset generation method, LastChange(loc(i)) and LastRunTime(SubsetType), that form the basis for avoiding duplications efficiently. We describe the function of these arrays, and the associated components that govern their updates, as follows.

(1) LastChange(loc(i)) – identifies the last (most recent) time that the solution stored in location loc(i) changed its identity (i.e., the last time a new solution was written over the old one in this location). More precisely, LastChange(loc(i)) is assigned the value NowTime when this change occurs. NowTime is increased by 1 each time one of the four algorithms prepares to generate the subsets of the type it deals with. As a result, NowTime is always 1 more than the value that could have been assigned to LastChange(loc(i)) on the previous execution of any (other) algorithm. Thus, the condition LastChange(loc(i)) = NowTime can only hold if a solution was changed in location loc(i) during the execution of the current algorithm for selecting elements to combine.

(2) LastRunTime(SubsetType) – identifies the last time (the value of NowTime on the last time) that the Algorithm SubsetType (= 1, 2, 3, or 4) was executed. Thus, for iLoc = loc(i), the condition LastChange(iLoc) < LastRunTime(SubsetType) means that the last time the solution x[iLoc] changed, occurred before the last time the Algorithm SubsetType was run. Hence x[iLoc] will now be the same as when the Algorithm looked at it previously. On the other hand, if LastChange(iLoc) ≥ LastRunTime(SubsetType), then the solution was changed either by Algorithm SubsetType itself, or by another algorithm executed more recently, and so x[iLoc] is not the same as when Algorithm SubsetType looked at it before.

6.4 Subset Generation Method

A basic part of the Subset Generation Method is the Subset Control Subroutine, which oversees the method and calls other subroutines to execute each Algorithm SubsetType (for SubsetType = 1 to 4). We indicate the form of this subroutine first. (The parameter StopCondition that governs the outer loop, which immediately follows, is initialized to 0 in the Initialization Step. When the cumulative number of executions of the Improvement Method, as it is applied within the various Algorithm Subroutines, exceeds a chosen limit, then StopCondition is set to 1 and the overall method thereby stops.)

Subset Control Subroutine

While StopCondition = 0 do

 SubsetType = SubsetType + 1

 If SubsetType > 4 then SubsetType = 1

 NowTime = NowTime + 1

 iNew = 0

 iOld = 0

(The next loop isolates all new (changed) solutions by storing their locations in LocNew(i), i = 1 to iNew, and all old (unchanged) solutions by storing their locations in LocOld(i), i = 1 to iOld. If iNew winds up 0, nothing has changed. When all algorithms are performed one after another, as here, regardless of sequence, the condition iNew = 0 means nothing has changed for any of them, and the method can stop.)

 For i = 1 to bNow

 iLoc = loc(i)

 if LastChange(iLoc) ≥ LastRunTime(SubsetType) then

 iNew = iNew + 1

 LocNew(iNew) = iLoc

 else

 iOld = iOld + 1

 LocOld(iOld) = iLoc

 Endif

 End i

 If iNew = 0 then end the Subset Control Subroutine

(iNew = 0 here implies all combinations of the four types of subsets have been examined for their current composition without generating any new solutions, and so the SS/PR Template can terminate as a result of exhaustively considering all relevant subsets of RefSet in its final composition.)

 If SubsetType = 1 Call Algorithm 1 Subroutine

 If SubsetType = 2 Call Algorithm 2 Subroutine

 If SubsetType = 3 Call Algorithm 3 Subroutine

 If SubsetType = 4 Call Algorithm 4 Subroutine

 (if StopCondition > 0 stop)

(Having identified the sets of old and new solutions and generated new combinations from them, we can update LastRunTime(SubsetType) to be the current NowTime value, so that the next time the algorithm is applied, LastRunTime(SubsetType) will in fact be the last (most recent) time the algorithm was run.)

 LastRunTime(SubsetType) = NowTime

End do

End Subset Control Subroutine

Next we identify the Algorithm Subroutines. Each Algorithm Subroutine works on the following principle. A subset X of RefSet can be new if and only if at least one

element of X has not been contained in any previous X for the same SubsetType . We exploit this by sorting the solutions into old and new components, and executing a loop that first generates all combinations of new with new, and then a loop that generates all combinations of new with old. Meanwhile, any solution that is changed on the present application of the algorithm is excluded from being accessed once it has changed, because all subsets that include this solution will be generated on a later pass. To access a solution after it changes its rank, but before the loop is completed, would create duplications (unless the solution changes again), and in any case may generate more solutions than necessary. The method generates the least number of solutions “currently known” to be necessary.

Algorithm 1 Subroutine

Begin Subroutine

(Currently $i_{\text{New}} > 0$. If $i_{\text{New}} > 1$, then we look at all combinations of new with new)

```

    If  $i_{\text{New}} > 1$  then
        For  $i = 1$  to  $i_{\text{New}} - 1$ 
             $i_{\text{Loc}} = \text{LocNew}(i)$ 
            If  $\text{LastChange}(i_{\text{Loc}}) < \text{NowTime}$  then
                (the solution in  $i_{\text{Loc}}$  is still unchanged, so we can use it, otherwise the method  $i$  skips it)
                Let  $x[i_{\text{Loc}}]$  be the first element of  $X$ 
                For  $j = i + 1$  to  $i_{\text{New}}$ 
                     $j_{\text{Loc}} = \text{LocNew}(j)$ 
                    If  $\text{LastChange}(j_{\text{Loc}}) < \text{NowTime}$  then
                        Let  $x[j_{\text{Loc}}]$  be the second element of  $X$ 
                        Create the set  $C(X)$  and execute the Improvement Method
                        (Optional check: if  $\text{LastChange}(i_{\text{Loc}}) = \text{NowTime}$  then jump to the end of the "I loop" to pick up the next  $i$ , and generate fewer solutions.)
                    Endif
                End j
            Endif
        End i
    Endif
    If  $j_{\text{Old}} > 0$  then
        For  $i = 1$  to  $i_{\text{New}}$ 
             $i_{\text{Loc}} = \text{LocNew}(i)$ 
            If  $\text{LastChange}(i_{\text{Loc}}) < \text{NowTime}$  then
                Let  $x[i_{\text{Loc}}]$  be the first element of  $X$ 
                For  $j = 1$  to  $j_{\text{Old}}$ 
                     $j_{\text{Loc}} = \text{LocOld}(j)$ 
                    If  $\text{LastChange}(j_{\text{Loc}}) < \text{NowTime}$  then
                        Let  $x[j_{\text{Loc}}]$  be the second element of  $X$ 
                        Create the set  $C(X)$  and execute the Improvement Method
                    Endif
                End j
            Endif
        End i
    Endif

```

(Optional check: if LastChange(iLoc) = NowTime then jump to the end of the “i loop” to pick up the next i, and generate fewer solutions.)

Endif

End j

Endif

End i

Endif

End Subroutine

Algorithm 2 Subroutine

Begin Subroutine

loc1 = loc(1)

Let x[loc1] be the first element of X

If LastChange(loc1) $\geq$ LastRunTime(SubsetType) then

(The solution in location loc1 is new, since last time.)

For i = 2 to bNow - 1

iLoc = loc(i)

If LastChange(iLoc) < NowTime then

(The solution in iLoc is still unchanged, so we can use it, otherwise the method skips it)

x[iLoc] is the second element of X

For j = i + 1 to bNow

jLoc = loc(j)

If LastChange(jLoc) < NowTime then

x[jLoc] is the third element of X

Create C(X) and execute the Improvement Method

(Optional check: if LastChange(iLoc) = NowTime then jump to the end of the “i loop” to pick up the next i, and generate fewer solutions.)

Endif

End j

Endif

End i

End Algorithm 2 Subroutine (if reach here)

Else

(The solution in location loc1 is not new, since last time.)

(If iNew > 1, then we look at all combinations of new with new.)

If iNew > 1 then

For i = 1 to iNew - 1

iLoc = LocNew(i)

If LastChange(iLoc) < NowTime Then

x[iLoc] is the second element of X

For j = i + 1 to iNew

jLoc = LocNew(j)

If LastChange(jLoc) < NowTime then

```

                                x[jLoc] is the third element of X
                                Create C(X) and execute the Improvement Method
                                (Optional check: if LastChange(iLoc) = NowTime
                                then jump to the end of the "i loop" to pick up the next
                                i, and generate fewer solutions.)
                                Endif
                            End j
                        Endif
                    End i
                Endif
            If jOld > 1 then
                For i = 1 to iNew
                    iLoc = LocNew(i)
                    If LastChange(iLoc) < NowTime then
                        Let x[iLoc] be the second element of X
                        For j = 2 to jOld
                            (loc1 is actually also LocOld(1))
                            jLoc = LocOld(j)
                            If LastChange(jLoc) < NowTime then
                                Let x[jLoc] be the third element of X
                                Create C(X) and execute the Improvement Method
                                (Optional check: if LastChange(iLoc) = NowTime
                                then jump to the end of the "i loop" to pick up the next
                                i, and generate fewer solutions.)
                                Endif
                            End j
                        Endif
                    End i
                Endif
            Endif
        End Subroutine

```

The optional checks in Algorithms 1 and 2 are based on the fact that the condition $\text{LastChange}(iLoc) = \text{NowTime}$ implies that $x[iLoc]$ has been changed by finding a new solution with the Improvement Method. No duplications are created if the algorithm continues its course, using the old version of $x[iLoc]$. But it is also legitimate to jump to the end of the "i loop", as indicated, to generate fewer solutions. Algorithm 2 can also include a more influential check (in the same locations) which asks if $\text{LastChange}(\text{loc1}) = \text{NowTime}$, and terminates the current execution of the algorithm if so. In this case, a variation on the general organization could allow Algorithm 2 to be re-initiated immediately, since all its subsets will not incorporate a new "best overall" solution. Similar comments apply to introducing optional checks within the Algorithm 3 Subroutine, where $\text{LastChange}(\text{loc2})$ can also be checked. We do not bother to include further mention of such options.

Algorithm 3 Subroutine**Begin Subroutine**

```

loc1 = loc(1)
loc2 = loc(2)
Let x[loc1] and x[loc2] be the first two elements of X
If LastChange(loc1) ≥ LastRunTime(SubsetType) or LastChange(loc2) ≥
LastRunTime(SubsetType) then
    (The solution in location loc 1 or in loc2 is new, since last time.)
    For i = 3 to bNow - 1
        iLoc = loc(i)
        If LastChange(iLoc) < NowTime Then
            (The solution in iLoc is still unchanged, so we can use it, otherwise
            the method skips it.)
            Let x[iLoc] be the third solution in X
            For j = i + 1 to bNow
                jLoc = loc(j)
                If LastTime(jLoc) < NowTime then
                    Let x[jLoc] be the fourth solution in X
                    Create C(X) and execute the Improvement Method
                Endif
            End j
        Endif
    End i
End Algorithm 3 Subroutine (if reach here)
Else
    (Solutions in locations loc1 and loc2 are not new, since last time)
    (If iNew > 1, then we look at all combinations of new with new.)
    If iNew > 1 then
        For i = 1 to iNew - 1
            iLoc = LocNew(i)
            If LastChange(iLoc) < NowTime Then
                Let x[iLoc] be the third solution in X
                For j = i + 1 to iNew
                    jLoc = LocNew(j)
                    If LastTime(jLoc) < NowTime then
                        Let x[jLoc] be the fourth solution in X
                        Create C(X) and execute the Improvement Method
                    Endif
                End j
            Endif
        End i
    Endif
    If jOld > 2 then
        For i = 1 to iNew
            iLoc = LocNew(i)
            If LastChange(iLoc) < NowTime then

```



```

        Let x[iLoc] be the third solution in X
        For j = 3 to jOld
            jLoc = LocOld(j)
            If LastChange(jLoc) < NowTime then
                Let x[jLoc] be the fourth solution in X
                Create C(X) and Execute Improvement Method
            Endif
        End j
    Endif
End i
Endif
End Subroutine

```

Algorithm 4 Subroutine

Begin Subroutine

```

    new = 0
    For i = 1 to 4
        iLoc = loc(i)
        Let x[iLoc] be the ith solution in X
        If LastChange(iLoc) ≥ LastRunTime(SubsetType) then new = 1
    End i
    For i = 5 to bNow
        If LastChange(iLoc) ≥ LastRunTime(SubsetType) then new = 1
        If LastChange(iLoc) < NowTime then
            Let x[iLoc] be the ith solution in X
            If new = 1 then
                Create C(X) and execute the Improvement Method
            Endif
        Endif
    End i

```

End Subroutine

The preceding subroutines complete the collection for generating subsets of the Reference Set, without duplication. The comments within the subroutines should be sufficient to make their rationale visible, and to provide a basis for variations of the forms previously discussed.

7 Improvement Method

There often exist alternative neighborhoods of moves available to construct improvement methods for various kinds of optimization problems. Experience from numerous applications suggests that there is merit in using more than one such neighborhood. For example, a common theme of strategic oscillation, as applied in

tabu search, is to cycle among alternative neighborhoods according to various patterns. Strategic oscillation also commonly operates by cycling through various regions, or “levels” of a given neighborhood.

The approach of cycling through different levels of a neighborhood is manifest in two types of candidate list strategies, the Filtration Strategy and the Sequential Fan Strategy, proposed with tabu search (see, e.g., Glover and Laguna, 1997). The goal of these strategies is to identify attractive moves with an economical degree of effort. In addition, however, the Filtration and Sequential Fan strategies offer a useful basis for converting a simple Improvement Method into a more advanced one. We propose a way to marry these two candidate list strategies to create a *Filter and Fan Method* which provides a convenient form of an Improvement Method for the SS/PR Template.

We emphasize that other options exist for creating improvement methods that can be incorporated in the SS/PR Template. In settings where an effective improvement method already exists and has demonstrated its utility, such a procedure can be relied on directly and the primary concern will be to exploit it by the routines described in the preceding sections. Alternatively, components of such a procedure – as, for example, the moves it relies on and the evaluations it uses to choose among these moves – can be embedded within the general approach we introduce here.

Component Moves

The moves to serve as building blocks for the proposed method will characteristically be simple types of moves as illustrated by adjacent integer changes for integer variables (e.g., “flip” moves for 0-1 variables) and by elementary insert or swap moves for permutation problems. We call the chosen component moves level 1 moves, or *1-moves*.

An associated Level 1 Improvement Method can be defined relative to the 1-moves, which operates by segregating a collection of 1-moves by a preliminary candidate list approach, such as an Aspiration Plus strategy (Glover and Laguna, 1997). A random selection of such a collection is possible, in the interest of simplification, but at an appreciable risk of reducing overall effectiveness. (When randomization is used, the initial list should typically be larger than otherwise required.)

A useful goal for the initial candidate list strategy is to assure that a number of these 1-moves are among the highest evaluation moves currently available, so that if none of them is improving, the method is likely to be at a local optimum relative to these moves. The Level 1 Method then terminates when no moves from its candidate list are improving moves, thus presumably stopping at a local optimum or a “near” local optimum (relative to a larger collection of moves that encompasses those of the candidate list). The candidate list construction for Level 1 can be dynamic to allow the size of the list to grow when no improving move is found. (The Aspiration Plus strategy has this character, for example.) Such an optimum makes it possible to assure that termination will occur at a local optimum, if desired. The Filter and Fan Method then goes beyond this stopping point to create higher level moves. For this purpose,

we extract a subset M of some number of best moves from those examined by the Level 1 method when it terminates, where for example $|M| = 20$ or 40 .

General Design.

The general design of the Filter and Fan Method is to isolate a subset $M(L)$ of the best moves at a given level L , to be used as a basis for generating more advanced moves at level $L+1$ when level L fails to yield an improving move. In case $L=1$, we choose $M(1)$ to be a subset of M .

Suppose that m is a given L -move from $M(L)$, and let $A(m)$, be a related set of 1-moves (derived from M) so that the result of applying any 1-move m' in $A(m)$, after applying m will create an $(L+1)$ -move which we denote by $m@m'$. By restricting $M(L)$ to consist of a relatively small number of the moves examined at Level L (e.g., choosing $|M(L)| = 10$ or 20), and likewise restricting $A(m)$ to consist of a relatively small number of 1-moves, the total number of $L+1$ moves $m@m'$ can be maintained at a modest size. For example, a steady state choice that always picks $|M(L)| = 16$ and $|A(m)| = 8$ (for each m in $M(L)$) will generate only 128 $(L+1)$ -moves to be examined at each level $L+1$. If none are improving, in this example the 16 best are selected to compose $M(L+1)$, and the process repeats.

The utility of this design is to avoid the combinatorial explosion of possibilities that results by generating the set of all possible $(L+1)$ -moves at each step. Instead the approach filters a subset $M(L)$ of best moves at level L , and for each of these moves likewise filters a set $A(m)$ of best 1-moves from M . The “fan” that generates $|M(L)||A(m)|$ potential $(L+1)$ -moves as candidates to examine at Level $L+1$ is therefore maintained to be of reasonable size.¹⁰

We say that $A(m)$ is *derived from* M , not only because $A(m)$ may be smaller than M , but also because some of the moves of M may not be legitimate once the L -move m in $M(L)$ is executed. The 1-moves available after applying move m may not precisely correspond to moves of the original set M . For example, if the 1-moves correspond to flipping the values of 0-1 variables, then a move m may have flipped values for several variables in M , and the corresponding 1-moves in M will no longer be accessible. However, it is generally easy to keep a record for each move m in $M(L)$ that identifies the moves of M that should be excluded from $A(m)$, allowing $A(m)$ to be composed of the best $|A(m)|$ remaining members of M . Similar comments apply to moves such as swap moves and insert moves.

A simple steady state version of the Filter and Fan method can be summarized as follows. Let n_0 be the chosen size of the initial M , n_1 be the size of each set $M(L)$ and n_2 be the size of each set $A(m)$ (where n_1 and n_2 do not exceed n_0). In many applications, n_0 will be at most 40 and n_1 and n_2 will be at most 20 (and smaller

¹⁰ The emphasis on controlling computational effort while producing good candidate moves can be facilitated in some settings by an accelerated (shortcut) evaluation process. This can occur by selecting members of an initial candidate list that provides the source of M , as illustrated by the use of surrogate constraint evaluations in place of a lengthier evaluation that identifies the full consequences of a move relative to all constraints of a problem. Accelerated evaluations can also be applied to isolating M from the initial candidate list, while reserving more extensive types of evaluations to isolating $M(1)$ from M , and to deriving $A(m)$ from M for the moves m generated at various levels.

values may be preferable). We call this version a *strict* improvement method because it does not allow steps that are nonimproving, and therefore terminates at a local optimum relative to the multilevel moves it employs.

Filter and Fan Strict Improvement Method

1. *Generate a candidate list of 1-moves for the current solution x .*
 - (a) If any of the 1-moves are improving: Choose the best member from the list and execute it to create a new current solution x . (The “best member” may be the only member if the list terminates with the first improving move encountered.) Then return to the start of step 1.
 - (b) If none of the 1-moves are improving: Identify the set M of the n_0 best 1-moves examined. Let $M(1)$ be a subset of the n_1 best moves from M , and let $X(1)$ be the set of solutions produced by these moves. Set $L = 1$ and proceed to step 2.
2. *For each L -move m in $M(L)$:* Identify the associated set $A(m)$ of the n_2 best compatible moves m' derived from M , and evaluate each resulting $(L+1)$ -move $m@m'$. (Equivalently, evaluate each solution that results by applying move m' to the corresponding member of $X(L)$.) When fewer than n_2 moves of M are compatible with move m , restrict consideration to this smaller set of moves in composing $A(m)$.
 - (a) If an improving move is found during the foregoing process: Select the best such move generated (by the point where the process is elected to terminate), and execute the move to create a new current solution x . Then return to step 1.
 - (b) If no improving move is found by the time all moves in $M(L)$ are examined: Stop if L has reached a chosen upper limit $\text{Max}L$. Otherwise, identify the set $M(L+1)$ of the n_1 best $(L+1)$ -moves evaluated (and/or identify the associated set $X(L+1)$). (If fewer than n_1 distinct $(L+1)$ -moves are available to be evaluated, then include all distinct $(L+1)$ -moves in $M(L+1)$.) Then set $L = L + 1$ and return to the start of Step 2.

The identification of $M(L+1)$ (and/or $X(L+1)$) in Step 2(b) can of course be undertaken as part of the process of looking for an improving move, rather than waiting until no improving move is found. The appropriate organization depends on the setting. Also, an evident option for executing the preceding method is to allow a variable-state version where n_1 and/or n_2 decreases as L increases, thereby reducing the number of candidates for successively higher levels of moves. Another option is to allow L to change its value by a different pattern. We now examine relevant considerations for implementing this method.

7.1 Avoiding Duplications

A slight change in the preceding description can improve the approach by avoiding the generation of a number of duplicate outcomes. Such duplications are especially

likely to arise when generating 2-moves, in the setting where solutions are generated by flipping 0-1 variables. To illustrate, suppose $n_0 = n_1 = n_2 = 20$. Thus, initially M consists of the 20 best 0-1 flips, and we consider the case where none are improving. Then at Step 2, having chosen $M(1) = M$ (since $n_1 = n_0$), the method as described would select each move m from $M(1)$ and extend it by applying a compatible 1-move m' taken from $A(m)$, where in this case $A(m)$ consists of all of M excluding the single flip that produced m . Thus, each move m in $M(1)$ would be matched with the 19 other 1-moves that constitute flips other than the one embodied in m . Over the 20 elements of $M(1)$ this yields $20 \times 19 = 380$ possibilities to evaluate. But this is double the number that is relevant, since each flip of two variables x_i and x_j will be generated twice — once when the x_i flip is in $M(1)$ and the x_j flip is in $A(m)$, and once when the x_j flip is in $M(1)$ and the x_i flip is in $A(m)$. Such duplication can easily be removed by restricting a flip of two variables x_i and x_j so that $j > i$, where x_i belongs to $M(1)$ and x_j belongs to $A(m)$. This indexing restriction may alternately be applied after the flips are sorted in order of their attractiveness. In either case, the result may be viewed as restricting the definition of $A(m)$.

Potential duplications for other types of moves can similarly be easily avoided at the level $L = 1$, where 2-moves are being generated in Step 2. In the case of swap and insert moves, a more balanced set of options can be created by restricting the number of moves recorded in M that involve any given element (or position). For example, if 5 of the 20 best swap moves involve swapping a given element i , then it may be preferable to record only the 2 or 3 best of these in M , and therefore complete the remainder of M with moves that may not strictly be among the 20 best.

As L grows larger, the chance for duplications drops significantly, provided they have been eliminated at the first execution of Step 2. Consequently, special restrictions for larger values of L can be disregarded. Instead, it is easier to screen for duplications at the point where a move becomes a candidate to include in $M(L+1)$ (or equivalently, the associated solution becomes a candidate to include in $X(L+1)$). The method for updating the Reference Set, given in the Section 3, can be used as a design to conveniently identify and eliminate such duplications in the present context as well.¹¹

7.2 Move Descriptions

The influence of move descriptions, where the same move can be characterized in alternative ways, can affect the nature of moves available independently of the neighborhood used. This phenomenon is worth noting, because standard analyses tend to conceive the neighborhood structure as the sole determinant of relevant

¹¹ This can be made simpler by recording “incremental solutions” — or more precisely the representations of solutions in $X(L)$ that result when the current solution x is represented as the zero vector — since these will generally have few nonzero components, and may be stored and accessed more quickly than complete solution vectors.

outcomes. The phenomenon arises in the Filter and Fan method because the move description implicitly transmits restrictions on deeper level moves in a manner similar to the imposition of *tabu* restrictions in tabu search. Thus, an implicit memory operates by means of the attributes of the moves considered, and these attributes depend on the move description.

To illustrate, a description that characterizes an insert move to consist of inserting element i in position p (and shifting other elements appropriately) can yield a different outcome, once intervening moves are made, than a description that characterizes the same move as inserting element i immediately before an element v . (Note that a more restrictive description, such as specifying that the move consists of inserting element i between elements u and v , may render the move impossible once either u or v changes its position.)

The phenomenon can be usefully demonstrated for swap moves by the situation where two moves in M are respectively characterized as swapping elements i and j and swapping elements i and k . After performing the first swap, the second will receive a changed evaluation, since i is no longer in the same position. If instead the same moves are characterized as swapping the elements in positions p and q , and in positions p and r (where elements i , j and k are currently in these positions), then the result of the first swap gives a different outcome for the second swap than the one illustrated previously; that is, the second swap now corresponds to swapping elements j and k rather than i and k . (Still another outcome results if a swap is characterized as a double insert move, e.g., as inserting an element i immediately after the current predecessor of j and inserting j immediately after the current predecessor of i .)

A preferable description of course depends in part on the nature of the problem. An interesting possibility is to allow two (or more) move characterizations, and then to choose the one in a given situation that yields the best result. This is analogous to allowing different solution attributes to define potential restrictions by the attribute-based memory of tabu search.

By the same token, it may be seen that greater flexibility can be obtained simply by relaxing the definition of a move. For example, in the setting of 0-1 problems, instead of characterizing M as a set of value-specific flips (as illustrated by stipulating that x_j should change from 0 to 1), we can allow M to be value-independent (as illustrated by stipulating that x_j should change to $1 - x_j$). The value-independent characterization allows greater latitude for generating moves from M , and is relevant as L grows beyond the value of 1. Such a characterization should be accompanied by a more explicit (and structured) use of tabu search memory, however, to control the possibility of cycling. The value-specific characterization is sufficiently limiting to avoid this need in the present illustration.

Another degree of latitude exists in deriving $A(m)$ from M . Suppose the moves of M are denoted $m(1)$, $m(2)$, ..., $m(u)$, where the moves with smaller indexes have higher evaluations. If we stipulate that $A(m)$ should consist of the r best of these moves, restricted to those that are compatible with m , we may choose to avoid some computation by ordering M in advance, as indicated, and then simply selecting the first r compatible members to compose $A(m)$. However, since the relative

attractiveness of the moves in M may change once the move m is made, an alternative strategy is instead to examine some larger number of compatible members m' of M to improve the likelihood of including the “true r best” options for the compound moves $m@m'$. This of course does not change the form of the method, since it amounts to another possibility for choosing the size of $|A(m)|$. However, this observation discloses that it may be preferable to choose the size of $|A(m)|$ to be larger relative to the size of $|M(L)|$ than intuition may at first suggest.

7.3 Definitions of 1-moves and the Composition of M

It is entirely possible in the application of the Filter and Fan Method that a 1-move may be defined in such a way to produce an infeasible solution, but a coordinated succession of 1-moves will restore feasibility. A simple example is provided by the graph partitioning problem, where feasible solutions consist of partitioning a set of $2n$ nodes into two sets that contain n nodes each. A 1-move that consists of swapping two nodes that lie in different sets will maintain feasibility, but a 1-move that consists of moving a node from one set to the other will not. Nevertheless, since the second (simpler) 1-moves are many fewer in number, a candidate list strategy that identifies attractive moves of this type, differentiated according to the set in which they originate, also provides a useful basis for composing compound moves by the Filter and Fan Method. In this situation, the successive 1-moves must alternately be chosen from different sets, and feasible solutions only occur for even values of L . Such implementations are easily accommodated within the general framework, and we do not bother to introduce more complicated notation to represent them.

Similarly, a method that applies 0-1 flips to an integer programming problem may allow M to include flips that create infeasibility (as in a multidimensional knapsack problem where any flip from 0 to 1 will drive a “boundary solution” infeasible). Rather than avoiding moves that produce infeasibility, the process may make provision for an infeasibility generated at a level L to be countered by focusing on moves to recover feasibility at level $L + 1$. (The focus may be extended to additional levels as necessary.)

A related but more advanced situation arises in ejection chain strategies where 1-moves may be defined so that no sequence of them produces a feasible solution. In this type of construction a reference structure guides the moves selected to assure that a feasible solution can always be generated by one or more associated trial solutions (see, e.g., Glover, 1992; Rego 1996; Rego and Roucairol, 1996). In these instances the Filter and Fan Method can accordingly be modified to rely on the trial solutions as a basis for evaluating the L -moves generated.

The composition of M can be affected by an interdependency among subcollections of moves. In certain pivot strategies for network flow optimization, for example, once the best pivot move associated with a given node of the network is executed, then the quality of all other moves associated with the node deteriorates. Thus, instead of choosing M to consist of the n_0 best moves overall, which could possibly include several moves associated with the same node, it can be preferable to allow M to

include only one move for any given node. More generally, M may be restricted by limiting the numbers of moves of different categories that compose it.

The potential to modify M , $M(L)$ and $A(m)$ as the method progresses can be expanded by allowing these sets to be “refreshed” by a more extensive examination of alternatives than those earmarked for consideration when $L = 1$. For example, in some settings the execution of a move m may lead to identifying a somewhat restricted subset of further moves that comprise the only alternatives from which a compound improving move can be generated at the next level. In such a case, M and $A(m)$ should draw from such alternatives even though they may not be encompassed by the original M . Such an expansion of M and its derivative sets must be carefully controlled to avoid losing the benefit of reducing overall computation that is provided by a more restricted determination of these sets.

Allowing for such an expansion can increase the appropriate value of $MaxL$. Or inversely, the reliance on smaller sizes of M , $M(L)$ and $A(m)$ can decrease the appropriate value of $MaxL$. A reasonable alternative is to determine $MaxL$ indirectly by a decision to terminate when the quality of the best current L -moves (or of the 1-moves that are used to generate these L -moves) drops below a chosen threshold. These considerations are relevant to the “partial refreshing” process of the Multi-Stream variant, described in the next section.

7.4 Advanced Improvement Alternatives

The preceding Strict Improvement version of the Filter and Fan method has a conspicuous limitation that can impair the quality of the solutions it produces. In the absence of applying a refreshing option, which can entail a considerable increase in computation or complexity, the merit of the set M as the source of the sets $M(L)$ and $A(m)$ diminishes as L grows. This is due to the fact that M is composed of moves that received a high evaluation relative to the current solution before the compounding effects of successive 1-moves is considered, and as these moves are drawn from M for progressively larger values of L , the relevance of M as the source of such moves deteriorates. Under such circumstances the value of $MaxL$ will normally not be usefully chosen to be very large (and a value as small as 4 or 5 may not be atypical).

This limitation of the strict improvement method can be countered by one of three relatively simple schemes:

- (1) A Shift-and-Update variant.
- (2) A Diversification variant.
- (3) A Multi-Stream variant.

We examine these three variants as follows.

Shift-and-Update Variant

The Shift-and-Update method operates by choosing a nonimproving move when the method would otherwise terminate, but shifting the outcome away from the lowest levels to insure that the choice will yield a new solution that lies some minimum number of 1-moves “away from” the current solution x . The new solution is then

updated to become the current solution, and the method repeats. The method can be described by the following rule.

Shift-and-Update Rule

Identify a value MinL such that $1 < \text{MinL} \leq \text{MaxL}$. When no improvement is found upon reaching $L = \text{MaxL}$, select the best solution found over the levels $L \geq \text{MinL}$. Then specify this solution to be the current solution x and return to the beginning of Step 1.

Simple recency-based tabu search memory can be used with this rule to keep from returning to a preceding solution and generally to induce successively generated solutions to continue to move away from regions previously visited. This variant creates a degree of diversification which may be increased by restricting attention to value-specific move descriptions, so that each set $M(L)$ will be progressively farther removed from $M(1)$.

Diversification Variant

A more ambitious form of diversification, which can be applied independently or introduced as a periodic alternative to the Shift-and-Update approach, operates by temporarily replacing the steps of the Filter and Fan Method with a series of steps specifically designed to move away from regions previously visited. We express this approach by the following rule.

Diversification Rule

Introduce an explicit Diversification Stage when the approach fails to find an improvement (and terminates with $L = \text{MaxL}$). In this stage change the evaluator to favor new solutions whose attributes differ from those of solutions previously encountered, and execute a chosen number MinL of successive 1-moves guided by the changed evaluator. Immediately following this stage, the next pass of the Filter and Fan Method (starting again from Step 1 with a new x) uses the normal evaluator, without a diversification influence.

This diversification approach can use tabu search frequency memory to encourage moves that introduce attributes that were rarely or never found in solutions previously generated, and similarly to encourage moves that eliminate attributes from the current solution that were often found in previous solutions. An alternative is to apply the Diversification Stage directly within the Filter and Fan structure, where M and its derivative sets are generated by keeping the value of n_0 , and especially the values n_1 and n_2 , small. The evaluator in this case may alternatively be a composite of the normal evaluator and a diversification evaluator, with the goal of producing solutions that are reasonably good as well as somewhat different from previous solutions. Then the values of n_0 , n_1 and n_2 may be closer to those during a non-diversification stage. For a simpler implementation, instead of using frequency memory in the

Diversification Stage, the approach can be applied as noted in Section 1 by introducing a Post-Improvement step that uses the Diversification Generator.

Multi-Stream Variant

The Sequential Fan candidate list strategy embodies an additional element, not yet considered, that consists of generating several solution streams simultaneously. The inclusion of this element in the Filter and Fan Method produces a method that increases the likelihood of finding improved solutions. The ideas underlying this variant may be sketched in overview as follows.

Starting from the current solution x in Step 1 of the Filter and Fan Method, the approach selects a small number s of the best moves from the set M . Each selected move is allowed to initiate a different stream by creating an associated set of additional moves to launch the rest of the Filter and Fan Method. The outcome effectively shifts the Filter and Fan Method by one step, so that the solutions produced by these s moves take the place of solution x to produce additional levels of moves.

Specifically, let $x[i]$, $i \in S = \{1, \dots, s\}$ denote the solutions produced by the s best moves derived from x . The evaluations for each $x[i]$ are updated to generate a refreshed candidate list for each. Let $M(0:i)$ denote the set of n_0 best 1-moves from the candidate list created for $x[i]$ (in the same way that M represents the set of n_0 best 1-moves from the candidate list created for x). Similarly, let $M(L:i)$ and $X(L:i)$, $i \in S$, denote the sets of moves and corresponding sets of solutions for an arbitrary level $L \geq 1$.

The Multi-Stream variant then operates exactly as the single stream variant, by allowing each $x[i]$ for $i \in S$ to take the role of x , except that the computation is restricted in a special way. The purpose of this restriction is to allow the Multi-Stream variant to require only slightly more effort than the single stream variant, except for the initial step that identifies a fresh candidate list for each $x[i]$. (To achieve greater speed, an option is to forego this identification and instead rely on choosing each $M(0:i)$ as a subset of an enlarged set M generated for x . The relevance of this alternative is determined by considerations discussed in Section 3.)

The restricted treatment of the sets $M(L:i)$ at each stage of the Multi-Stream variant produces the following modifications of Steps 1 and 2 of the Filter and Fan Method.

Multi-Stream Implementation

1A. (Modification of step 1.)

- (a) As in the original step 1(a), return to the start of step 1 with a new solution x if an improving move is identified while examining the candidate list for x . Similarly, if no such improving move is found, but an improving move is identified in the process of examining the candidate list for one of the solutions $x[i]$, $i \in S$, then choose the best such move (up to the point where the process is elected to discontinue), and return to the start of step 1.
- (b) If no improving moves are found in step 1A (a), then amend the approach of the original step 1 (b) as follows. Instead of creating each set $M(1:i)$ as the n_1 best moves from $M(0:i)$, coordinate the streams so that the entire

collection $M(1:i)$, $i \in S$, retains only the n_1 best moves from the collection $M(0:i)$, $i \in S$. (Thus, on average, each $M(1:i)$ contains only n_1/s moves. Some of these sets may be empty.) Then set $L = 1$ and proceed to step 2A.

2A. (*Modification of step 2*). Consider each $i \in S$ such that $M(L:i)$ is not empty. For each L-move in $M(L:i)$ identify the associated set $A(m)$ of the n_2 best compatible 1-moves taken from $M(0:i)$, to produce candidate $(L+1)$ -moves of the form $m@m'$ for $m' \in A(m)$.

- (a) If an improving $(L+1)$ -move is found, select such a move as in the original step 2(a) and return to the start of step 1 with a new x .
- (b) If no improving move is found while examining the set of moves in the collection $M(L:i)$, $i \in S$: Stop if L has reached MaxL . Otherwise, identify the collection $M(L+1:i)$, $i \in S$, to consist of the best n_1 moves generated from the entire collection $M(L:i)$, $i \in S$. Then set $L = L + 1$ and return to the start of step 2A.

The Multi-Stream variant requires some monitoring to assure that the sets $M(L:i)$ do not contain duplicate L-moves (i.e., the sets $X(L:i)$ do not contain duplicate solutions), or to assure that such duplications are removed when they occur. Simple forms of TS memory can be used for this purpose, or again the type of screening used to update the set of Reference Solutions for the SS/PR Template (as described in Section 3) can be employed with the method. Special restrictions for level $L = 1$, as previously noted for the single-stream case discussed in the first part of Section 5, can be readily extended to the Multi-Stream case.

The Multi-Stream approach can be joined with the Shift-and-Update variant or the Diversification variant, and is particularly susceptible to being exploited by parallel processing.

8 Conclusions

This paper is intended to support the development of scatter search and path relinking methods, by offering highly specific procedures for executing component routines. We have also undertaken to indicate a type of Improvement Method that exploits the marriage of two candidate list strategies – the Filtration and Sequential Fan strategies – that have been proposed in connection with tabu search, and which invite further examination in their own right. We emphasize that there are additional ways to implement scatter search and path relinking, and our purpose is not to be exhaustive in considering the best alternatives. Nevertheless, the SS/PR Template and its subroutines offer a potential to facilitate the creation of initial methods and to reduce the effort involved in creating additional refinements.

REFERENCES

1. Consiglio, A. and S.A. Zenios (1996). "Designing Portfolios of Financial Products via Integrated Simulation and Optimization Models," Report 96-05, Department of Public and Business Administration, University of Cyprus, Nicosia, CYPRUS, to appear in *Operations Research*. [<http://zeus.cc.ucy.ac.cy/ucy/pba/zenios/public.html>]
2. Consiglio, A. and S.A. Zenios (1997). "a Model for Designing Callable Bonds and its Solution Using Tabu Search." *Journal of Economic Dynamics and Control* 21, 1445-1470. [<http://zeus.cc.ucy.ac.cy/ucy/pba/zenios/public.html>]
3. Crowston, W.B., F. Glover, G.L.Thompson and J.D. Trawick (1963). "Probabilistic and Parametric Learning Combinations of Local Job Shop Scheduling Rules," ONR Research Memorandum No. 117, GSIA, Carnegie Mellon University, Pittsburgh, PA
4. Cung, V-D., T. Mautor, P. Michelon, A. Tavares (1996). "Scatter Search for the Quadratic Assignment Problem", Laboratoire PRISM-CNRS URA 1525. [http://www.prism.uvsq.fr/public/vdc/CONFs/ieee_icec97.ps.Z]
5. Davis, L., ed. (1991). *Handbook of Genetic Algorithms*, Van Nostrand Reinhold.
6. Fisher, H. and G.L. Thompson (1963). "Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules," *Industrial Scheduling*, J.F. Muth and G.L. Thompson, eds., Prentice-Hall. 225-251.
7. Fleurent, C., F. Glover, P. Michelon and Z. Valli (1996). "A Scatter Search Approach for Unconstrained Continuous Optimization," *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, 643-648.
8. Freville, A. and G. Plateau (1986). "Heuristics and Reduction Methods for Multiple Constraint 0-1 Linear Programming Problems," *European Journal of Operational Research*, 24, 206-215.
9. Freville, A. and G. Plateau (1993). "An Exact Search for the Solution of the Surrogate Dual of the 0-1 Bidimensional Knapsack Problem," *European Journal of Operational Research*, 68, 413-421.
10. Glover, F. (1963). "Parametric Combinations of Local Job Shop Rules," Chapter IV, ONR Research Memorandum no. 117, GSIA, Carnegie Mellon University, Pittsburgh, PA.
11. Glover, F. (1965). "A Multiphase Dual Algorithm for the Zero-One Integer Programming Problem," *Operations Research*, Vol 13, No 6, 879.
12. Glover, F. (1968). "Surrogate Constraints," *Operations Research*, 16, 741-749.
13. Glover, F. (1975). "Surrogate Constraint Duality in Mathematical Programming," *Operations Research*, 23, 434-451.
14. Glover, F. (1977). "Heuristics for Integer Programming Using Surrogate Constraints," *Decision Sciences*, Vol 8, No 1, 156-166.
15. Glover, F. (1992). "Ejection Chains, Reference Structures and Alternating Path Methods for Traveling Salesman Problems," University of Colorado. Shortened version published in *Discrete Applied Mathematics*, 1996, 65, 223-253. [<http://spot.colorado.edu/~glover> (under Publications)]
16. Glover, F. (1994a). "Genetic Algorithms and Scatter Search: Unsuspected Potentials," *Statistics and Computing*, 4, 131-140. [<http://spot.colorado.edu/~glover> (under Publications)]
17. Glover, F. (1994b). "Tabu Search for Nonlinear and Parametric Optimization (with Links to Genetic Algorithms)," *Discrete Applied Mathematics*, 49, 231-255. [<http://spot.colorado.edu/~glover> (under Publications)]
18. Glover, F. (1995). "Scatter Search and Star-Paths: Beyond the Genetic Metaphor," *OR Spectrum*, 17, 125-137. [<http://spot.colorado.edu/~glover> (under Publications)]

19. Glover, F., J. P. Kelly and M. Laguna (1996). "New Advances and Applications of Combining Simulation and Optimization," *Proceedings of the 1996 Winter Simulation Conference*, J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain (Eds.), 144-152. [<http://spot.colorado.edu/~glover> (under OptQuest heading)]
20. Glover, F. and M. Laguna (1997). *Tabu Search*, Kluwer Academic Publishers. [<http://spot.colorado.edu/~glover> (under Tabu Search heading)]
21. Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*, Reading, Massachusetts: Addison-Wesley.
22. Greenberg, H. J. and Pierskalla, W.P. (1970). "Surrogate Mathematical Programs," *Operations Research*, 18, 924-939.
23. Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI.
24. Karwan, M.H. and R.L. Rardin (1976). "Surrogate Dual Multiplier Search Procedures in Integer Programming," School of Industrial Systems Engineering, Report Series No. J-77-13, Georgia Institute of Technology.
25. Karwan, M.H. and R.L. Rardin (1979). "Some Relationships Between Lagrangean and Surrogate Duality in Integer Programming," *Mathematical Programming*, 17, 230-334.
26. Kelly, J., B. Rangaswamy and J. Xu (1996). "A Scatter Search-Based Learning Algorithm for Neural Network Training," *Journal of Heuristics*, Vol. 2, pp. 129-146.
27. Laguna, M. and R. Martí (1997). "GRASP and Path Relinking for 2-Layer Straight Line Crossing Minimization," Research Report, University of Colorado [<http://www.bus.colorado.edu/Faculty/Laguna/Papers/crossmin.html>]
28. Laguna, M. (1997). "Optimizing Complex Systems with OptQuest," Research Report, University of Colorado, [<http://www.bus.colorado.edu/Faculty/Laguna/Papers>]
29. Laguna, M., R. Martí and V. Campos (1997). "Tabu Search with Path Relinking for the Linear Ordering Problem," Research Report, University of Colorado. [<http://www.bus.colorado.edu/Faculty/Laguna/Papers/lop.html>]
30. Muhlenbein, H. (1997). "The Equation for the Response to Selection and its Use for Prediction," to appear in *Evolutionary Computation*. [<http://set.gmd.de/AS/ga/ga.html>]
31. Rana, S. and D. Whitley (1997). "Bit Representations with a Twist," Proc. 7th International Conference on Genetic Algorithms, T. Baeck ed. pp: 188-196, Morgan Kaufman. [<http://www.cs.colostate.edu/~whitley/Pubs.html>]
32. Rego, C. (1996). "Relaxed Tours and Path Ejections for the Traveling Salesman Problems," to appear in the *European Journal of Operational Research*. [<http://www.uportu.pt/~crego>]
33. Rego, C. and C. Roucairol (1996). "A Parallel Tabu Search Algorithm Using Ejection Chains for the Vehicle Routing Problem," in *Meta-Heuristics: Theory & Applications*, 661-675, I.H. Osman and J.P. Kelly, (eds.), Kluwer Academic Publishers. [<http://www.uportu.pt/~crego>]
34. Reeves, C.R. (1997). "Genetic Algorithms for the Operations Researcher," to appear in *INFORMS Journal on Computing* (with commentaries and rejoinder).
35. Rochat, Y. and É. D. Taillard (1995). "Probabilistic diversification and intensification in local search for vehicle routing". *Journal of Heuristics* 1, 147-167. [<http://www.idsia.ch/~eric>]
36. Taillard, É. D. (1996). "A heuristic column generation method for the heterogeneous VRP", Publication CRT-96-03, Centre de recherche sur les transports, Université de Montréal. To appear in *RAIRO-OR*. [<http://www.idsia.ch/~eric>]
37. Whitley, D. and J. Kauth, (1988). "GENITOR: A Different Genetic Algorithm," *Proceedings of the 1988 Rocky Mountain Conference on Artificial Intelligence*.

38. Whitley, D. (1989). The GENITOR Algorithm and Selective Pressure: Why Rank Based Allocation of Reproductive Trials is Best, Morgan Kaufmann, J. D. Schaffer, ed., pp. 116-121.
39. Yamada, T. and C. Reeves (1997). "Permutation Flowshop Scheduling by Genetic Local Search," 2nd IEE/IEEE Int. Conf. on Genetic Algorithms in Engineering Systems (GALESIA '97), pp. 232-238, Glasglow, UK.
40. Yamada, T. and R. Nakano (1996). "Scheduling by Genetic Local Search with Multi-Step Crossover," 4th International Conference on Parallel Problem Solving from Nature, 960-969.

APPENDIX 1 Construction-by-Objective: Mixed Integer and Nonlinear Optimization

The generality of the zero-one diversification generators of Section 3 can be significantly enhanced by a *construction-by-objective* approach, which makes it possible to generate points that satisfy constraints defining a polyhedral region. This approach is particularly relevant for mixed integer programming (MIP) problems and nonlinear optimization problems that include linear constraints. The resulting solutions can also be processed by standard supporting software to produce associated points that satisfy integer feasibility conditions that accompany MIP problems.

The approach stems from the observation that a specified set of vectors produced by a zero-one diversification generator can instead be generated by introducing appropriately defined objective functions which are optimized over the space of 0-1 solutions. These same objective functions can then be introduced and optimized, either exactly or heuristically, over other solution spaces, to create a more general process for generating trial solutions.

Denote the set of x vectors that satisfy a specified set of constraining conditions by XC , and denote the set of zero-one vectors by $X(0,1)$. We then identify a *construction function* $f(x)$ relative to a complementary pair of 0-1 solutions x' and x'' so that

$$x' = \operatorname{argmax} (f(x) : x \in X(0,1))$$

$$x'' = \operatorname{argmin} (f(x) : x \in X(0,1)).$$

Such a function can be given by any linear function $f(x) = \sum (f_j : j \in N)$, where $f_j > 0$ if $x'_j = 1$ and $f_j < 0$ if $x'_j = 0$. Let $H\text{-argmax}$ and $H\text{-argmin}$ denote heuristic counterparts of the argmax and argmin functions, which are based on applying heuristic methods that approximately maximize or minimize $f(x)$ over the more complex space XC . Then we define solutions $xMax$ and $xMin$ in XC that correspond to x' and x'' in $X(0,1)$ by

$$xMax = H\text{-argmax} (f(x) : x \in XC)$$

$$xMin = H\text{-argmin} (f(x) : x \in XC)$$

If $f(x)$ is a linear function as specified above, and XC corresponds to a bounded feasible linear programming (LP) region, which we denote by XLP , then the method that underlies $H\text{-argmax}$ and $H\text{-argmin}$ can in fact be an exact method for LP problems, to produce points $xMax$ and $xMin$ that are linear optima over XLP . This observation provides the foundation for a procedure to generate trial solutions for MIP problems.

Following the scatter search strategy of generating weighted centers of selected subregions, we apply the construction-by-objective approach by first identifying a small collection of *primary centers*. These constitute simple centers given by the midpoints of line segments that join pairs of points $xMin$ and $xMax$, together with the

center of all such points generated. Additional *secondary centers* are then produced by an accelerated process that does not rely on the exact solution of LP problems, but employs the primary centers to create approximate solutions. Both processes are accompanied by identifying *subcenters*, which are weighted centers of subregions composed by reference to the primary and secondary centers, and include the boundary points $xMax$ and $xMin$.

Method to Create Primary Centers

1. Identify a construction function $f(x)$ and initialize the set PC of primary centers and the set SubC of subcenters to be empty. Apply a Diversification Generator to produce a small collection of diversified points $x' \in X(0,1)$. For each point generated, execute the following steps.
2. Identify points $xMax$ and $xMin$ by maximizing and minimizing $f(x)$ over XLP .
3. Let $xCenter = (xMax + xMin)/2$ and add $xCenter$ to PC. Also, create associated subcenters $xSubCenter = xMax + w(xMax - xMin)$, where the scalar weight w takes the values 0, 1/4, 3/4 and 1, and add these subcenters to the set SubC.
4. After the chosen x' vectors have been generated and processed, create a final point $xCenter^*$ which is the mean of all points $xMin$ and $xMax$ generated, and add $xCenter^*$ to PC.

The accelerated process for generating secondary centers, which augments the preceding method, continues to rely on a zero-one diversification generator to create 0-1 points $x' \in X(0,1)$, but bypasses LP optimization. The 0-1 points are first mapped into points of an associated region $X(0,U)$, where U is a vector of upper bounds chosen to assure that $0 \leq x \leq U$ is satisfied by all solutions $x \in XLP$. Then each point $x' \in X(0,1)$ produces a corresponding point $xTest \in X(0,1)$ by specifying that $x'_j = 0$ becomes $xTest_j = 0$, and $x'_j = 1$ becomes $xTest_j = U_j$.

Since these created points $xTest \in X(0,U)$ are likely to lie outside XLP , we map them into points of XLP by reference to the elements of PC. For a given $xTest$ and a given primary center $xCenter \in PC$, identify the line

$$L(w) = xTest + w(xCenter - xTest)$$

where w is a scalar parameter. Then a simple calculation identifies the values

$$wMin = \min(w : L(w) \in XLP)$$

$$wMax = \max(w : L(w) \in XLP)$$

The associated points of XLP , which are given by

$$xNear = L(wMin)$$

$$xFar = L(wMax),$$

constitute the two points of XLP nearest to and farthest from $xTest$ on the line through $xCenter$.

The secondary centers are chosen as midpoints of the line segments joining the endpoints $xNear$ and $xFar$. However, since such line segments may be created from

more than one primary center $xCenter$, we limit consideration for a given $xTest$ to the primary center that creates the longest of the indicated segments, i.e., which maximizes $D(xNear, xFar)$, where D is a distance metric such as the L1 or L2 norm. The method may be described as follows.

Method for Generating Secondary Centers

1. Generate a collection of points $x' \in X(0,1)$ and map each into an associated point $xTest \in X(0,U)$. Begin with the set SC of secondary solutions empty.
2. For each $xTest$, choose one or more elements $xCenter$ from PC, including $xCenter^*$.
3. For each $xCenter$ chosen, identify the pair of solutions $xNear$ and $xFar$ (by reference to $xCenter$ and $xTest$), and identify the specific pair $xNear^*$ and $xFar^*$ to be the pair that maximizes $D(xNear, xFar)$.
4. Let $xNewCenter = (xNear^* + xFar^*)/2$, and add $xNewCenter$ to SC. Also, create associated subcenters given by $xSubCenter = xNear + w(xFar - xNear)$, as the scalar weight w takes the values 0, 1/4, 3/4 and 1, and add these to SubC.

To take advantage of generating larger numbers of points x' by the Diversification Generator, the foregoing process may be modified to create each secondary center from multiple $xTest$ points. Specifically, for a chosen number *TargetNumber* of such points, accumulate a vector $xSum$ that starts at 0 and is incremented by $xSum = xSum + xNear^* + xFar^*$ for each pair of solutions identified in Step 3 of the foregoing procedure. Once *TargetNumber* of such points have been accumulated, set $xNewCenter = xSum/TargetNumber$, and start again with $xSum$ equal to the zero vector. The overall primary center $xCenter^*$ may optionally be updated in a similar fashion.

The values of w for generating subcenters can be varied, and used to generate more or fewer points. An alternative to choosing $xNear^*$ and $xFar^*$ by maximizing $D(xNear, xFar)$ is to specify that $xNear^*$ is selected to minimize $D(xTest, xNear)$. This latter criterion is also relevant for mapping an LP-infeasible point $xTest$ generated by a scatter search combination process into an LP-feasible point $xNear^*$, similarly using candidate points $xCenter$ as a basis for determining associated points $xNear$. The process for transforming the points ultimately selected into solutions that are also integer-feasible is examined next.

Creating Reference Solutions

The primary centers, secondary centers and subcenters provide the source solutions which, after improvement, become candidates for the set RefSet of reference solutions to be combined by scatter search. To create an initial RefSet, we cull out an appropriately dispersed subset of these source solutions by creating a precursor set PreRefSet, whose members will be made integer-feasible and hence become full fledged candidates for RefSet. This may be done by starting with the primary center

$xCenter^*$ as the first element of PreRefSet, and then applying the criterion indicated in Section 3 to choose each successive element of PreRefSet to be a source solution that maximizes the minimum distance from all solutions thus far added to PreRefSet.

Once PreRefSet reaches its targeted size, the final step is to modify its members by an adaptive rounding process that yields integer values for the discrete variables. For problems small enough that standard MIP software can be expected to generate feasible MIP solutions within a reasonable time, such software can be used to implement the rounding process by introducing an objective function that minimizes the sum of deviations of the integer values from the initial values. (The deviations may be weighted to reflect the MIP objective, or to embody priorities as used in tabu search.) Alternatively, a simple adaptive scheme of the following form can be used to exploit such an objective.

Adaptive Rounding Method to Create Reference Solutions

1. For each integer-infeasible variable in a given candidate solution, introduce a bound at an integer value neighboring its current value, and establish a large LP penalty for deviating from this bound. (Variables not constrained to be integer retain their ordinary objective function coefficients, or can be mildly penalized for deviating from their current values.)
2. Find an LP optimal solution for the current objective. If all integer-constrained variables receive integer values, stop. (The resulting solution is the one sought.)
3. If the LP solution is not MIP feasible, use postoptimality penalty calculations for each integer-constrained variable to identify the cost or profit that results by releasing its current penalty and seeking to drive the variable to a new bound at the closest integer value in the opposite direction from its current bound. Choose the variable that yields the greatest profit (or smallest cost) and impose a penalty for deviating from the indicated new bound. Then return to Step 2.

A natural priority scheme for Step 3 of the preceding approach is to give preference to selecting integer-infeasible variables to be those driven to new bounds, though this type of priority is not as relevant as it is in branch and bound methods. A simple tabu search memory can be used as a basis for avoiding cycling, while allowing greater flexibility of choices than provided by branch and bound.

After applying such a rounding procedure, the transformed members of PreRefSet are ready to be submitted to the customary scatter search processes of applying an Improvement Method and generating combined solutions. (Improvement heuristics can be included as part of the transformation process.) Future diversification phases of the search may be launched at various junctures by following the same pattern as indicated, where the diversification generator may be started from the point where it was discontinued in the preceding phase, or re-started by reference to a different seed solution. The initial PreRefSet for such future phases is populated by elements chosen to maximize the minimum distance from the collection of points previously generated

as members of RefSet and PreRefSet, as well as from members currently added to PreRefSet.

APPENDIX 2: Checking for Duplicate Solutions

An additional source of potential duplications arises among solutions x' that are generated as combinations of other solutions (in the phase of Scatter Search or Path Relinking that generates such combined solutions). These solutions x' are inputs to the Improvement Method rather than outputs of this method. By the philosophy of scatter search and path relinking, it is valuable to avoid duplications in these input solutions as well as to avoid duplications in the solutions saved in RefSet. To do this, we store only the $r = rNow$ most recent solutions generated (allowing $rNow$ to grow to a maximum of $rMax$ different solutions recorded), following a scheme reminiscent of a simple short-term recency memory approach in tabu search. In particular, we keep these solutions in an array $xsave[r]$, $r = 1$ to $rNow$, and also keep track of a pointer $rNext$, which indicates where the next solution x' will be recorded once the array is full, i.e., once all $rMax$ locations are filled.

Let $E0$ and $Hash0$ be defined for x' as before, and denote associated values for the $xsave[r]$ array by $Esave(r)$ and $Hashsave(r)$. These are accompanied by a “depth” value, which is 0 if no duplication occurs, and otherwise tells how deep in the list – how far back from the last solution recorded – a duplication has been found. For example, $depth = 3$ indicates that the current solution duplicates a solution that was recorded 3 iterations ago. (This is not entirely accurate, since, for example, $depth = 3$ could mean the solution was recorded 5 iterations ago and then 2 other duplications occurred, which still results in recording only 3 solutions.)

An appropriate value for $rMax$ can be determined by initial testing that sets this value larger than expected to be useful. An array $CountDup(depth)$, for $depth = 1$ to $rMax$, can then be kept that counts how often duplications are found at various depths. If the array discloses that very few duplications occur for depths beyond a given value, then $rMax$ can be reduced to such a value, without the risk of having to process many solutions that duplicate others encountered. (Although the reduced value of $rMax$ will save some effort checking for duplications, it may be the effort will not be too great anyway, if a quick check based on using $Hash0$ can screen out most of the potential duplications.)

To keep track of auxiliary information we introduce counters corresponding to $DupCheck$, $FullDupCheck$ and $FullDupFound$ of the RefSet Update Routine, which we give the names $DupCheckA$, $FullDupCheckA$, and $FullDupFoundA$. Finally, we keep track of the number of times the routine is called by a value $DupCheckCall$.

Initialization Step:

$rNow = 0$
 $rNext = 0$
 $CountDup(depth) = 0$, for $depth = 1$ to $rMax$
 $DupCheckA = 0$
 $FullDupCheckA = 0$

FullDupFoundA = 0

DupCheckCall = 0

Duplication Check Subroutine.

Begin Subroutine.

DupCheckCall = DupCheckCall + 1

depth = 0

If rNow = 0 then:

 rNow = 1; rNext = 1;

 xsave[1] = x' (record x' in xsave[1]),

 Esave(1) = E0; Firstsave(1) = FirstIndex0

 End the Subroutine

Elseif rNow > 0 then:

(Go through the solutions in “depth order”, from the one most recently stored to the one least recently stored. When a duplication is found, the loop index r (below) indicates the value of rMax that would have been large enough to identify the duplication.)

 i = rNext

 For r = 1 to rNow

 If Esave(i) = E0 then:

 DupCheckA = DupCheckA + 1

 If Hash0 = Hashsave(i) then:

 FullDupCheckA = FullDupCheckA + 1

 If $x' = x[i]$ then:

 (x' duplicates a previous solution)

 FullDupFoundA = FullDupFoundA + 1

 depth = r

 CountDup(depth) = CountDup(depth) + 1

 End the Duplication Check Subroutine

 Endif

 Endif

 Endif

 i = i-1

 if i < 1 then i = rNow

End r

(Here, no solutions were duplicated by x' . Add x' to the list in position rNext, which will replace the solution previously in rNext if the list is full.)

rNext = rNext + 1

If rNext > rMax then rNext = 1

If rNow < rMax then rNow = rNow + 1

xsave[rNext] = x'

Esave(rNext) = E0

Hashsave(rNext) = Hash0

Endif

End of Duplication Check Subroutine

An Analysis of Punctuated Equilibria in Simple Genetic Algorithms

Sangyeop Oh and Hyunsoo Yoon

Dept. of CS & CAIR, KAIST, Daejeon, Korea
{syoh,hyoon}@camars.kaist.ac.kr

Abstract. In the running of a genetic algorithm, the population is liable to be confined in the local optimum, that is the metastable state, making an equilibrium. It is known that, after a long time, the equilibrium is punctuated suddenly and the population transits into the better neighbor optimum. We adopt the formalization of Computational Ecosystems to show that the dynamics of the Simple Genetic Algorithm is represented by a differential equation focusing on the population mean of a phenotype. Referring to the studies of differential equations of this form, we show that the duration time of metastability is exponential in the population size and other parameters, on the one dimensional bistable fitness landscape which has one metastable and one stable state.

1 Introduction

Genetic Algorithms (GAs) are optimization methods modeled from some operations which are used during the natural reproduction and the natural selection [3]. Since the concepts of GAs are introduced by Holland [5], various GAs show practical successes in various fields. Among them, the Simple Genetic Algorithm (SGA) is the simplest genetic algorithm containing the essential operators : selection, mutation, and crossover.

Like the other optimization methods, SGAs have the problem of metastability that the population is liable to be trapped in a local optimum, making an equilibrium. If there is a better optimum state in the vicinity, the local optimum is the metastable state since the punctuated equilibrium appears [12]. The punctuated equilibrium is the phenomenon in which the system in a metastable state shows a sudden transition into the more stable neighbor state after a long time. Punctuated equilibria are analyzed in various fields including Computational Ecosystems (CEs) and neo-Darwinian evolution models [1, 9].

The analysis of CEs is based on the time derivative of the probability distribution P , where $P(\mathbf{r}, t)$ is the probability that the system state is $\mathbf{r}$ at time t in the ensemble of possible system states. They use dP/dt to find $d \langle z \rangle / dt$ where $\langle z \rangle$ is the ensemble mean of the interested system character z [6].

In the neo-Darwinian model, the population mean $\bar{x}$ of a genetically determined character is governed by the equation $d\bar{x} = F'(\bar{x}) + \varepsilon dB$ where F is a landscape on x , $B(t)$ is a standard Brownian process, and ε is a small constant. They use the theoretical results from diffusion processes [8] to show that

the punctuated equilibria appear in the natural evolution with the exponential duration time of metastability as ε decreases to zero [9].

In this paper, we adopt the formalization of Computational Ecosystems to show that the dynamics of the Simple Genetic Algorithm is represented by a differential equation like $d\bar{x} = F'(\bar{x}) + \varepsilon dB$ focusing on the population mean of a phenotype. Referring to the studies of differential equations of this form, we show that the duration time of metastability is exponential in the population size and other parameters, on the one dimensional bistable fitness landscape which has one metastable and one stable state.

In section 2, the CE and the equation from the diffusion process are described. We analyze the dynamics of the SGA adopting the methods of CEs in section 3. The bistable fitness landscape is introduced and the simulation results are shown in section 4. In section 5 we discuss the results obtained in previous sections focusing on the duration time of metastability. Conclusion and further work are covered in section 6.

2 Background

2.1 Computational Ecosystems

The CE is a very similar model to GAs. A population contains N agents, and each agent chooses one from R resources to get some payment which is determined by the payment function f . f is the function of the chosen resource and the population state which is represented by a vector $\mathbf{r} = (r_1, r_2, \dots, r_R)$ whose i -th element is the ratio of agents choosing the resource i . During a unit time, each agent has the α chances to change the resource to the new one according to ρ_i , which is the probability that resource i perceived to be the best choice and is the function of $\mathbf{r}$.

The possible population states at time t compose the ensemble represented by $P(\mathbf{r}, t)$ which is the probability of the population state to be $\mathbf{r}$ at t . Considering only one possible change of an agent's resource in a short time interval,

$$\frac{dP(\mathbf{n}, t)}{\alpha dt} = -P(\mathbf{n}, t) \sum_{i \neq j} n_j \rho_i + \sum_{i \neq j} P(\mathbf{n}^{[j, i]}, t) (n_j + 1) \rho_i^{[j, i]} \quad (1)$$

where $\mathbf{n}^{[j, i]}$ is such that $n_j^{[j, i]} = n_j + 1$, $n_i^{[j, i]} = n_i - 1$ and all other elements are the same as $\mathbf{n}$. And ρ_i and $\rho_i^{[j, i]}$ are evaluated at $\mathbf{n}$ and $\mathbf{n}^{[j, i]}$, respectively.

The ensemble mean $\langle r_i \rangle$ of r_i satisfies

$$\frac{d \langle r_i \rangle}{dt} = \alpha (\langle \rho_i \rangle - \langle r_i \rangle). \quad (2)$$

using equation (1) [6].

2.2 Diffusion processes

Consider the one dimensional diffusion process $z(t)$ satisfying

$$dz(t) = F'(z) + \varepsilon dB(t) \quad (3)$$

where $B(t)$ is the standard Brownian process and ε is a small constant. Let F satisfy the following conditions :

- F is a differentiable function defined on $-\infty < z < \infty$,
- there exists $z_1 < z_2 < z_3$ such that F is strictly increasing on $(-\infty, z_1] \cup [z_2, z_3]$ and strictly decreasing on $[z_1, z_2] \cup [z_3, \infty)$, and
- $F(z_1) < F(z_3)$.

If $z(0) = z_1$, then the punctuated equilibrium appears and the duration time T of metastability satisfies

$$T \propto \exp\left(\frac{2(F(z_1) - F(z_2))}{\varepsilon^2}\right) \quad (4)$$

when $\varepsilon \ll 1$. And the transition is unidirectional in the sense that the system remains in the stable state permanently [8]. These results about the duration time can be expanded to the cases that F has more than two peaks or z is multidimensional [8, 2].

3 Analysis

The analysis of genetic algorithms in the genotype is somewhat complex since it considers vector values and matrix operations. We focus on the phenotype x_i of an individual where x_i corresponds to the genotype i .

3.1 Simple Genetic Algorithms

A SGA in this paper deals with a population which consists of N individuals. Each individual is a binary string of L bits, each bit has one of two values 0 and 1, and there are $R = 2^L$ genotypes. The phenotype for the genotype i is x_i and the fitness f is the function on the phenotype domain. The population state is represented by $\mathbf{n} = (n_1, n_2, \dots, n_R)$ or $\mathbf{r} = (r_1, r_2, \dots, r_R)$ where $n_i(t)$ is the number of individuals with the genotype i and $\mathbf{r} = \mathbf{n}/N$.

The population of the next generation is produced from the current one through some SGA operators : *roulette wheel selection*, *1-point crossover* and *simple mutation* [3]. After two individuals are selected from the current population by the roulette wheel selections, the 1-point crossover and the simple mutation are applied to the pair. In the 1-point crossover, each individual is cut at the same point and divided into two substrings, and then the second substrings are exchanged. The cutting point is determined randomly among all the points between two bits. And then, the simple mutation toggles each bit of

individuals with the probability p_m . After the mutations, one of two children is chosen randomly and added into the next generation.

Repeating this process N times, the new generation with the generation gap 1 can be completely produced. For the simplicity, the crossover is not considered in the analysis and is mentioned in section 5.2. The parameters and the functions related to selection, crossover and mutation are superscripted with s , c and m respectively.

3.2 Brownian part

As it is defined in the CE, let $P(\mathbf{r}, t)$ be the probability that the population state is $\mathbf{r}$ at time t . Then the ensemble mean and the population mean of z are represented, respectively as $\langle z \rangle = \sum_{\mathbf{r}} z P(\mathbf{r}, t)$ and $\bar{z} = \sum_i z r_i$.

Let α be the average number of generations per unit time. Then the SGA changes the population state $\alpha \Delta t$ times during Δt . When an individual is chosen after selection, crossover and mutation, the phenotype of the individual can be considered as a random variable X . Using the central limit theorem [11], the population mean $\bar{X}$ of the phenotypes follows

$$\bar{x}(t + \Delta t) - \bar{x}(t) = \langle \bar{x}(t + \Delta t) \rangle - \bar{x}(t) + \alpha \Delta t G \quad (5)$$

where G is a Gaussian random variable with mean 0 and variance $V(\bar{X})$.

Since the dispersion of the ensemble starts from the instantiated state with $\bar{x}(t)$, $\langle \bar{x}(t) \rangle = \bar{x}(t)$. The accumulation of Gaussian random variables, each of which has the variance 1, makes the standard Brownian process. Then, since $V(\bar{X}) = V(X)/N$ by the central limit theorem,

$$d\bar{x}(t) = d \langle \bar{x}(t) \rangle + \alpha \sqrt{\frac{V(X)}{N}} dB(t). \quad (6)$$

Ignoring the crossover, the random variable X of phenotype is composed of X^s and X^m ,

$$X = X^s + X^m \quad (7)$$

where X^s is the result of the selection and X^m is the change due to the mutation. The effects of the mutation is dependent on how the genotype is decoded into the phenotype, and then into the fitness. To obtain a general feature of the simple mutation, we define the phenotype as $x_i = l_i/L$, where l_i is the number of bits with the value 1 in the genotype i . In case of $L = 1$, X^m has the probability distribution with mean $(1 - 2l(t)/L)p_m$ and variance $p_m - (1 - 2l(t)/L)^2 p_m^2$. Thus, in general, X^m has the Gaussian distribution with

$$\begin{aligned} E(X^m) &= (1 - 2l(t)/L)p_m \quad \text{and} \\ V(X^m) &= [p_m - (1 - 2l(t)/L)^2 p_m^2]/L \simeq p_m/L \end{aligned} \quad (8)$$

by the central limit theorem.

Considering only the selection, the variance of the phenotype seems to be dependent on $f^{(1)}/\bar{f}$, where $f^{(m)}$ is the m -th derivative of $f(x)$. Even when the

fitness landscape is flat, the selection makes the variance decrease by the factor of $1/N$ [10], and then

$$0 < V_{steepest} \leq V(X) \leq V_{flat} = \frac{Np_m}{L} \quad (9)$$

where $V_{steepest}$ and V_{flat} are variances when $|f^{(1)}/\bar{f}|$ is maximum and zero, respectively. That is, $V(X)$ is finite within the range of equation (9).

3.3 Ensemble mean part

Consider the case of the generation gap $1/N$, in which a child is produced from the SGA operators, a victim is chosen randomly from the current generation and then it is replaced by the child, producing the next generation. When genotypes of the child and the victim are i and j respectively, this random event corresponds with the resource changing from j to i for an agent in CEs. That is, the SGA with the generation gap $1/N$ is the special case of the CE, when ρ_i is interpreted as the probability that the child with the genotype i would be produced by the SGA operations.

However, if the generation gap is 1, equation (1) can not be used by the SGA since it is the result of considering maximally one change of an agent in a given time interval. Thus we focus on the macroscopic equation (2) whose right hand side is interpreted as $N\alpha$ generations per unit time multiplied by the effect of one generation, $(\langle \rho_i \rangle - \langle r_i \rangle)/N$ on condition that the generation gap is $1/N$. If one generation contains N changes of individuals and they are all relatively independent, the effect of one generation is enlarged to be $(\langle \rho_i \rangle - \langle r_i \rangle)$. Strictly speaking, the changes of individuals in one generation of the SGA is not independent since the victim is selected in round robin. But it makes the same effect to the independent case since the round robin guarantees that each individual has the same probability to be selected as a victim. Modifying the definition of α as the average number of generations per unit time, equation (2) can also be used for the SGA with the generation gap 1.

From equation (2), we can obtain

$$\frac{d\langle \bar{x}(t) \rangle}{dt} = \alpha \sum_{i=1}^R x_i (\rho_i(t) - r_i(t)). \quad (10)$$

Considering only the selection,

$$\sum_{i=1}^R x_i \left(\frac{f(x_i)r_i}{\bar{f}} - r_i \right) = \frac{\overline{(x - \bar{x})f}}{\bar{f}} = \frac{1}{\bar{f}} \sum_{m=0}^{\infty} \frac{f^{(m)}(\bar{x})}{m!} \overline{(x - \bar{x})^{m+1}}. \quad (11)$$

On condition that

$$f(x) \simeq f(\bar{x}) + (x - \bar{x})f^{(1)}(\bar{x}) \quad (12)$$

for each individual in the population, equation (11) becomes

$$\sum_{i=1}^R x_i \left(\frac{f(x_i)r_i}{\bar{f}} - r_i \right) = s_X^2 \frac{\partial \log f(\bar{x})}{\partial \bar{x}} \quad (13)$$

where $s_X^2 = \overline{(x - \bar{x})^2}$, the population variance.

The change due to the mutation is, from equation (8),

$$\left[\frac{d < \bar{x}(t) >}{\alpha dt} \right]^m = p_m(1 - 2\bar{x}^s) \simeq p_m(1 - 2\bar{x}). \quad (14)$$

3.4 Dynamics due to selection and mutation

Let V^s be the some constant representing the selection term of $V(X)$ within the range of equation (9). If we assume that s_X^2 is independent of $\bar{x}$ for the simplicity then the global fitness function F can be defined by

$$F(\bar{x}) = s_X^2 \log f(\bar{x}) + p_m(\bar{x} - \bar{x}^2). \quad (15)$$

This assumption will be mentioned again in section 5.3. Finally, from equations (6), (8), (13) and (14), we can summarize the dynamics of the SGA as

$$\frac{d\bar{x}(t)}{\alpha dt} = \frac{\partial F(\bar{x})}{\partial \bar{x}} + \sqrt{\frac{V^s + p_m/L}{N}} \frac{dB(t)}{dt} \quad (16)$$

considering selections and mutations except the crossover.

4 Simulation

4.1 Bistable landscape

Punctuated equilibria appear if F has the landscape as is shown in section 2.2. In this simulation, we restrict f to have three features. First, F satisfies the conditions in section 2.2 except that F is not differentiable at finite number of points. Secondly, the selection pressures from the barrier are the same for the metastable and stable attractors. And lastly, the condition of equation (12) is satisfied for each individual. Strictly speaking, the third is not always true since there are some generations in which some individuals are in the right side of the barrier and the others remain in the left side. But this dispersion is condensed into a population in one side very quickly and hence we ignore the effect. The phenotype x is defined by $x_i = l_i/L$ as in the case of mutation analysis.

The typical f and F are shown in figure 1, where $d = f(0) - f(1/3)$ is the barrier depth in f landscape. Other fitness function like the generalized deceptive functions [13] could also be used if it has the deceptiveness and the multistep trajectory from the local optimum to the global optimum state, but are not covered in this paper.

Figure 2 shows typical punctuated equilibria appeared in the running of the SGA. The population starts from the state $\bar{x} = 0$, converges quickly into the metastable state, and shows perturbations around it. After a long duration, $\bar{x}$ transits the fitness barrier to the stable state suddenly.

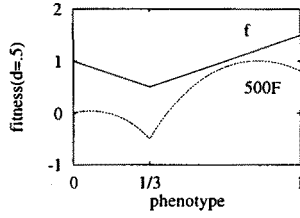


Fig. 1. A fitness function with barrier depth d

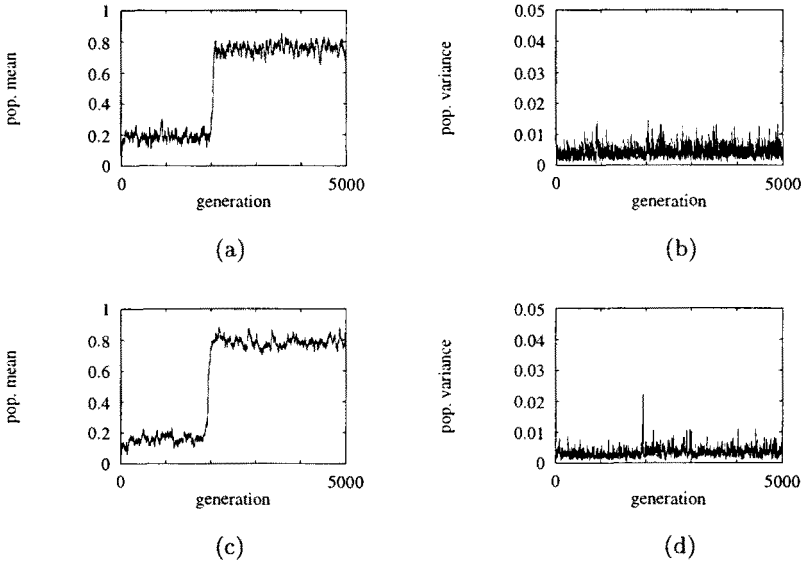


Fig. 2. The punctuated equilibria are shown on the graph of $\bar{x}$ as generations are succeeded. (a) and (b) are $\bar{x}$ and s_X^2 respectively, where $N = 20$, $p_c = 1.0$, $p_m = 0.012$ and $d = 0.5$. (c) and (d) are $\bar{x}$ and s_X^2 respectively, where $N = 30$, $p_c = 0.0$, $p_m = 0.008$ and $d = 0.5$.

4.2 The time of metastability

Beginning with the population state with $\bar{x} = 0$, we record the duration T of metastability varying some parameters, where T is defined as the number of generations till the transition to the stable state occurs. The time required for the transition is so short relative to T that it is ignored. Considered parameters are the population size N , the barrier depth d in f landscape, and the mutation probability p_m .

Figure 3 is the simulation results when $p_c = 0$. Figure 3(a) and 3(b) shows that T is a rapidly increasing function of N and d . And figure 3(c) shows that T is a rapidly decreasing function of p_m .

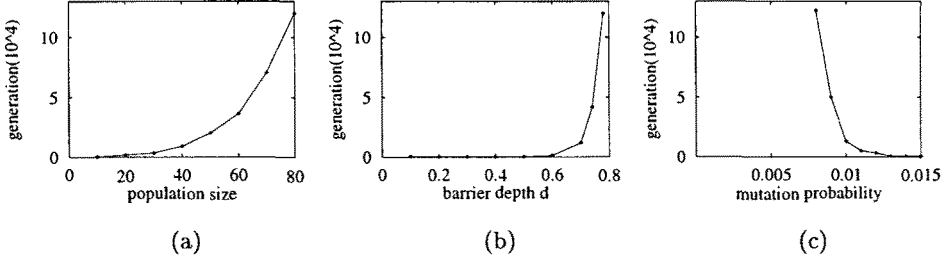


Fig. 3. The duration T of metastability versus some parameters. Each point is an average of 30 runs. Default parameter values are $L = 30$, $N = 40$, $p_c = 0$, $p_m = 0.01$ and $d = 0.7$.

In case of $p_c = 1$, figure 4 is the counterpart of figure 3. This shows that the crossover makes the duration be longer for any parameter set in this simulation environment.

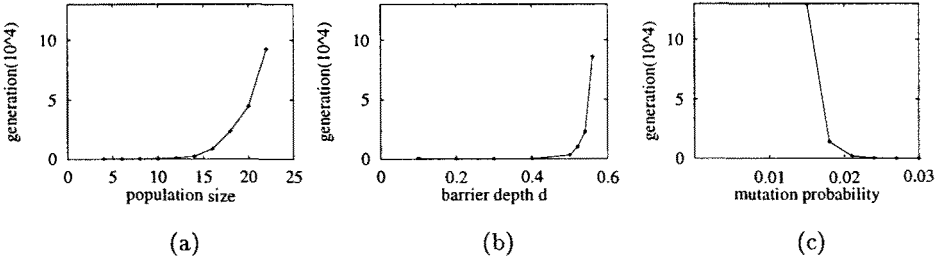


Fig. 4. The duration T of metastability versus some parameters. Each point is an average of 30 runs. Default parameter values are $L = 30$, $N = 40$, $p_c = 1$, $p_m = 0.01$ and $d = 0.7$.

5 Discussion

5.1 Effects of selection and mutation

Since equation (16) has the form of equation (3), the duration T of metastability satisfies

$$T \propto e^{\frac{2DN}{V^s + p_m/L}} \quad (17)$$

by equation (4), where D is the barrier depth in F landscape.

The mean population phenotype $\bar{x}(t)$ oscillates around the equilibrium value x_e which is determined from the non-Brownian part of equation (16), $\partial F(\bar{x})/\partial \bar{x} = 0$. Given p_m and d , the relation between s_X^2 and x_e can be obtained from this equation and is approximately consistent with the graphs of figure 2.

Among the SGA runs in the simulation, there is not any case that the population returns to the metastable state once the punctuated equilibrium occurs, provided that $p_m \ll 1$. This means that the transition is unidirectional.

Figure 3(a) qualitatively confirms that T is exponential in the population size N . But figure 3(b) needs an explanation since the barrier depth d is the quantity of f landscape. Let x_e be the root of $\partial F(\bar{x})/\partial \bar{x} = 0$ in the metastable area. Then, from equation (17),

$$T \propto e^{K[\log f(x_e) - \log(1-d)]} = \left[\frac{f(x_e)}{1-d} \right]^K \quad (18)$$

where K is a constant. That is, the shape of the graph of T is that of $1/(1-d)$ rather than that of e^d . Figure 3(c) shows that why the condition of $p_m \ll 1$ is needed for the punctuated equilibria to be appeared, and is consistent to equation (17).

5.2 Effects of crossovers

In a bistable problem, individuals can be divided into two types according to which basin of attractor they belong to : A -type and B -type, respectively in metastable and stable area. Let r_B and f_B be the ratio and the average fitness of B -type individuals, respectively.

For most generations, $r_B = 0$, and r_B becomes positive at long intervals by crossovers or mutations. When this event occurred at τ -th generation,

$$r_B(\tau + 1) \simeq [r_B^s(\tau)]^2 + \beta r_A^s(\tau) r_B^s(\tau) \quad (19)$$

since the term of $[r_A^s(\tau)]^2$ and the effect of mutation can be ignored. The parameter β is the rate that B -type children are produced from the crossover of one A -type and one B -type parents.

When the phenotype is defined as $x = l/L$, the crossover has the tendency that the x 's of two parents are averaged. If one A -type and one B -type parents crossover then the children would be around the barrier terminating the appearance of the B -type, and hence $\beta \ll 1$. That is, the crossover not only enhances the appearance of B -type individual, but also would eliminate it. Figure 5 shows that the crossover interrupts the appearance of the B -type as a whole if the genotype is decoded into the fitness as is done in section 4. On condition that β is larger than a particular criterion, this interrupt would be replaced by the enhancement. But the crossover is highly dependent on the definitions of x and f , and the generalization is not considered in this paper.

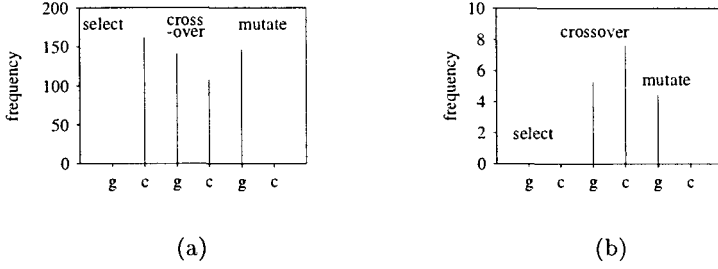


Fig. 5. The individual with the maximal phenotype in the population is traced. It sometimes goes and comes across the particular phenotype criterion value after each SGA operation. The criterion is $1/3$ for (a), and $14/30$ for (b). For the parameters in this simulation, the individuals in $1/3 \leq x \leq 14/30$ disappear by the selections. Hence we focus on (b) to elucidate the contributions of GA operations to the transition into the stable state. The number of goings or comings in a run is counted until the transition occurs, and then averaged over 100 runs. g and c represent 'going' and 'coming', respectively. Parameter values are $L = 30$, $N = 20$, $p_m = 0.012$ and $d = 0.5$.

5.3 Variance of phenotype

Since the population variance s_X^2 is the sample variance corresponding to the ensemble variance $V(X)$, the sampling of s_X^2 has the distribution with mean $E(s_X^2) = V(X)$ and variance $V(s_X^2) = 2(V(X))^2/(N - 1)$, in which $V(s_X^2)$ represents the sampling perturbation [11].

The selection makes $V(X)$ decrease according to the relative gradient of f , $(\partial f / \partial x) / \bar{f}$. As the relative gradient of f decreases, the selection force decreases and $V(X)$ increases. However the increment is small relative to the change of $\log f$ as shown in figure 6 and this supports the assumption for equation (15).

As the population size N increases, the diversity of genotypes within the population increases but it does not directly mean that the phenotype variance increases. Instead, figure 7 shows two things that, as N increases, the increment of $V(X)$ is negligibly small compared with the linear increment, and the amplitude of sampling perturbations of s_X^2 decreases as predicted above. The first supports the assumption that V^s is approximately independent of N . The second means that, as N increases, the effect of increasing $V(X)$ could be dominated by that of decreasing $V(s_X^2)$ in the small range of N .

6 Conclusion and future work

In this paper, we have analyzed the dynamics of the SGA to get a differential equation about the population mean of phenotype values. It is divided into the Brownian part and non-Brownian part by the central limit theorem, as shown in equation (6). Detailing the equation, we have considered selections and mutations except the crossover for the simplicity, and the result is equation (16).

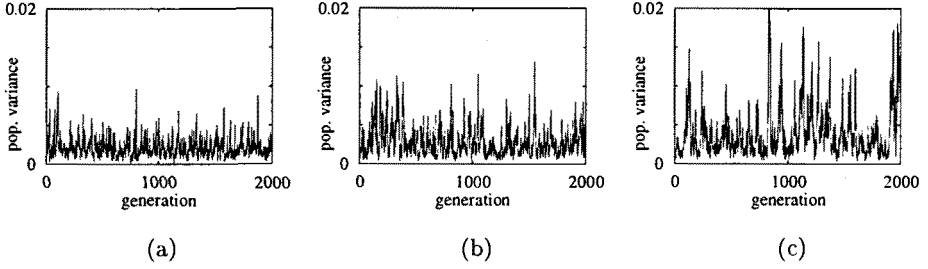


Fig. 6. The population variance s_X^2 of phenotype at each generation. Parameter values are $N = 20$, $p_c = 0$ and $p_m = 0.01$ except that $d = 0$ and $d = 0.5$ for (a) and (b) respectively. The fitness function of (c) is the same as that of (b) except that the gradient is increased to 1500 on $x > 1/3$. The population goes into the stable area after several generations in these cases, where the gradients are 0, 1.5 and 1500 for (a), (b) and (c), respectively.

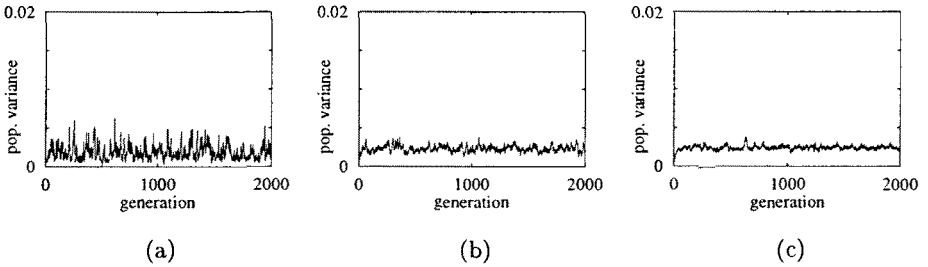


Fig. 7. The population variance s_X^2 of phenotype at each generation. Parameter values are $p_c = 0$, $p_m = 0.005$ and $d = 0.5$, except that $N = 40$, $N = 400$, and $N = 1600$ for (a), (b) and (c), respectively.

The non-Brownian part is the dynamics of the ensemble mean of phenotype values and has been analyzed adopting the method of the CE. The effect of roulette wheel selections is proportional to the population variance and the logarithm of the fitness function in condition of equation (12). The effects of mutations and crossovers are dependent on how the genotype is decoded into the phenotype, and then into the fitness. When the phenotype is defined as $x = l/L$, the effect of simple mutations is proportional to the mutation probability and $(1/2 - \bar{x})$.

For the equation (16) has the typical form which has been analyzed in diffusion processes, we can adopt the theoretical results from them. That is, running the SGA on the bistable global fitness landscape F which is defined as equation (15), (i) the punctuated equilibrium appears, (ii) the duration time of metastability is exponential as shown in equation (17), and (iii) the transition is unidirectional. These theoretical results are qualitatively confirmed by the simulation results.

These results about the duration time could be expanded to the cases that F has more than two peaks or x is multidimensional [8, 2].

Though this paper shows some interesting results, it has some defects to be supplemented. The major one is that the effect of crossovers is just roughly analyzed. When the phenotype is defined as $x = l/L$, 1-point crossovers make the duration of metastability be longer. The next one is that, we have regarded the variance of the phenotype $V(X)$ to be constant, on a basis of simulation results. For the more accurate analysis, the relation between $V(X)$ and GA parameters such as the population size and the gradient of the fitness landscape should be examined.

Since GAs have direct relations with CEs, the results obtained from the studies of CEs could be applied to GAs. These include the issues about time delay, cooperation, competition, chaos, and so on [7, 4]

References

1. H. A. Ceccatto and B. A. Huberman. Persistence of nonoptimal strategies. *Proc. Natl. Acad. Sci. USA*, 86:3443–3446, 1989.
2. A. Galves, E. Olivieri, and M. E. Vares. Metastability for a class of dynamical systems subject to small random perturbations. 15(4):1288–1305, 1987.
3. D. E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, 1989.
4. Tad Hogg and Bernardo A. Huberman. Controlling chaos in distributed systems. *IEEE trans. on SMC*, 21(6):1325–1332, 1991.
5. J. H. Holland. *Adaptation in natural and artificial systems*. Univ. of Michigan Press, 1975.
6. B. A. Huberman and T. Hogg. The behavior of computational ecologies. In B. A. Huberman, editor, *The Ecology of Computation*. North-Holland, 1988.
7. J. O. Kephart, T. Hogg, and B. A. Huberman. Dynamics of computational ecosystems. *Phys. Rev. A*, 40(1):404–421, 1989.
8. C. Kipnis and C. M. Newman. The metastable behavior of infrequently observed, weakly random, one-dimensional diffusion processes. *SIAM J. Appl. Math.*, 45(6):972–982, 1985.
9. C. M. Newman, J. E. Cohen, and C. Kipnis. Neo-darwinian evolution implies punctuated equilibria. *Nature*, 315:400–401, May 30 1985.
10. A. Prügel-Bennett and Jonathan L. Shapiro. An analysis of genetic algorithms using statistical mechanics. *Phys. Rev. Lett.*, 72:1305–1309, 1994.
11. R. L. Scheaffer and J. T. McClave. *Probability and statistics for engineers*. PWS-KENT, 1990.
12. M. D. Vose. Punctuated equilibria in genetic search. *Complex Systems*, 5:31–44, 1991.
13. L. Darrell Whitley. Fundamental principles of deception in genetic search. In Gregory J. E. Rawlins, editor, *FOGA*. Morgan Kaufmann, 1991.

SGA Search Dynamics on Second Order Functions

Bart Naudts* and Alain Verschoren

Dept. of Mathematics and Computer Science
University of Antwerp, RUCA
Groenenborgerlaan 171, B-2020 Antwerpen, Belgium
e-mail: [bnaudts,aver]@ruca.ua.ac.be

Abstract. By comparing its search dynamics to that of a simple $\mathcal{O}(n^2)$ heuristic, we are able to analyze the behavior of the simple genetic algorithm on second order functions, whose optimization is shown to be an NP-equivalent problem. It appears that both algorithms approach these fitness functions in the same, intuitive way: they start by deciding the obvious and most probable, and then they proceed to make more difficult decisions. Useful information about the optimization problem is, among others, provided by statistical physics: lattice gases can be modeled as second order functions.

1 Introduction

In this note, we analyze the behavior of the simple genetic algorithm (SGA) on instances of a particular NP-equivalent optimization problem. (See [10] for a general introduction to complexity theory.) By choosing such a problem, we are ensured of instances of real world difficulty level, while still having a single and easily definable fitness function class to focus on.

A consequence of this choice is that we have to face the structural difficulties typical to NP-completeness. Firstly, the optimum of a function is something unreachable: exhaustive search is too expensive, and no efficient algorithm can tell a local optimum from a global one. A second problem is that it is very difficult to give a general characterization of difficult instances. E.g., recently a special volume [2] was dedicated to locating difficult instances of the 3SAT problem. Finally, one often has no idea of the characteristics of the fitness landscape, while still knowing that it contains many local optima which are fairly close to the global optimum. Only one path could lead to the global optimum, and there is no information to tell a good path from a bad one.

Throughout this note, we consider fitness functions of the form $f : \mathcal{S} \rightarrow \mathbb{Z}$, where $\mathcal{S}$ is the set of bit strings indexed by $\Lambda = \{0, 1, \dots, \ell-1\}$, i.e., $\mathcal{S} = \{0, 1\}^\Lambda$. Our very basic function class is the class of *first order functions*

$$f(s) = \sum_{i \in \Lambda} g_i(s_i), \quad g_i : \{0, 1\} \rightarrow \mathbb{Z}, \quad s \in \mathcal{S}, \quad (1)$$

* Research assistant of the Fund for Scientific Research – Flanders (F.W.O.) (Belgium)

also called *linear* or *fully separable* functions. Typical is that each locus contributes to the function value independently of the other loci, which makes these simple functions almost perfect for SGA optimization because the uniform crossover operator literally “exchanges properties”.

First order functions occur all over the place. The epistasis measure (e.g., [1],[6]) computes the least squares distance between a fitness function and the class of first order functions, whereas the bit decidability metric [6] is based on the concept of independent decidability of loci. Integer Programming (e.g., [11]) requires a first order function to be optimized, under the restriction of a number of linear relations. In Sect. 2.2, we see that the well known problem from statistical physics of finding the ground state of a lattice gas can be modeled into a fitness function maximization problem. There, the loci represent the sites of the lattice: each site can be occupied by a gas molecule or can be empty, or, alternatively, can be associated with a magnetic spin with either north (e.g., bit value 0) or south pole (e.g., bit value 1). In the latter case, the first order function represents an external magnetic field.

One adds second order components $g_{ij} : \{0,1\}^2 \rightarrow \mathbb{Z}$ to Eq. 1 to obtain the *second order functions*

$$f(s) = \sum_{i \in A} g_i(s_i) + \sum_{i < j \in A} g_{ij}(s_i, s_j). \quad (2)$$

The optimization problem associated with the second order functions will be denoted as 2ORD. It is a restriction of this class (hence a subproblem of 2ORD) which will be the subject of our analysis.

As we will see in the next section, a fair number of real world optimization problems can be embedded into 2ORD. This connection with extensively studied problems is important as a source of algorithms to compare with, a source of theoretic results which provides insight in the structure of the class, and, if nothing else, as a source of difficult instances. Two additional motivations for the choice of 2ORD are that it is probably the simplest yet very rich function class beyond the first order functions, and that there is no encoding from an abstract problem space to the space of bit strings involved. Also, note that Kauffman’s *NK*-landscapes [4] with $K = 1$ and $N = \ell$ can essentially be embedded in the substantially richer class 2ORD.

It has been proved in [7] that 2ORD is an NP-equivalent problem. In view of this complexity, one may wonder whether any metric based on first order functions can grasp the richness of the second order functions and render it using one single value. Simple as it is, though, the bit decidability measure provides valuable (but by no means exhaustive!) information on the GA-hardness of the instances.

The principal aim of this note is to show that the search dynamics of the SGA can be predicted for many functions in U-2ORD-01, an NP-equivalent subclass of 2ORD. This is done by comparing the dynamics of the SGA to those of a very intuitive $\mathcal{O}(\ell^2)$ heuristic, which tries to optimize U-2ORD-01 functions by deciding one locus at a time. One observes that the SGA and the heuristic

follow the same strategy, in particular, one finds a very high correlation between the order in which both algorithms decide the values of the loci. Moreover, the dynamics of the heuristic is a very good indicator of GA-hardness, the essential reason being the fact that it faces the same structural difficulty as the SGA does.

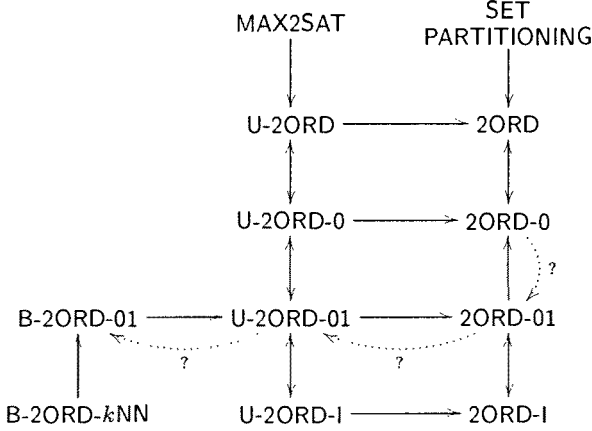


Fig. 1. Overview of inclusions and reductions. An arrow from one problem to another indicates a reduction or inclusion, e.g., U-2ORD can be reduced into U-2ORD-0, while U-2ORD-0 is trivially included in U-2ORD.

2 U-2ORD-01 and Relatives

Before sketching the big picture, let us first motivate the naming scheme for the function classes and associated optimization problems introduced in this note. The U- prefix stands for “unsigned”, e.g., U-2ORD is the subset of 2ORD instances which only have non-negative second order components $g_{ij}(\cdot, \cdot)$. The suffix -01 indicates that second order terms will only be included in the sum when $s_i = 0$ and $s_j = 1$; similarly the -0 suffix excludes all $g_{ij}(1, \cdot)$ terms. The -I suffix stands for “interaction”: terms are only included when $s_i \neq s_j$. More formally,

$$\text{2ORD-0: } f(s) = \sum_{i \in A} g_i(s_i) + \sum_{i < j \in A} g_{ij}(s_j) \cdot (1 - s_i), \quad (3)$$

$$\text{2ORD-01: } f(s) = \sum_{i \in A} g_i(s_i) + \sum_{i < j \in A} g_{ij} \cdot (1 - s_i)s_j, \quad (4)$$

$$\text{2ORD-I: } f(s) = \sum_{i \in A} g_i(s_i) + \sum_{i < j \in A} g_{ij} \cdot (s_i + s_j - 2s_i s_j). \quad (5)$$

Here, the g_i and g_{ij} take integer values. Note that none of the annotations affect the first order terms.

Via MAX2SAT it is proved in [7] that optimizing U-2ORD-01 functions is an NP-equivalent problem. Figure 1 shows the necessary reductions, as well as some other problems and reductions: from SET PARTITIONING into 2ORD, from U-2ORD-I into U-2ORD-01, etc. Even two processor network balancing can be modeled into U-2ORD, although the arbitrary processor version is much more interesting. See [8] for more details.

The following two subsections complete the overview of optimization problems and function classes related to 2ORD.

2.1 Subclasses of U-2ORD-01

The B- (for *binary*) prefix indicates that the second order components g_{ij} take values in $\{0, 1\}$, while the first order components keep their larger range. The fact that a number of very difficult U-2ORD-I instances can be written as B-2ORD-01 instances (as in Sect. 4), leads us to conjecture that B-2ORD-01 is NP-equivalent.

The B-2ORD-2NN class consists of those functions of B-2ORD-01 where $j-i > 2$ implies that $g_{ij} = 0$. Hence its name: in the worst case a locus is dependent on its two *nearest neighbors*. Instances of B-2ORD-2NN can be optimized polynomially by a simple *divide and conquer* algorithm. In general, if one restricts the span of the edges to a constant distance $k \geq 1$, then the resulting optimization problem U-2ORD- k NN remains polynomial.

2.2 Interaction models

The most interesting instances of interacting particle systems [5] can be modeled in the class U-2ORD-I, where a component g_{ij} contributes to the function value when s_i and s_j have opposite values. Finding the ground state of an anti-ferromagnetic model then accords to maximizing a particular fitness function.

Example. Let us consider the anti-ferromagnetic one dimensional Ising model (see [12], e.g.) with cyclic boundary conditions. The model is represented by an instance of U-2ORD-I; for $\ell = 6$ we get

$$f(s) = (s_0 + s_1 - 2s_0s_1) + (s_1 + s_2 - 2s_1s_2) + (s_2 + s_3 - 2s_2s_3) + \\ (s_3 + s_4 - 2s_3s_4) + (s_4 + s_5 - 2s_4s_5) + (s_5 + s_0 - 2s_5s_0). \quad (6)$$

This fitness function can be visualized by

$$*0 \longleftrightarrow *1 \longleftrightarrow *2 \longleftrightarrow *3 \longleftrightarrow *4 \longleftrightarrow *5 \quad (7)$$

Each vertex in this picture represents a locus; the label of the vertex indicates the locus index. A bidirectional edge between vertices i and j indicates that $g_{ij} = 1$; components not represented by an edge in the picture have a value of zero.

The asterisk ‘*’ in each label is a place-holder for a bit value. While the picture in Eq. 7 represents f , the following picture represents f together with the string 010101, which happens to be one of the two global optima:

$$0_0 \xleftrightarrow{\quad} 1_1 \xleftrightarrow{\quad} 0_2 \xleftrightarrow{\quad} 1_3 \xleftrightarrow{\quad} 0_4 \xleftrightarrow{\quad} 1_5 \quad (8)$$

By changing their first order component, we can reduce U-2ORD-I instances into U-2ORD-01 instances, which brings them within the domain of the heuristic. A detailed proof of this fact may be found in [7]; the following example illustrates the main technique:

Example. Let $\Lambda = \{0, 1\}$ and consider the simple U-2ORD-I instance

$$f(s) = a(s_0 + s_1 - 2s_0s_1), \quad (9)$$

with $a \in \mathbb{Z}$. We may rewrite this as

$$f(s) + a = as_0 + a(1 - s_1) + 2a(1 - s_0)s_1, \quad (10)$$

to recognize in the right hand side an instance of U-2ORD-01. As the constant term a in the left hand side can be ignored in the context of function optimization, we have performed a correct reduction; it is visualized by

$$*0 \xleftrightarrow{v} *1 \quad \Longrightarrow \quad \begin{array}{ccc} & & v \\ & & \nearrow \\ *0 & \xrightarrow{2v} & *1 \\ \nwarrow & & \\ & v & \end{array} \quad (11)$$

The picture also shows three more features of our system of visualizing second order functions. Firstly, when visualizing 2ORD-01 instances, we use unidirectional edges. Secondly, if an edge is labeled, it represents a second order component g_{ij} with the value of that label, instead of the value 1. Lastly, the first order components are visualized by edges that do not connect but point to resp. from a vertex, indicating non-zero $g_i(1)$ resp. $g_i(0)$ values.

Another interesting example is the two dimensional anti-ferromagnetic Ising model on a square lattice. Figure 2 describes it as an instance of U-2ORD-I and of U-2ORD-01.

The well established field of statistical physics gives a number of indications why finding ground states can be difficult. We conclude this section by putting three of them, *dimensionality*, *symmetry* and *frustration*, in the context of second order function optimization.

Dimensionality refers to the degree of connectedness of the edges of a function visualization. We have already come across an optimization problem in which one dimensional interaction models can be embedded: U-2ORD- k NN. Here, the span of the edges is restricted to the constant k . In statistical physics, one dimensional models can be dealt with analytically; we see that the corresponding

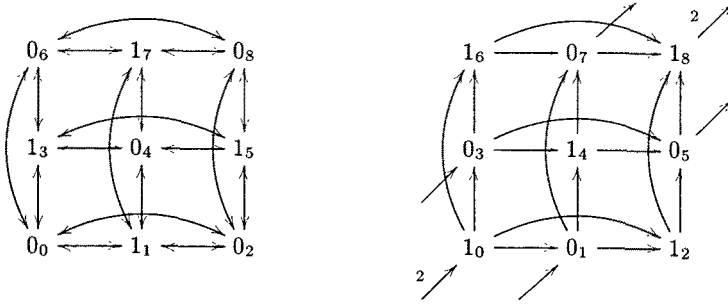


Fig. 2. The $d=2$ anti-ferromagnetic Ising model on a square lattice, with 3^2 sites, as an instance of U-2ORD-I and U-2ORD-01. This model has two ground states, i.e., two global optima: the two checkerboard patterns shown together with the model.

optimization problem is polynomial. It is visually clear that the model of Eq. 6 is one dimensional and that that of Fig. 2 is two dimensional. Interaction models become easier with the dimensionality increasing, the two dimensional models being the hardest. We will observe this behavior in Sect. 5, when we consider U-2ORD- k NN instances for k near ℓ .

It is easy to see that the Ising models are highly symmetrical, which is a potential source of hardness, mainly because sequential optimization algorithms do not have a clue of where to start the optimization process. The effects of symmetry are most easily seen in conjunction with the third hardness indicator, frustration.

Frustration occurs, e.g., in the model visualized in Fig. 3 below. The figure shows a two dimensional anti-ferromagnetic Ising model, this time not on a square but on a triangular lattice. It is beyond the scope of this note to formally define the notion of frustration, but we can easily illustrate it as follows.

The two ground states of the Ising model on a square lattice are checkerboard patterns, i.e., for each square in the model, one finds optimal solutions

$$\begin{array}{cc}
 \begin{array}{ccc}
 0_2 & \longleftrightarrow & 1_3 \\
 \updownarrow & & \updownarrow \\
 1_0 & \longleftrightarrow & 0_1
 \end{array} &
 \begin{array}{ccc}
 1_2 & \longleftrightarrow & 0_3 \\
 \updownarrow & & \updownarrow \\
 0_0 & \longleftrightarrow & 1_1
 \end{array}
 \end{array} \tag{12}$$

A sequential algorithm would start optimizing, e.g., in the lower left corner, by choosing a value of 0 for locus 0. Then it would decide to put 1's at loci 1 and 2, to make sure that it gets the components g_{01} and g_{02} in the fitness sum. The decisions made by this sequential algorithm turn out to be successful, since by putting a 0 at locus 3, the algorithm has found the global optimum.

The symmetry of the square lattice allows the algorithm to start at any point, and reach one of the two global optima by proceeding in a greedy or hill climbing sort of way. We say that all paths lead to the global optimum.

This equilibrium can easily be disturbed by adding a diagonal component, to obtain the repeated subproblem of the Ising model on a triangular lattice

$$\begin{array}{ccc}
 0_1 & \longleftrightarrow & ?_2 \\
 \updownarrow & \nearrow & \\
 1_0 & &
 \end{array} \tag{13}$$

Our sequential algorithm now faces a problem: after setting the value of 1 at locus 0, the combinations (0,0), (0,1), and (1,0) for loci 1 and 2 all result in the same fitness value. But only one of these choices might be on the path leading to the global optimum if this triangle is only a small subproblem.

Example 2 of Sect. 4 also exhibits this typical form of frustration.

3 The Heuristic

The SGA, with uniform crossover and ordinary mutation, does not seem to have great problems with optimizing randomly sampled U-2ORD-01 functions. But, neither does the simple hill climbing heuristic described below.

Algorithm 1 (Heuristic for U-2ORD-01). Let $f \in \text{U-2ORD-01}$ be the fitness function. This heuristic tries to optimize f by deciding the value of one locus at a time. Initially all loci are “free”; this is formalized by using a control set F , initialized as $F = A$. The heuristic will stop when there are no free loci anymore, i.e., when $F = \emptyset$. A locus i which is not free anymore is called “fixed”; it is fixed to the value $h_i \in \{0, 1\}$. The outcome of the heuristic is the string h with values h_i .

Thus the set F registers the evolution of the algorithm. Each of the free loci is given a score, and the best one will be chosen to be fixed. The score of a locus is based on a study of its incoming and outgoing edges. In what follows, we will define the sets of incoming and outgoing edges of a free locus, and divide them into three subsets, without continually referring to the dependency on F .

Let T_0^i be the set of labeled edges which start in a free locus $i \in F$, and T_1^j the set of labeled edges which end in a free locus $j \in F$. More formally,

$$\begin{aligned}
 T_0^i &= \{j > i; g_{ij} > 0\}, & \text{for } i \in F, \\
 T_1^j &= \{i < j; g_{ij} > 0\}, & \text{for } j \in F.
 \end{aligned} \tag{14}$$

Each set of edges T_b^k is then divided into three groups: Let

$$\begin{aligned}
 L_0^i &= \{j > i; g_{ij} > 0, j \notin F \text{ and } h_j = 0\}, \\
 L_1^j &= \{i < j; g_{ij} > 0, i \notin F \text{ and } h_i = 1\},
 \end{aligned} \tag{15}$$

which might be called the “lost edges” of i resp. j . On the other hand,

$$\begin{aligned} C_0^i &= \{j > i; g_{ij} > 0, j \notin F \text{ and } h_j = 1\}, \\ C_1^j &= \{i < j; g_{ij} > 0, i \notin F \text{ and } h_i = 0\} \end{aligned} \quad (16)$$

are the “promising edges”. Obviously $C_b^k \cap L_b^k = \emptyset$ for all values b , free loci $k \in F$ and states of F . Finally we define the “free edges” of $k \in F$, for each b , as

$$F_b^k = T_b^k \setminus \{C_b^k \cup L_b^k\}. \quad (17)$$

Now if one sets, for each free locus $i \in F$,

$$\begin{aligned} \delta_0^i &= 2g_i(0) + 2 \sum_{i < j \in C_0^i} g_{ij} + \sum_{i < j \in F_0^i} g_{ij}, \\ \delta_1^i &= 2g_i(1) + 2 \sum_{i > j \in C_1^i} g_{ji} + \sum_{i > j \in F_1^i} g_{ji}, \end{aligned} \quad (18)$$

and $\Delta_i = |\delta_0^i - \delta_1^i|$, then the heuristic picks one of the loci with maximal Δ as the next locus to be decided, i.e., it picks an $n \in F$ such that

$$\Delta_n = \max_{j \in F} \Delta_j. \quad (19)$$

The algorithm proceeds by determining the value of h_n , according to the sign of $\delta_0^n - \delta_1^n$: put $h_n = b$ if $\delta_b^n - \delta_{1-b}^n > 0$. One additional rule comes in to decide the value if the difference of the δ 's is 0: if $n < \ell/2$ then $h_n = 0$, otherwise $h_n = 1$. The algorithm moves on to the next state, by setting $F \leftarrow F \setminus \{n\}$. $\square$

The heuristic contains three parameters. First there is the factor with which the number of promising edges and first order components in the calculation of δ_b^i is multiplied. A value of 2 is chosen because intuitively one sees that the odds of changing a free edge into a lost edge could equal those of changing it into a promising one. A second “parameter” is the rule for picking a locus when there is more than one optimal Δ_i . In practice random selection is used. The third “parameter” is the additional rule for deciding the value h_n when $\Delta_n = 0$. Again, one sees that, at least for randomly sampled instances, loci 0 up to, but not including, $\ell/2$ are best set to 0, and the others to 1.

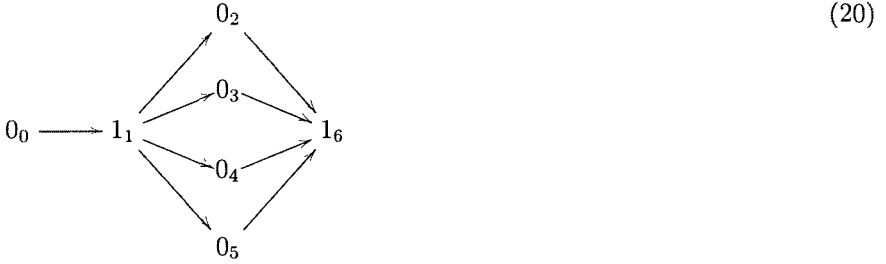
The multiplication factor 2 is used both for the promising edges and the first order component. This leads to another way of viewing the workings of the heuristic. When the value of a locus is decided, the locus is removed and its influence, in the form of promising and lost edges, is transferred to the first order component of the free loci. In other words, the search problem is reduced to a problem with fewer loci but with a modified first order component.

Although one observes that the heuristic optimizes a great number of randomly sampled U-2ORD-01 functions, it certainly cannot do all of them, as we will now show.

4 Where the Heuristic Gets Lost

Example 1 shows a B-2ORD-01 instance which completely misleads the heuristic. It shows that one cannot give a bound on the error of the heuristic. As a matter of convenience, we will add a super-index to the symbol Δ_i used in the algorithm to indicate the number of loci which have been fixed so far.

Example 1. Let $A = \{0, \dots, 6\}$, and let us visually define a B-2ORD-01 instance, together with its optimum:

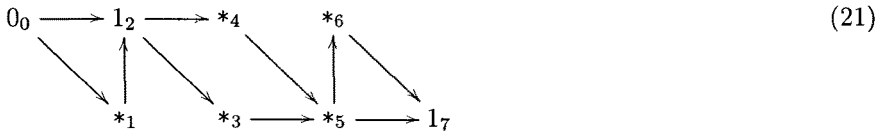


Initially, all loci and edges are free, and the loci have Δ_i^0 values of 1, 3, 0, 0, 0, 0 and 4 respectively. Hence locus 6 gets fixed to 1 ($h_6 = 1$). The loci 2–5 now have one free and one promising edge, which results in $\Delta_2^1 = \dots = \Delta_5^1 = 1$, while we still have $\Delta_0^1 = 1$ and $\Delta_1^1 = 3$. Hence the next step is to fix locus 1 to 0, which leads away from the optimum, because the component g_{01} will not be included in the fitness sum, and the components g_{26} , g_{36} , g_{46} and g_{56} would have been included anyway.

In practice the heuristic starts by fixing the values of all loci which have either only incoming or only outgoing edges to 1 resp. 0. Even then the heuristic is lead away from the optimum: after fixing loci 0 and 6, Δ_1^2 is still greater than Δ_2^2 , Δ_3^2 , Δ_4^2 and Δ_5^2 .

Another typical way of misleading the heuristic is to force it to decide between a number of loci which all have $\Delta = 0$. If then only one “path”, or correct choice, leads to the global optimum, there is no information of where to go, and any search algorithm is forced into exhaustive search.

Example 2. Consider the following frustrated B-2ORD-2NN instance, together with the outcome of the algorithm after three steps:



Initially, the Δ_i^0 values are 2, 0, 0, 0, 0, 0, 0, and 2 respectively. Let us say that the algorithm fixes locus 0 to 0. Now both locus 1 and locus 2 have one promising

edge, which results in $\Delta_1^1 = 1$, $\Delta_2^1 = 1$ and the other Δ_i^1 's remaining unchanged. Locus 7 is fixed to 1, and we obtain $\Delta_1^2 = \Delta_2^2 = \Delta_5^2 = \Delta_6^2 = 1$, and $\Delta_3^2 = \Delta_4^2 = 0$. Assume that locus 2 gets fixed to 1, then $\Delta_1^3 = 0$ and $\Delta_3^3 = \Delta_4^3 = \Delta_5^3 = \Delta_6^3 = 1$. Locus 1 is not important anymore.

If the algorithm now makes the mistake of fixing locus 5 to value 0, then it is lead away from the optimum. Alternatively, assume that locus 4 is fixed to 0. Then $\Delta_3^4 = \Delta_6^4 = 1$ and $\Delta_5^4 = 0$, and the algorithm is lead to the optimum, regardless of further choices.

In this context the Ising models of Figs. 2 and 3 prove to be nice examples. Already in the first step, the heuristic is faced with zero Δ_i 's for all loci! In the square model, all paths lead to the optimum (one can see the fitness landscape as a hyper-paraboloid), whereas in the triangular model, a large number of paths mislead the heuristic.

The case where all loci initially have a Δ_i of 0 shows how symmetry can obstruct the search of the heuristic. This is the reason why the heuristic is fairly good at optimizing random samples: they don't contain too much symmetry. Random samples typically also have a very high degree of connectedness, which indicates a high dimension.

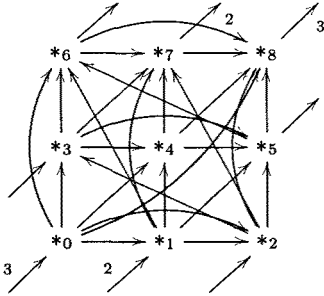


Fig. 3. The $d=2$ anti-ferromagnetic Ising model on a triangular lattice, with 3^2 sites, as an instance U-2ORD-01.

5 The Heuristic and the Simple Genetic Algorithm

Before we can start comparing the workings of the heuristic and the SGA, we have to decide on a stopping criterion for the latter, bearing in mind that we cannot know the maximum of our fitness functions at all. A natural choice is to stop the SGA when its population is dominated by a string with fitness value at least the value provided by the heuristic. This also allows for an efficiency comparison between the two algorithms.

5.1 The SGA's Fixing Chronology

Given the evolution $(\mathcal{P}_t)_{t=0}^T$ of an initial population $\mathcal{P}_0$ of the SGA, we can extract *whether* and *when* a certain locus got fixed to a final position if we assume that when the SGA stops at time T , a certain string has taken over a large proportion of the population. Let us call this string m , and define, for each locus i , the set

$$A_i = \left\{ 0 \leq t \leq T; \forall t \leq u \leq T. \frac{|\{s \in \mathcal{P}_u; s_i = m_i\}|}{|\mathcal{P}_u|} \geq \frac{7}{8} \right\}. \quad (22)$$

Obviously the value $7/8$ is a parameter we can freely choose. It seems a good idea to set it well beyond $1/2$.

We say that locus i got fixed if $A_i \neq \emptyset$, in which case its fixing time $I_{\text{SGA}}(i)$ is defined to be the earliest evolution step found in A_i , i.e., $I_{\text{SGA}}(i) = \min A_i$. If $A_i = \emptyset$, the locus did not get fixed; by convention we set its fixing time to the last step in the evolution of the population: $I_{\text{SGA}}(i) = T$.

Consider this example of a fixing chronology (short for ‘set of fixing times’):

$$5 \quad 29 \quad 28 \quad 5 \quad 5 \quad 35 \quad 6 \quad 9 \quad 23 \quad 10 \quad 10 \quad 35, \quad (23)$$

produced by an SGA with $\ell = 12$ which ran for 35 iterations. It shows that sometimes the SGA fixes a number of loci simultaneously, as it processes the loci in parallel. We also see that there can be a significant number of iterations between two fixing times.

5.2 The Heuristic's Fixing Chronology

Although the heuristic is sequential in nature, and its fixing times can be defined in a trivial way, it can also be thought of as processing a number of loci in parallel, namely those which have equal Δ_i values.

To take this effect into account, we modify the way the heuristic's chronology is computed.

Algorithm 2 (Fixing Times of the Heuristic). Initially, each locus i 's fixing time $I_H(i)$ is undefined, and a counter k is set to 0. Every time a locus is removed from F , the counter k is changed and a number of fixing times are set, in the following way.

At stage F , let Γ_F be the set of loci i which have an optimal Δ_i , and out of which one locus γ is picked to be fixed. If $I_H(\gamma)$ has already received a value, then k is set to $I_H(\gamma) + 1$. Next, the fixing times $I_H(\lambda)$ are set to k for all $\lambda \in \Gamma_F$ with undefined fixing time. Finally, k is incremented by 1. $\square$

As an example, let us apply this algorithm to obtain the fixing chronology associated with the heuristic's evolution of Example 2 in Sect. 4.

Example. At the initial stage $F = A$, two loci have an optimal Δ_i : $\Delta_0^0 = 2$ and $\Delta_7^0 = 2$, so $\Gamma_A = \{0, 7\}$. Locus 0 is chosen and since the fixing times of loci 0 and 7 are still undefined, they are set to 0, and the counter k is incremented from 0 to 1. After the first locus has been fixed, we see that $\Delta_7^1 = 2$ is still the greatest of the Δ_i^1 , resulting in $\Gamma_{A \setminus \{0\}} = \{7\}$. Since $I_H(7)$ has already been set, we set k to $I_H(7) + 1 = 1$, and increment it once more to obtain $k = 2$. In the third step, $\Gamma_{A \setminus \{0, 7\}} = \{1, 2, 5, 6\}$, and since their fixing times are still undefined, we set $I_{H(1)} = I_{H(2)} = I_{H(5)} = I_{H(6)} = k = 2$, and we increment k by one.

If we assume that the heuristic proceeded by fixing loci 4, 6, 5, 3 and 1, we obtain the fixing chronology 0, 2, 2, 3, 3, 2, 2, 1.

5.3 Results

Let us start by giving an overview of all parameters involved. The SGA, with uniform crossover (rate 0.7), and ordinary bit-flip mutation, (rate 0.003), ranking selection and a population size of 128 strings, reaches the heuristic's value if, within 1000 iterations, a string with at least this value takes up at least 50 percent of the population. Each (average, standard deviation) pair in Tables 1 and 2 is the result of averaging the observed correlation over 20 instances. The correlations are obtained by averaging both the heuristic's and the SGA's chronologies over 10 runs.

Randomly sampled B-2ORD-01 instances are defined by letting $g_{ij} = 1$ with probability p . Table 1 shows observed averages and standard deviations of the correlation between the chronologies for different p values. We also observed that the SGA almost always reached the value found by the heuristic, that it usually needed less than 100 iterations, and that different runs of the heuristic almost always produced the same chronology.

To obtain Table 2, we set the range of the U-2ORD- k NN components to $\{0, \dots, 15\}$. The correlation values, the percentage of successful SGA runs and the bit decidability values all indicate that the lower the span of the edges, hence the lower the degree of connectedness and the flatter the landscape, the harder the functions are for the SGA.

Finally, it is of interest to compare the heuristic and the SGA on the highly symmetrical models from statistical physics. As has been announced, the triangular model is a lot harder to optimize than the square model: not even once out of 50 runs, the SGA could reach the heuristic's value, whereas the square model resulted in 28 successful runs out of 50. Moreover, all heuristic runs on the square model resulted in the same optimum, whereas only 33 out of 50 did so for the triangular model. Computing the correlation is not all that useful for these two models, since we know that all Δ 's are initially 0. However, we observe relatively high correlations, simply because the fact that each different SGA run produces a different chronology, which, in the end, averages to a flat chronology. Both models share the worst value $\tau = 1$.

Table 1. Average and standard deviation of the correlation between the chronologies of heuristic and SGA, for randomly sampled B-2ORD-01 instances. The parameter p gives the probability of g_{ij} being 1.

p	$ A = 32$		$ A = 80$	
	avg	std dev	avg	std dev
0.1	0.74	0.06	0.67	0.11
0.15	0.76	0.10	0.69	0.12
0.2	0.83	0.09	0.80	0.12
0.3	0.86	0.09	0.83	0.07
0.4	0.85	0.09	0.85	0.09
0.5	0.87	0.12	0.84	0.11
0.6	0.91	0.08	0.88	0.03
0.7	0.90	0.08	0.90	0.03
0.8	0.92	0.06	0.87	0.08
0.9	0.93	0.02	0.91	0.02
0.95	0.93	0.02	0.90	0.02
0.99	0.93	0.02	0.90	0.02
1.0	0.94	0.01	0.91	0.01

Table 2. Average and standard deviation of the correlation between the chronologies of heuristic and SGA, for randomly sampled U-2ORD- k NN instances with components uniformly drawn from the range $\{0, \dots, 15\}$. We also give the percentage of SGA runs which reached the heuristic's value, and the percentage of runs amongst them with higher value than the heuristic's. Finally we stress the strong correlation with the bit decidability metric τ , introduced in [6].

k	avg	std dev	pct. success	pct. better	τ
2	0.38	0.15	32	52	0.87
3	0.41	0.12	20	59	0.85
5	0.46	0.12	27	57	0.82
7	0.54	0.14	29	72	0.79
10	0.51	0.15	31	65	0.75
15	0.58	0.13	50	70	0.67
20	0.69	0.07	57	61	0.59
25	0.68	0.16	59	45	0.51
40	0.73	0.10	62	37	0.26
50	0.79	0.07	96	4	0.15
60	0.84	0.03	96	0	0.15
70	0.84	0.10	93	4	0.15
79	0.90	0.03	98	0	0.14

5.4 Heuristic versus SGA

Which algorithm is best? It is important to note that we stop the SGA when it reaches the heuristic's value, while it might have found better values after some more iterations. Even if we take this into account, we are lead to believe that the heuristic is a lot more efficient in providing us with a decent approximation of the optimum, at least on a majority of reasonable instances. But, we have no guarantee at all that the SGA will always do better than the heuristic on the more difficult instances.

6 Conclusion

Having observed the (sometimes very high) correlation between the chronologies of the heuristic and the SGA, we have a means of explaining and predicting the search dynamics of the SGA. More precisely, for a large class of U-2ORD-01 instances we can algorithmically predict *how* the SGA proceeds from a random population to one containing the first local optimum it comes across. Often, this local optimum is the one found by the heuristic.

While other researchers often consider fitness functions with high or varying epistatic interaction, we focus on second order interaction. We prove that the related optimization problem 2ORD is NP-equivalent, hence certainly rich enough for general analysis. Moreover, the connection with statistical physics, a very well established field, and with some other search problems, proves to be a very valuable resource.

From the theoretical point of view, the next step now is to formalize the correspondence between the two algorithms. This might be achieved by constructing a chain of intermediate algorithms, closely related to each other. E.g., one might be able to prove the correspondence between the heuristic and a more probabilistic version of it, which adheres closer to the SGA. Or, one might study the correspondence between the correlation values and the bit decidability values. It is also interesting to study the way the SGA parameters, like crossover and mutation rate, selection scheme, population size etc., influence the chronologies produced by the algorithm.

On the practical side, it would be instructive to know if the dashed arrows in Fig. 1 can be proved or disproved, maybe allowing for some other real world problems to be reduced to the domain of the heuristic.

Acknowledgements

The authors wish to thank J. Naudts (University of Antwerp, Dept. of Physics) for pointing out the connection with statistical physics, and all fruitful discussions which followed after that. They are also grateful for the comments of the two anonymous referees.

References

1. Davidor, Y. (1991): "Epistasis Variance: A Viewpoint on GA-Hardness". In: G.J.E. Rawlins ed., *Foundations of Genetic Algorithms*, Morgan Kaufmann Publishers, San Francisco, Ca., pp. 23-35.
2. Hogg, T., Hubermann, B., Williams, C., eds. (1996): "Special Volume on Frontiers in Problem Solving: Phase Transitions and Complexity". *Artificial Intelligence*, vol. 82.
3. Jones, T., Forrest, S. (1995): "Fitness Distance Correlation as a Measure of Problem Difficulty for Genetic Algorithms". In: L.J. Eshelman ed., *Proceedings of the Sixth International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, San Francisco, Ca., pp. 184-192.
4. Kauffman, S.A. (1989): "Adaptation on Rugged Fitness Landscapes". In: D.L. Stein, ed., *Lectures in the Sciences of Complexity*, Addison-Wesley, Reading, Ma., pp. 527-618.
5. Liggett, T.M. (1985): "Interacting Particle Systems". Springer Verlag, Heidelberg, Germany.
6. Naudts, B., Suys, D., Verschoren, A. (1997): "Epistasis as a Basic Concept in Formal Landscape Analysis". In: Th. Baeck, ed., *Proceedings of the 7th International Conference on Genetic Algorithms*, Morgan Kaufmann, San Francisco, Ca., pp. 65-72.
7. Naudts, B., Verschoren, A. (1997): "Second Order Functions for the SGA". In preparation.
8. Naudts, B. (1998): "Network Balancing: SGA versus Hillclimber". In preparation.
9. Naudts, B. (1998): "A Theoretical Approach to GA-Hardness". Ph.D. thesis, University of Antwerp, RUCA, Belgium. In preparation.
10. Papadimitriou, C.H. (1993): "Computational Complexity". Addison-Wesley, Reading, Ma.
11. Salkin, H.M. (1975): "Integer Programming". Addison-Wesley, Reading, Ma.
12. Thompson, C.J. (1972): "Mathematical Statistical Mechanics". Macmillan, New York.

Asymptotical Convergence Rates of Simple Evolutionary Algorithms under Factorizing Mutation Distributions

Günter Rudolph

Universität Dortmund, Fachbereich Informatik, LS XI, D-44221 Dortmund

Abstract. The standard choice for mutating an individual of an evolutionary algorithm with continuous variables is the normal distribution. It is shown that there is a broad class of alternative mutation distributions offering local convergence rates being asymptotical equal to the convergence rates achieved with normally distributed mutations. Such mutation distributions must be factorizing and the absolute fourth moments must be finite. Under these conditions an asymptotical theory of the convergence rates of simple evolutionary algorithms can be established for the entire class of distributions.

1 Introduction

The standard choice to represent mutations in evolutionary models dealing with continuous quantities is the normal distribution. This choice is usually justified by the central limit theorem: Since mutations in nature are caused by a variety of physical and chemical influences that are not identifiable or measurable to a degree that allows for a deterministic model, these influences are considered as independent random perturbations whose normed sum approaches a normal random variable in the limit, provided that the first two absolute moments of the distributions of these random perturbations are finite and that the so-called Lindeberg condition is obeyed. Therefore it is not surprising that evolutionary algorithms with continuous search space model mutations by normally distributed random variables as well. But the biological original needs not necessarily be the best choice when mutations play the role of an exploration operator—as it is the case in evolutionary algorithms (EAs). It was noted several times [1, 2, 3] that mutation distributions with slowly (i.e., not exponentially) decreasing tails should offer a larger probability to escape from local optima (also see fig. 1 & 2). Although this claim is certainly correct if the variance is held fixed, it is still an open question whether this theoretical property carries over to practical EAs employing an auto-adaptive adjustment of the variances. But this question will not be addressed here. Instead, it is investigated to which extent non-normal mutation distributions may affect the local convergence behavior of evolutionary algorithms.

Two simple evolutionary algorithms will be studied here: The $(1+1)$ -EA and the $(1, \lambda)$ -EA. The first one generates a single offspring by mutation and accepts the offspring only if it is better than the parent, whereas the latter one generates

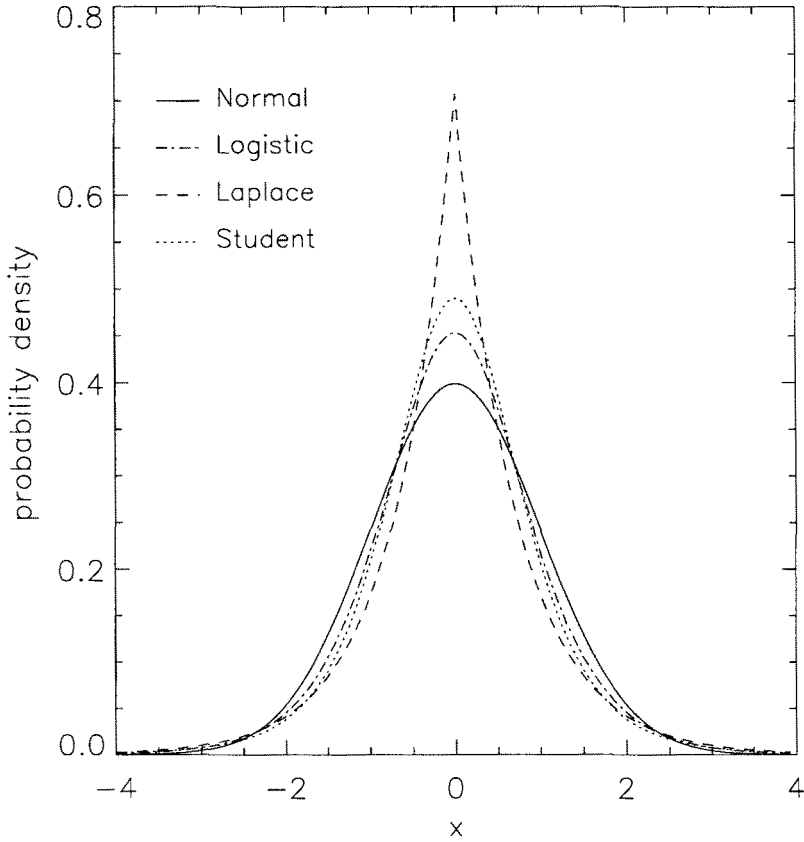


Fig. 1. The shape of some symmetrical univariate distributions with zero mean and unit variance.

distribution	density function
Normal	$\exp(-x^2/2)/\sqrt{2\pi}$
Logistic	$\pi \operatorname{sech}^2(\pi x/\sqrt{12})/\sqrt{48}$
Laplace	$\exp(- x \sqrt{2})/\sqrt{2}$
Student (d=5)	$8(1+x^2/3)^{-3}/\sqrt{27\pi^2}$

Table 1. Probability density functions of some symmetrical univariate distributions with zero mean and unit variance.

$\lambda \geq 2$ offspring independently with the same mutation distribution and chooses the best offspring among the λ offspring to serve as new parent (regardless of

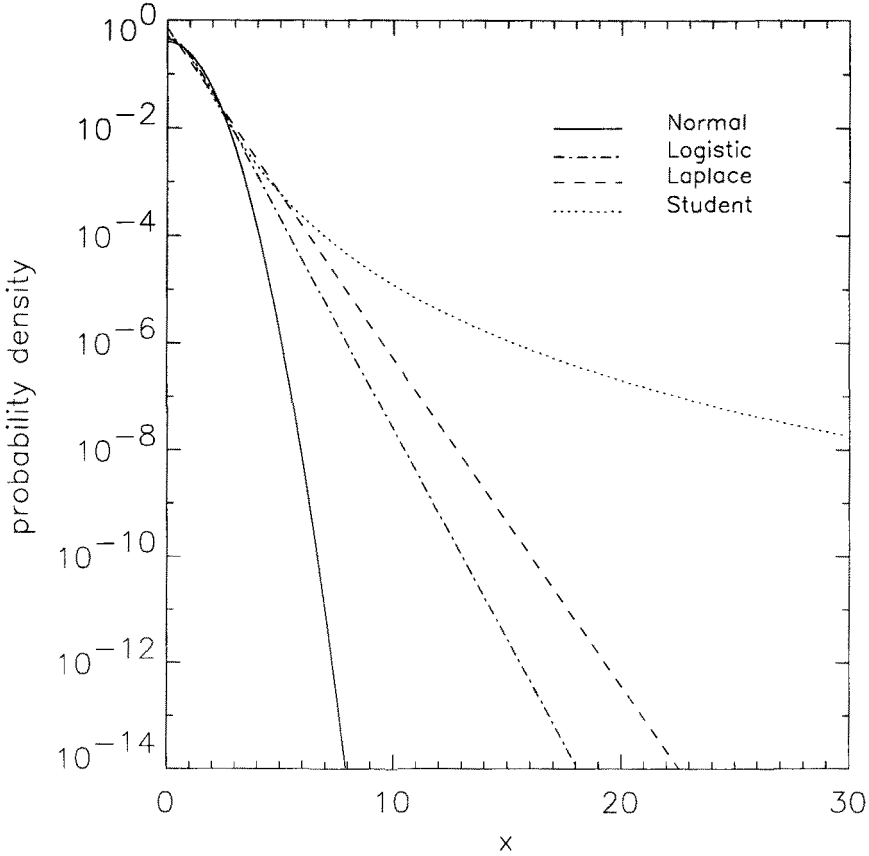


Fig. 2. The decay of the right tails of the symmetric Normal, Logistic, Laplace, and Student distribution. The tails of the first three distributions decline exponentially whereas the tail of the Student distribution (with 5 degrees of freedom) follows a power law.

the quality of the old parent). If $\theta \in \mathbb{R}^n$ denotes the current position of the EA in the search space, then a mutation is modeled by adding a random vector $\mathbf{Z}$ that must fulfill some conditions (details will follow shortly). Thus, an offspring $\mathbf{X}$ is represented by the random variable $\mathbf{X} = \theta + \mathbf{Z}$.

The test problem is the minimization of the objective function $f(\mathbf{x}) = \mathbf{x}'\mathbf{x}$ with $\mathbf{x} \in \mathbb{R}^n$. It will be assumed that n is large ($n \geq 100$). This test function reflects to some extent the case of a local optimum, and it is usually used to assess the local convergence behavior of evolutionary algorithms. To be comparable to previous work, this common practice is followed here.

2 Asymptotical Results

The *fundamental assumption* made in the remainder is that it will be postulated that the product moments of random vector $\mathbf{Z}$ do exist up to order 4. Further conditions on $\mathbf{Z}$ are given in the definition below which specifies the distribution class of random vector $\mathbf{Z}$.

Definition 1. The distribution of random vector $\mathbf{Z}$ is termed a *mutation distribution* if $E[\mathbf{Z}] = \mathbf{0}$. In this case, random vector $\mathbf{Z}$ is called a *mutation vector*. A mutation distribution is said to be *factorizing* if the joint probability density function of the mutation vector $\mathbf{Z}$ can be written as

$$f_{\mathbf{Z}}(z_1, \dots, z_n) = \prod_{i=1}^n f_{Z_i}(z_i)$$

with $f_{Z_1}(\cdot) = \dots = f_{Z_n}(\cdot)$ where n denotes the dimension. $\square$

Let $\mathbf{Z}$ possess a factorizing mutation distribution. Since the random objective function value of an offspring is given by

$$f(\theta + \mathbf{Z}) = \sum_{i=1}^n (\theta_i + Z_i)^2$$

each of the summands above is mutually independent to the remaining ones. As a consequence, the objective function value is representable by a sum of independent random variables. If such a sum is appropriately normed, then its distribution converges to some limit distribution as $n \rightarrow \infty$. This fact will be exploited to develop an asymptotical theory with regard to the convergence rates. In order to obtain the desired norming constants some preparatory results are necessary.

Lemma 2. Let Z be a symmetrical random variable with $E[Z^{2k-1}] = 0$ for $k \in \mathbb{N}$ and set $X = \theta + Z$ with $\theta \in \mathbb{R}$. Then $E[X^2] = \theta^2 + E[Z^2]$ and $V[X^2] = 4\theta^2 E[Z^2] + V[Z^2]$. $\square$

The proof of this lemma is trivial and therefore omitted while the next result is an immediate consequence of the lemma above.

Proposition 3. Let $Z_1, \dots, Z_n$ be independent and identically distributed symmetrical random variables with $E[Z_i^{2k-1}] = 0$ for $i = 1, \dots, n$ and $k \in \mathbb{N}$. If $X_i = \theta_i + Z_i$ and $S_n = \sum_{i=1}^n X_i^2$ then

$$\begin{aligned} E[S_n] &= \|\theta\|^2 + n E[Z_1^2] \\ V[S_n] &= 4 \|\theta\|^2 E[Z_1^2] + n V[Z_1^2] \end{aligned}$$

where $\theta \in \mathbb{R}^n$ and $\|\cdot\|$ denotes the Euclidean norm. $\square$

The central limit theorem (see [4], p. 262) ensures that the distribution of the appropriately normed random scalar product $S_n = \mathbf{X}'\mathbf{X} = \|\mathbf{X}\|^2$ converges weakly to the standard normal distribution. Thus, since

$$\frac{S_n - E[S_n]}{V[S_n]^{1/2}} \rightarrow N \sim N(0, 1)$$

as $n \rightarrow \infty$ one obtains

$$\begin{aligned} S_n &\approx E[S_n] + V[S_n]^{1/2} \cdot N \\ &= \|\theta\|^2 + n E[Z^2] + (4\|\theta\|^2 E[Z^2] + n V[Z^2])^{1/2} \cdot N. \end{aligned} \quad (1)$$

Let $\eta^2 = V[Z] = E[Z^2]$ and suppose that $V[Z^2] = a\eta^4 = aV[Z]^2$ for some $a > 0$. Then the random variable $S_n/\|\theta\|^2$ can be written as

$$\frac{S_n}{\|\theta\|^2} \approx 1 + \frac{1}{n} \left[\gamma^2 + \left(4\gamma^2 + \frac{a\gamma^4}{n} \right)^{1/2} \cdot N \right] \quad (2)$$

where $\gamma = n\eta/\|\theta\|$. After having established this approximation one can begin to calculate the expected asymptotical progress rates for the $(1+1)$ -EA and the $(1, \lambda)$ -EA provided that the objective function is $f(\mathbf{x}) = \|\mathbf{x}\|^2$. At first consider the $(1+1)$ -EA. Assume the current position is $\theta \in \mathbb{R}^n$. Since the $(1+1)$ -EA only accepts improvements the relative progress is given by

$$E \left[\max \left\{ \frac{\|\theta\|^2 - \|\theta + \mathbf{Z}\|^2}{\|\theta\|^2}, 0 \right\} \right] = E \left[\max \left\{ 1 - \frac{S_n}{\|\theta\|^2}, 0 \right\} \right].$$

It will be useful to normalize the relative progress by the dimension n . This quantity will be called *normalized progress*. Owing to eqn. (2) one obtains the normalized progress

$$E \left[\max \left\{ n \left(1 - \frac{S_n}{\|\theta\|^2} \right), 0 \right\} \right] \approx E[\max\{-\gamma^2 + \gamma \sqrt{4 + a\gamma^2/n} \cdot N, 0\}]. \quad (3)$$

Proposition 4. Let $\eta^2 = V[Z] = E[Z^2]$ and suppose that $V[Z^2] = a\eta^4 = aV[Z]^2$ for some $a > 0$. If $n \gg 1$ then the expected normalized progress rate of the $(1+1)$ -EA is asymptotically given by

$$h(\gamma, a, n) = 2\gamma \sqrt{1 + \frac{a\gamma^2}{4n}} \varphi \left(\frac{\gamma}{2} \sqrt{\frac{4n}{4n + a\gamma^2}} \right) - \gamma^2 \Phi \left(-\frac{\gamma}{2} \sqrt{\frac{4n}{4n + a\gamma^2}} \right)$$

with $\gamma = n\eta/\|\theta\|$ and where $\varphi(\cdot)$ and $\Phi(\cdot)$ denote the probability density and distribution function of the standard normal distribution, respectively.

Proof: Let $W = -\gamma^2 + \gamma \sqrt{4 + a\gamma^2/n} \cdot N$ with $N \sim N(0, 1)$. The expected normalized progress as given in eqn. (3) becomes $E[\max\{W, 0\}]$. Since $\max\{W, 0\} = W \cdot 1_{(0, \infty)}(W)$, where $1_A(x)$ is the indicator function of set A , one obtains

$$E[\max\{W, 0\}] = E[W \cdot 1_{(0, \infty)}(W)] = \int_0^\infty \frac{w}{\gamma \sqrt{4 + a\gamma^2/n}} \varphi \left(\frac{w + \gamma^2}{\gamma \sqrt{4 + a\gamma^2/n}} \right) dw$$

where $\varphi(\cdot)$ is the probability density function of the standard normal distribution. The determination of the integral yields the desired result. $\square$

In principle, the same kind of approximation was presented in [5] for the special case of normally distributed mutations. Additionally, it was argued that the term $a\gamma^2/n$ in eqn. (3) becomes small for large n so that this term can be neglected. As a consequence, the random variable W reduces to $\widetilde{W} = -\gamma^2 + 2\gamma \cdot N$ and the expected normalized progress becomes

$$\tilde{h}(\gamma) = 2\gamma \cdot \varphi(\gamma/2) - \gamma^2 \cdot \Phi(-\gamma/2)$$

attaining its maximum $\tilde{h}(\gamma^*) = 0.404913$ at $\gamma^* = 1.224$ which is exactly the same result established 20 years earlier by Rechenberg [6]. Since all factorizing mutation distributions (with finite absolute moments) in Proposition 4 only distinguish from each other by the constant a , an analogous argumentation for an arbitrary factorizing mutation distribution leads to the result that the normalized improvement is asymptotically equal for all factorizing mutation distributions. Evidently, this kind of approximation is too rough to permit a sound comparison of the progress offered by different factorizing mutation distributions.

distribution	a	γ^*	$h(\gamma^*, a, 100)$
Normal	2	1.24389	0.40801
Logistic	16/5	1.25648	0.40992
Laplace	5	1.27639	0.41289
Student ($d = 5$)	8	1.31273	0.41811

Table 2. Optimal expected normalized progress rates for the $(1 + 1)$ -EA for some factorizing mutation distributions in case of dimension $n = 100$ under the assumption $E[\max\{n(1 - S_n/\|\theta\|^2), 0\}] \equiv h(\gamma, a, n)$.

Table 2 summarizes the optimal expected normalized progress rates for some factorizing mutation distributions under the assumption that the approximation of Proposition 4 is exact. The surprising observation which can be made from Table 2 is that the normal distribution is identified as yielding the least progress compared to the other distributions, provided that the assumption $h(\gamma, a, n) \equiv E[\max\{n(1 - S_n/\|\theta\|^2), 0\}]$ holds true. The validity of this assumption, however, deserves careful scrutiny since the norming constants $a_n = E[S_n]$ and $b_n^2 = V[S_n]$ used in the central limit theorem do not necessarily represent the best choice for a rapid approach to the normal distribution. In fact, there may exist constants α_n, β_n obeying $\beta_n \sim b_n$ and $\alpha_n - a_n = o(b_n)$ that lead much faster to the limit [7, p. 262]. As a consequence, it may happen that the ranking of the distributions in Table 2 is reversed after using these (unknown) constants. Thus, unless the error of the approximation of Proposition 4 has been quantified, this kind of approximation is also too rough to permit a sound ranking of the

mutation distributions. Nevertheless, the small differences in Table 2 provide evidence that (at least for $n \geq 100$) every factorizing mutation distribution offers a local convergence rate being comparable to that of a normal distribution.

The quality of the approximation in Proposition 4 can be checked in case of normally distributed mutations. As shown in [5], the random variable $V_n = S_n/||\theta||^2$ follows a noncentral χ^2 distribution with probability density function

$$f_{V_n}(v; \delta) = \frac{\delta^2}{2} v^{(n-2)/4} \exp\left(-\frac{\delta^2(v+1)}{2}\right) I_{n/2-1}(\delta^2 \sqrt{v}) \cdot 1_{(0, \infty)}(v)$$

where $I_m(\cdot)$ denotes the m th order modified Bessel function of the first kind and where $\delta = ||\theta||/\eta$ is the noncentrality parameter. Since $V_n > 0$ one obtains $\max\{n(1 - V_n), 0\} = n(1 - V_n) \cdot 1_{(0,1)}(V_n)$ and hence

$$g(n, \delta) = E[\max\{n(1 - V_n), 0\}] = n \int_0^1 (1 - v) f_{V_n}(v; \delta) dv. \quad (4)$$

This integral can be evaluated numerically for any given n and δ . Since $\delta = ||\theta||/\eta$ and $\gamma = n\eta/||\theta||$ it remains to maximize the function $g(n, \delta) = g(n, n/\gamma)$ with respect to $\gamma > 0$. For example, in case of $n = 100$ a numerical optimization leads to $\gamma^* = 1.224$ with $g(n, n/\gamma^*) = 0.4049$. Figures 3 & 4 show that the optimal variance factor γ^* and the optimal normalized progress $g(n, n/\gamma^*)$ quickly stabilizes for increasing dimension n . In fact, the theoretical limits are almost reached for $n = 30$.

A similar investigation might be made for other mutation vectors $\mathbf{Z}$ with factorizing mutation distributions, if the distribution of $S_n = \sum_{i=1}^n (\theta_i - Z_i)^2$ were to be known. But this does not seem to be the case. For this reason and realizing that the knowledge of the true limits is of no practical importance, it is refrained from taking the burden of determining the density of S_n for other mutation vectors.

Even numerical simulations do not easily lead to a statistically supported ranking: Although the average of the outcomes of random variable

$$Y = \max\{n(1 - S_n/||\theta||^2), 0\}$$

is an unbiased point estimator of the expectation, there is neither a standard parametric nor standard nonparametric test permitting a statistically supported decision which mean is the largest among the random variables Y generated from different mutation distributions. For example, the parametric t -test presupposes at least approximative normality of Y whereas the nonparametric tests require the continuity of the distribution function of Y . Neither of these requirements is fulfilled, so that it would be necessary to develop a specialized test for this kind of random variables. This is certainly beyond the scope of this paper.

Instead, the attention is devoted to the expected progress rates of the $(1, \lambda)$ -EA. Since this EA generates $\lambda \geq 2$ offspring independently with the same distribution and accepts the best among them, the expected progress is simply

$$E\left[\max_{i=1, \dots, \lambda} \{||\theta||^2 - ||\theta + \mathbf{Z}_i||^2\}\right].$$

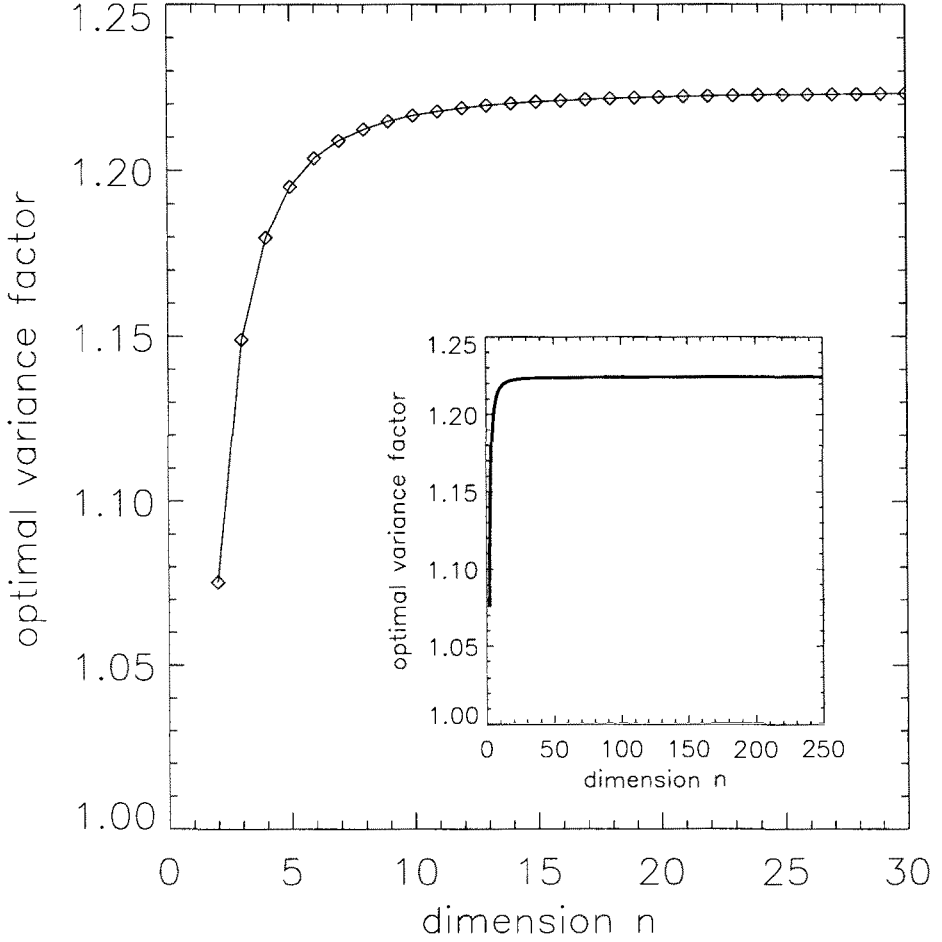


Fig. 3. The optimal variance factor γ^* in case of normal mutation vectors for increasing dimension n .

Following the lines of Proposition 4 and owing to eqn. (2) the normalized expected progress is approximately

$$h(\gamma, a, n) = -\gamma^2 + \gamma \sqrt{4 + a \gamma^2 / n} \cdot E[N_{\lambda:\lambda}] \quad (5)$$

where $N_{\lambda:\lambda}$ denotes the maximum of λ independent and identically distributed standard normal random variables. Let $c_\lambda = E[N_{\lambda:\lambda}]$. Then the optimal expected normalized progress rate of the $(1, \lambda)$ -EA is attained at

$$\gamma^* = \left(\frac{2 c_\lambda^2}{1 - a c_\lambda^2 / n + \sqrt{1 - a c_\lambda^2 / n}} \right)^{1/2}$$

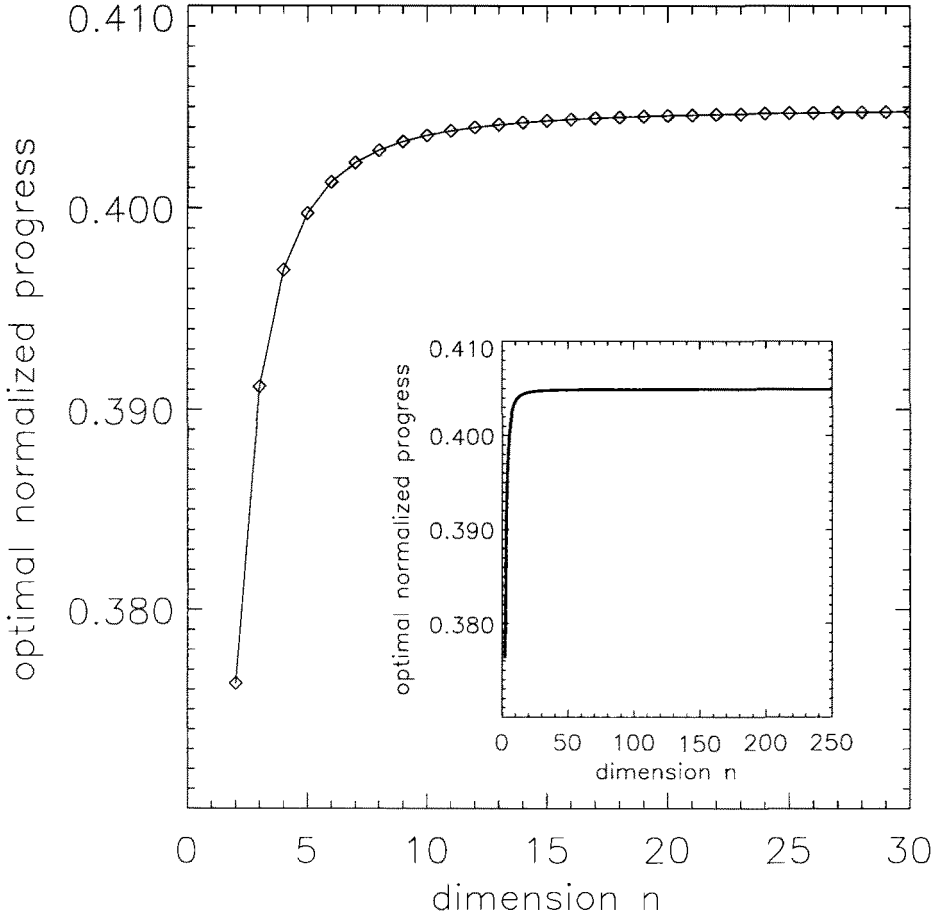


Fig. 4. The optimal normalized progress $g(n, n/\gamma^*)$ in case of normal mutation vectors for increasing dimension n .

which reduces to $\tilde{\gamma}^* = c_\lambda$ as $n \rightarrow \infty$. In general, the relation $h(\gamma^*, a, n) > c_\lambda^2$ is valid. Moreover, $h(\gamma, a + \epsilon, n) > h(\gamma, a, n)$ for arbitrary $\gamma > 0$ and $\epsilon > 0$ which follows easily from eqn. (5). Consequently, the expected progress becomes larger for increasing $a > 0$, provided that the approximation given in (2) holds with equality. But it has been seen in case of the $(1+1)$ -EA that this approximation does not permit a sound ranking of the distributions. At this point there might arise the question for which purpose the approximations presented in this paper are good for at all. The answer is given in the next section.

3 Conclusions

Under the conditions of the central limit theorem an asymptotical theory of the expected progress rates of simple evolutionary algorithms has been established. If the mutation distributions are factorizing and possess finite absolute moments up to order 4, then each of these distributions offer an almost equally fast approach to the (local) optimum. The optimal variance adjustment w.r.t. fast local convergence is of the type $\eta_k = \gamma \|\mathbf{X}_k - \mathbf{x}^*\|/n$ for each of the distributions considered here. This implies that the self-adaptive adjustment of the “step sizes” originally developed for normal distributions needs not be modified in case of other factorizing mutation distributions. In the light of the theory developed in [8] it may be conjectured that these results carry over to population-based EAs *without* crossover or recombination.

Finally, notice that Student's t -distribution with d degrees of freedom converges weakly to the normal distribution as $d \rightarrow \infty$ whereas it is called the Cauchy distribution for $d = 1$. All results remain valid for $d \geq 5$. Lower values of d cannot be investigated within the framework presented here, since it was presupposed that the absolute moments of $\mathbf{Z}$ are finite up to order 4. If these moments do not exist the central limit theorem does not hold true. Rather, then there emerges an entire class of limit distributions [9] as already mentioned in [1]. But this case is beyond the scope of this paper and it remains for future research.

Acknowledgment

Besides my thanks to all anonymous reviewers whose suggestions led to several improvements, special thanks must be addressed to one of the reviewers who detected a severe flaw in my argumentation following Proposition 4. Therefore, the part after Proposition 4 was completely revised. As a result, the message of this paper is completely different from the original (wrong) one.

Finally, it should be mentioned that this work is a result of the *Collaborative Research Center “Computational Intelligence” (SFB 531)* supported by the German National Science Foundation (DFG).

References

1. C. Kappler. Are evolutionary algorithms improved by large mutations? In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Parallel Problem Solving From Nature—PPSN IV*, pages 346–355. Springer, Berlin, 1996.
2. X. Yao and Y. Liu. Fast evolutionary programming. In L. J. Fogel, P. J. Angeline, and T. Bäck, editors, *Proceedings of the Fifth Annual Conference on Evolutionary Programming*, pages 451–460. MIT Press, Cambridge (MA), 1996.
3. X. Yao and Y. Liu. Fast evolution strategies. In P. J. Angeline, R. G. Reynolds, J. R. McDonnell, and R. Eberhart, editors, *Proceedings of the Sixth Annual Conference on Evolutionary Programming*, pages 151–161. Springer, Berlin, 1997.

4. W. Feller. *An Introduction to Probability Theory and Its Applications*, Vol. 2. Wiley, New York, 2nd edition, 1971.
5. T. Bäck, G. Rudolph, and H.-P. Schwefel. Evolutionary programming and evolution strategies: Similarities and differences. In D. B. Fogel and W. Atmar, editors, *Proceedings of the 2nd Annual Conference on Evolutionary Programming*, pages 11–22. Evolutionary Programming Society, La Jolla (CA), 1993.
6. I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog Verlag, Stuttgart, 1973.
7. Y. S. Chow and H. Teicher. *Probability Theory*. Springer, New York, 1978.
8. G. Rudolph. *Convergence Properties of Evolutionary Algorithms*. Kovač, Hamburg, 1997.
9. B. V. Gnedenko and A. N. Kolmogorov. *Limit Distributions for Sums of Independent Random Variables*. Addison-Wesley, Reading (MA), revised edition, 1968.

Methodologies

Wings Were Not Designed to Let Animals Fly

Eric Dedieu, Olivier Lebeltel, Pierre Bessi re
Leibniz-IMAG laboratory, Grenoble, France
Eric.Dedieu@imag.fr

“Functional change in structural continuity,” i.e., the opportunistic evolution of functions together with structures, is a major feature of biological evolution. However it has seldom struck a robotician’s mind as very relevant for building robots, i.e., for *design*. This paper proposes starting points for investigating this unusual issue.

1. Introduction

Wings were not originally designed to let animals fly. Insect wings probably were at first heat-regulating structures, which happened to give some gliding abilities as a side-effect of increasing the global size of the animal (Gould 1991 ch. 6). Because it gave a selective advantage, this emergent functionality eventually became the main one of the structure. This is called “functional change in structural continuity” (Gould 1991 ch. 6; 1993 ch. 6, 7 & 22; 1980; 1977b; Jacob 1981 ch. 2). It is usually felt as an unusual idea. You will probably find figure 1 incongruous; however it is not unlike natural evolution often worked.

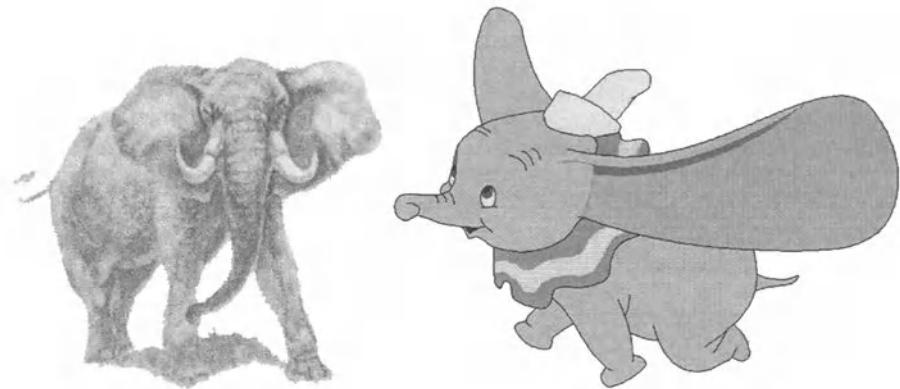


Fig. 1. Functional change in structural continuity:
Dumbo’s wings were originally not designed to let him fly.

It is not surprising, then, that this kind of “trick” has seldom struck a robotician’s mind as very appropriate for developing robots. Roboticians are, before all, engineers: they develop robots to achieve well defined, *intended* tasks. But Nature teaches us that there may be a co-evolution of functions and structures. Evolution is not fully *driven by* predefined functionalities demanding adaptation, but functionalities themselves may *opportunistically emerge* from the structural mechanisms of evolution. Can we adapt this lesson of natural science to robotics? That is, robot

development would not necessarily be fully driven by intended functionalities. Functionalities might be altered through development, and unintended but relevant functionalities might also emerge from it.

This paper proposes a first step in that direction. Though the issue now strikes us as essentially an evolutionary one, our first approach was grounded on probabilistic methods (Dedieu 1995). We turned to evolutionary thinking only lately, and are not yet able to provide technical results involving evolutionary methods. However we feel that our experience is relevant to artificial evolution (AE), if only for the void that we observed when surveying the AE literature that is related to robotics or artificial life. A major aspect of biological evolution seems to have escaped AE investigations.

Section 2 illustrates the issue with two “true stories” that will seem familiar to roboticians — however usually happening without attracting their attention. Section 3 clarifies the scientific context of our work by putting the lessons from these particular stories in a wider perspective. Section 4 then proposes a principle for making opportunities for development emerge, and offers AE as the most promising perspective for our research. Section 5 is a critical discussion of related work. Section 6 concludes and summarizes our contribution.

2. Two true stories

The door-step story

One day I was watching a mobile robot that was doing path-planning and path-following, using a geometric map of the laboratory. When passing a door, however, a small bulging door-step caused it to slip and collide the wall. This is what we call an *unexpected event*, in the strong meaning of: *unanticipated by the designer of the robot*. A crude way to deal with it could be to remove the door-step, i.e., to control the environment to make the model valid (which implicitly assumed no slipping). This solution is efficient for industrial robotics, but it is hardly acceptable for autonomous robotics. Indeed, a major goal of autonomous robotics is to cope with such non-controlled situations as automatically as possible.

Actually this particular problem of door-steps happens very seldom. Usually, the rubber of the wheel does not slip and the problem is handled by low-level control. One wheel passes the door-step first, then the other catches up with it without anyone noticing it. Higher level tasks do not have to know about this parasitic thing, and the designers are satisfied with this solution.

But was the door-step really a “problem” to be handled, a “parasite” to be get rid of? This is the usual regard of the engineer on anything preventing the intended task to be done. And, once it is handled and does no longer interfere with the task, it becomes irrelevant, to be ignored.

A possible suggestion, however, would be to use this empirically observed particular behavior at door-steps to help the robot recognize a door location. For that matter, this robot was using sophisticated relocation methods using camera vision of landmarks. Opportunistic solutions are not often considered by roboticians.

The dancing Khepera

Another true story makes this point even more striking. Figure 2 shows the Khepera robot (a product of the LAMI laboratory of the EPFL at Lausanne, Switzerland). It is controlled by its left and right wheel-speeds, and uses a ring of photocells as sensors. It was programmed with a light-following reflex, driving in the direction of the most activated cell (figure 2).

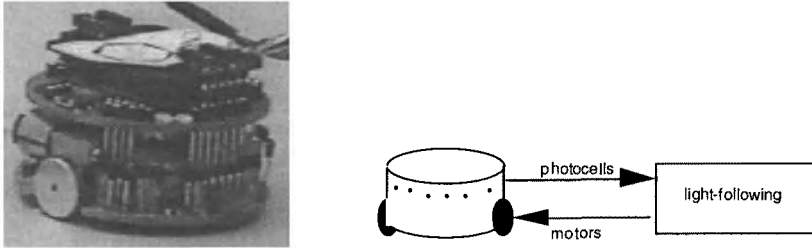


Fig. 2. The Khepera robot and the light-following reflex.

This reflex was being tested by letting the robot follow a torch. But when turning off the torch, the robot did something totally unexpected. It turned back, heading toward a large window *and avoiding obstacles in its path*, then turned toward a half-closed door. It managed the narrow passage to the office room behind, and finally kept “dancing” under the light bulb, as if showing that it had reached its goal. This interpretation forced itself upon all the witnesses, who would recall the story spontaneously several years later.

After thought, this behavior is no longer surprising. Obstacles were avoided because of their shadows, then the robot was caught by the light cone coming through the door, which guided it to the office. The final dance was just indecision in a local maximum of light intensity.

Unless what had happened in the door-step story, this was a pleasant surprise, hardly a “parasite”. However, people were just pleased to observe it as an unintended emerging behavior. A pleasant surprise indeed, but thanks to luck only, and with no future. They never sought to *exploit* it — for example, to opportunistically invent new tasks that could take advantage of this discovery.

Such unexpected phenomena as door-steps or shadows are very common in robotics — and they are scarcely regarded as relevant to anything interesting. Our basic issue is: can’t we imagine systematic approaches to opportunistically (indeed, “creatively”) take advantage of them?¹

This question reflects unusual concerns that we will detail now.

¹ This was hardly our initial concerns when starting robotic investigations. However, the unexpected witnessing of the two “true stories” changed our reflections and eventually our research goals. This is how our main thesis has applied to our own work!

3. Scientific concerns

Autonomy and development

Classical engineering applications are right to try to get rid of parasitic phenomena, i.e., to insure a stable problem specification. This allows to tackle it in a systematic, essentially top-down way. If the context of the application qualitatively or quantitatively changes, engineers may re-identify their models, re-design their work, or invent a new contraption.

We intend to make such aspects as automatic as possible, and we propose that this should be the meaning of *autonomy* for artificial systems. This term often means “achieving its tasks with no human intervention in some real application.” This is hardly different from “automated”. We regard the specificity of autonomy, from the design perspective, as lying in the contingent aspect of the environment. The eventual appearance of conditions unexpected by the designer is inescapable, which makes continuing and adaptive development necessary. Being autonomous means having automatic mechanisms for dealing with such adaptation. This is where opportunism and bottom-up methods show up.

In robotics the level at which adaptive change occurs is ambiguous. There is no lifetime and no reproduction (except as metaphors). Nearly all changes focus on software, for experimental convenience. The distinction between evolution and development is then hard to draw (unless biological metaphors are explicitly intended). We generically will use “development” for continuing design and refinement, possibly involving evolutionary techniques.

Design and artificial evolution

Task evolution does not fit well in the usual design paradigms. Should the very tasks of a robot be allowed to change opportunistically, they could no longer be the *constant guide for design* that they usually are. What is to replace them? Hopefully, evolutionary methods seem able to support “functionally open” design principles, such as functional change in structural continuity.

There is an old debate on how the notion of design might fit in natural evolution, related to the status of finalism and teleology within Darwinism. Dennett (1996) argues for evolution as engineering, but we agree with Jacob (1981) or Laganey (1979) to find it a deceptive notion. The way evolution achieves “design” is dramatically different from human engineering. It is more like an opportunistic accumulation of “tinkering”, which in human practice would inescapably lead to bad engineering. And indeed, AE has mostly focused on the aspects of evolution that are more *directly* exploitable for design. Evolutionary methods have been typically regarded as optimization processes for some fixed “fitness” formula coding intended functionalities. This makes AE an efficient tool, but does hardly exhaust the inspiration that design might take from biological evolution.

In robotics the question of design is quite concrete: we know for sure who the designer is, and how he usually *does* think in finalistic terms. We propose

opportunism¹ and open-ended development as an alternative to finalism that is worth investigating. However, the idea is not mature enough to yield applied research. Before we can achieve substantial engineering repercussions, our work in robot development should be considered as basic scientific research related to cognitive science. The purpose of cognitive science is to model animal and human cognitive abilities, i.e., understand and reproduce the mechanisms of “intelligent” behavior.

Embodiment and contingency

Even in the cognitive side of robotics (the “animat” approach), people still most often focus on problem-solving or task-achieving issues. The fact that a robot is in sensorimotor interaction with its environment is usually considered a problem, rising many hard and tedious constraints like calibration, mechanical or electronic adjusting, motor powering, sensor noise, etc. These constraints are regarded as taking time and, all things considered, parasitic compared to “truly” interesting cognitive issues such as high-level navigation, planning or task selection. Therefore, in order to “get to the point” more easily, many robotics systems are developed as computer simulations, somewhat assuming that a physical implementation would raise purely technical problems. When they are developed in a physical environment, this environment is carefully adapted and controlled to let the chosen method work. For example, grasp planning in CAD models is often implemented using objects that were manufactured according to their model — a strange causality between models and reality. (Note that fixing the environment is a very good engineering practice, only inadequate to support claims about *autonomy*.)

The use of simulations actually eludes a major specificity of robotics, which is the *physically embodied* nature of robots. This means their interaction with a complex world that can be only partially and inaccurately described, let alone predicted. Embodiment thus sooner or later leads way to unexpected, contingent phenomena, which should not be regarded as just parasites: they may be the very ones to be understood and, even, *exploited*. Our concern is to study and explore the contribution (rather than interference) of embodiment in developing cognitive abilities (Webb & Smithers 1992).

Animal evolution repeatedly showed a knack for exploiting contingent phenomena. The study of embodied systems (robots) offers a concrete approach to what contingency looks like in artificial systems, as well as a feel of its importance to the issue of autonomy. It is a natural field of investigations for artificial evolution.²

4. The dancing Khepera — revisited

Our first approach was to propose to the human designer a principle for handling opportunism and open-ended development himself. This section presents and illustrates this principle. However we did not succeed in completing a satisfying methodology to apply it repeatedly in a systematic way. Robot development was to be

¹ A very different concern about opportunism may be found (e.g. Maes 1992), as watching for an opportunity to do something that one was somehow expecting to do.

² We do not claim that embodiment is necessary to study opportunism. It should be possible to devise kinds of simulations allowing for interesting contingent phenomena.

guided by the observer's satisfaction and imagination — informal but manageable criteria that have made first investigations possible but have proved, in practice, more limited than expected, because of the lack of technical tools. We hope that an evolutionary approach might overcome those limits, as it has done for other kinds of design (Harvey, Husbands, Cliff, Thompson & Jakobi 1977). See Dedieu (1995) for further discussion about the probabilistic and non-evolutionary aspects of this work. Evolutionary perspectives will appear in the next section.

Let us now get back to the observation that following lights may (in certain conditions) lead to avoid obstacles as a side-effect of avoiding their shadows ("dancing Khepera" story).

For simplicity, the translation speed of the robot is held constant, and only the rotation speed (noted *rot*) is controlled for action. The Khepera can also use a ring of proximeters, accounted for by two preprocessed variables: *dir*, roughly being the direction of the nearest obstacle, and *prox*, roughly being the proximity of the nearest obstacle.¹ The robot moves on a 1m × 1m platform, obstacles being either the outer walls or big colored Lego bricks. It is controlled through a serial line, sending its sensory values and receiving orders with a fixed 10 Hz frequency.

When avoiding obstacles, the values of the proximeters and the rotation speed are not independent. The dependence may be represented as a probability distribution on the possible values of *rot*, *dir* and *prox*, sketched on figure 3 (the Bayesian framework that we have adopted is detailed and discussed in Dedieu (1995)).

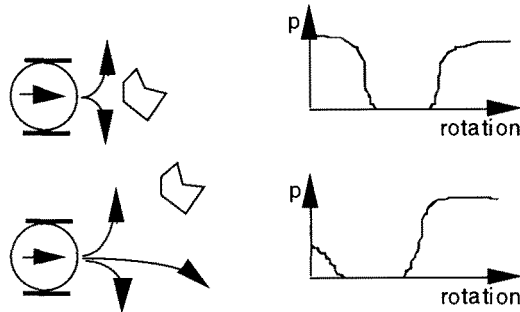


Fig. 3. Some schematic views of a probabilistic dependence between *rot*, *dir* and *prox* that conveys an obstacle-avoiding behavior. The curves sketch probability distributions. When facing the obstacle and quite close to it, the robot must be turning left or right. When close to an obstacle, the robot is unlikely to be turning toward it, and more likely to be turning right than left. In free space, the movement would be unconstrained (flat distribution).

Such a dependence may be observed as soon as the robot is avoiding obstacles *whatever the reason for it, be it designed, or emergent, or by chance*. By discovering that such a dependence holds while following lights, the robot may recognize that it is being avoiding obstacles even though not programmed for it (figure 4).

¹ Precisely, the six front infrared proximeters being called $\pi_1 \dots \pi_6$, $dir = \frac{90 (\pi_6 - \pi_1) + 45 (\pi_5 - \pi_2) + 5 (\pi_4 - \pi_3)}{1 + 9 (\pi_1 + \pi_2 + \pi_3 + \pi_4 + \pi_5 + \pi_6)}$ and $prox = \frac{\max(\pi_1, \dots, \pi_6)}{16}$. Actually photocells and proximeters involve the same physical sensors.

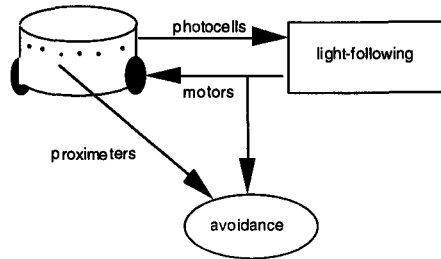


Fig. 4. The robot can recognize an emerging obstacle avoidance aspect in its behavior, by observing a dependence between its proximeters and motor values. In these figures, square boxes show dependences that are controlling the robot, whereas round boxes show dependences used as observers.

Moreover, if this dependence is initially unknown, the robot can learn it (i.e., identify the parameters of a parametric model of probability distributions) during the light-following behavior. Then it can use it back to avoid obstacles even when they cast no shadows (figure 5).

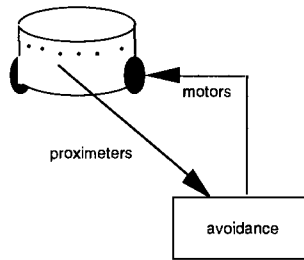


Fig. 5. An opportunistically learned dependence may be used to program an obstacle avoiding reflex using the proximeters rather than the photocells, i.e., avoid obstacles even when there are no shadows.

In a sense, such a dependence can be seen as able to represent an emergent functionality, because it allows to discover it and then take advantage of it.

This may remind you of a very simple supervised learning. Somehow, it is. However, the approach is *conceptually* very different from the usual learning paradigms. The robot does not learn what to do in a given sensory situation, i.e., it does not learn input-output associations. Instead, it learns a dependence to be used with Bayesian inference, with no imposed inputs and outputs: all variables can be both.¹ Figures 4 and 5 have thus shown a same dependence used in two different ways. These ways were still quite “classical” uses, though. We will now describe a more surprising one.

Let us come back to figure 4, in which the robot is avoiding obstacles through avoiding their shadows with the light-following reflex. In the figure, the “avoidance” box was used as a passive recognition device, surveying a possible dependence

¹ Note also that there is no notion of goal in building an avoidance dependence. The robot does not notice that it does not hit obstacles (another possible definition for obstacle avoidance). What is characterized by “obstacle avoidance” is *not* a goal, but a behavior, i.e., a particular observable interaction with the environment.

between motor and sensory values. But it can also be used for inferring motor values, given sensory values. In figure 5 these inferred values were used to control the robot, but they need not necessarily be used that way. They may remain just “inferred values” while the robot is actually controlled with another mechanism.

This is done in figure 6. Now, the question is: shouldn’t the comparison of the *actual* motor values and the *inferred* one (see the figure) give some useful information?

If the environment does not change, since the “avoidance” dependence was identified from the very behavior generated from the “light-following” one, they should roughly predict the same values. In that case, the two boxes in the figure can be seen as two characterizations of (different aspects of) the robot’s behavior. They are both “valid,” one using the photocells, one using the proximeters. Both will predict motor values that are compatible (more precisely, they will infer probability distributions that should be compatible in some sense).

Let us now change the environment slightly, by adding obstacles that are higher than the previous ones. Those will cast longer shadows, and the robot will avoid them from a greater distance. The predictions of the motor values from the “avoidance” dependence will be fooled: the values will be underestimated. Then, by comparing the motors as given by the light-following dependence and by the obstacle-avoiding one, *the difference may be related to the height of the obstacles* (figure 6).

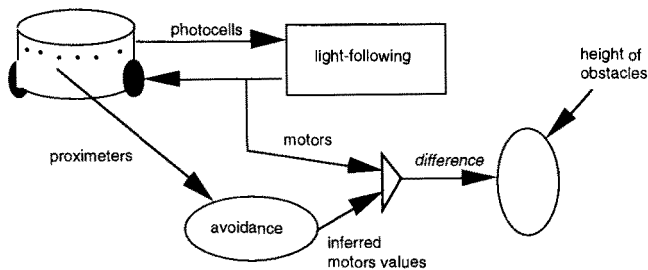


Fig. 6. The light-following dependence is controlling the robot again. The avoidance one is used, in an a-priori unexpected way, to help characterize the height of obstacles in sensori-motor terms (see text).

Figure 6 was an opportunistic idea. If given the task of measuring the height of obstacles, nobody would have *designed* such a scheme: let the robot reactively avoid the obstacle’s shadow and measure an inference discrepancy as shown in the figure. They would rather devise something using a camera, for example, or a geometric map given by the designer. As an echo to the paper’s introductory sentence that “wings were not designed to let animals fly,” we can add, “proximeters were not designed to let robots measure the height of obstacles!”...

What general lessons should be drawn from this particular example? The basic principle is that when the context changes, a dependence does not necessarily become useless or invalid: it can yet be used in other opportunistic ways, depending on the concrete new context. We actually propose the following principle of functional change in structural continuity for artificial systems (pictured on figure 7):

1. A particular behavior is characterized by observable stable probabilistic dependences between sensori-motor variables. It is the robot’s only resource for

characterizing its behavior, even if human observers will more naturally use functional terms (see previous footnote).

2. The dependences are permanently making predictions (inferences about some variables), and confronting the predictions to the observed values. This yields a discrepancy value. Note that a discrepancy is not an error: the situation in which it is computed may be very far from that in which the dependence was intended to be predictive. However the inference is made and the discrepancy computed even when not knowing a priori whether it will be interpretable and interesting.
3. The discrepancy value is checked against other variables. If a systematic dependence is found, it is considered as relevant information that should be potentially exploitable to achieve some new functionality. (Currently, it is up to the observer to interpret and imagine how such new informational resource can be exploited. See Dedieu (1995) for a thorough discussion of the role of the observer in developing autonomous systems.)

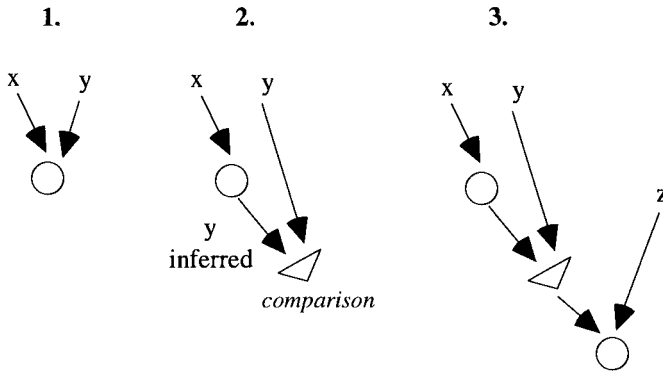


Fig. 7. Development of a new informational resource — see text above. Circles represent dependences, triangles represent a comparison operation.

This principle was first used as a guideline for designing experiments that were implemented in an ad-hoc way. It can account for building the dependence for obstacle avoidance while following light, and building the dependence for recognizing obstacle heights. Dedieu (1995) details two other examples than the Khepera one.

However, this principle raises lots of problems when thinking of making it an automatic process. The greatest difficulty may be that it is a pure bottom-up scheme. As such, it lacks some global guide, which can only be top-down. This is where evolutionary perspectives are attractive, providing the top-down notion of selection. As we shall see now, AE covers the bulk of related work.

5. Related work

The progressive elaboration of functionalities was indeed a concern for the early investigations in bottom-up, behavior-based robotics (Braitenberg 1984). The paradigmatic example is that of Brooks at the MIT AI Lab, who endeavored to design functionalities in a bottom-up way using the “subsumption architecture” (Brooks 1991). However, aspects of concrete bottom-up *elaboration* only appear in early

papers (Brooks 1989). Most of the work concretely consisted in implementing pre-defined functions as sensor-based reflexes running in parallel, these reflexes *not* being designed progressively and separately. Design was global (e.g. Brooks 1991 fig. 2, Connell 1988); only debugging was bottom-up in that intermediate stages yield effective robot behaviors relying on sensors as primitive information sources. Bottom-up elaboration can be found again in Horswill (1993), who was concerned with the analysis of specialized behaviors to provide design clues for further development.

Pierce and Kuipers (1997) explore how a robot can structure an initially uninterpreted sensori-motor apparatus using only experimental data and minimal structural prior knowledge about sensori-motor interactions. Goals emerge as structures that happen to be controllable by the robot. The corresponding behavior, for example wall-following, appears as the result of doing what happens to be structurally feasible, rather than as the initial target for design. Such concerns are very close to ours regarding bottom-up development.

Miglino, Nolfi & Parisi (1996) and Lund & Parisi (1996) explored how neutral (previously nonfunctional) structures may be recruited by artificial evolution to help achieve some function. Agent development is modeled at several levels of organization. Change at some of these levels may be neutral regarding fitness, hence selection, but neutral structures may eventually serve as a basis for adaptive change. Nonetheless the agents were developed for a single fixed task (gathering food). By contrast, it is the very tasks that we propose to develop, which should call for other mechanism than only ensuring the availability of neutral structured material for bottom-up development.

Other models of development also “missed” functional change by focusing on single tasks. Kodjabachian & Meyer (1997) proposed a model the incremental aspect of which is to add “plug-in” modules to improve on a basic locomotion module. Functional change might appear in such schemes by letting plug-in modules change their contribution toward a different behavioral module; however no concern was shown for such an idea. Dellaert & Beer (1996) or Sims (1994) addressed development, too, but showed no interest for the intermediate stages of development; only the “adult” form is evaluated regarding a single task. The (biologically obvious) co-development of tasks and organisms is shunned, which we think would likely involve functional change.

In the “evolutionary robotics” framework, top-down control over a bottom-up behavior-building process is achieved by specifying only global behavioral criteria and letting intermediary functional structures emerge without being explicitly asked for. Cliff, Harvey & Husbands (1996) analyzed some of such evolved structures, namely parts of neural networks. However their analysis was only descriptive; they hardly sought to *exploit* emergent functional structures in the sense that we propose. At stake would be the evolution from an *already adapted* agent to another adapted agent, not as successive generations toward a same intended adaptive goal, but across contextual change implying to *reuse* interesting structures in another way. Typically, here is an aspect where functional change must have occurred but that the mentioned authors have ignored. Before relying on vision only, the controller had to rely on proximal sensors first (whiskers). Successive generations then showed vision taking over. During this evolution toward vision-based behavior, collections of units may have been functional regarding whiskers-based behavior and been reused for vision-based behavior. But the analysis that was done considered only the final stage

controller.¹ Such “modality transfer” between proximal and distal sensors is also described by Mondada (1996, ch. 4) or Steels (1994), and is indeed related to functional change.

Mondada (1996, ch. 5) lets a Khepera robot controller evolve by a genetic algorithm, with the fitness function *not* encoding the major “viability” constraint of the robot’s behavior: recharging batteries at a given “nest” location every 20s or less. Evolved controllers that do not fit the survival constraint are just eliminated. The explicit fitness function to be optimized only describes a minor goal: foraging. The result is a robot that comes back to the “nest” every 19s, and goes foraging meanwhile. The unspecified “task” naturally emerges from the viability constraints, and with an optimal timing, without being processed as an optimization problem. This illustrates an idea that is dear to us, i.e., that viability and task-optimization should be more separate notions. This is an echo in artificial science (i.e., involving design) of an old argument about the nature of natural selection (in evolutionary science see especially Eldredge (1996); in cognitive science see Varela, Thompson & Rosch (1993, ch. 9), Saunders (1989), Stewart (1993), Stengers (1997)).

Much of the work about emergent functionalities (or emergent structures) actually lies in the framework of auto-organization, i.e., is studied at another level than in this paper (Steels 1994; Stengers 1997). Auto-organization points at modeling issues, whereas we point at the relevance of opportunism during design and development, with no reference to collective, cooperative or across-level phenomena. Nonetheless, those may well be the technical basis that we need for further investigation.

We have described the closest concerns to functional change that we know of — and functional change was never explicitly addressed. Even in bottom-up approaches, robots are usually meant to be problem-solving or task-achieving systems, which implicitly means solving *pre-specified* problems and achieving *pre-intended* tasks. This has led to very positive achievements; but it may be unfortunate for studying the most challenging aspects of autonomy. As Varela put it (1989), “The most important ability of all living cognition is precisely, to a large extent, to *pose* the relevant questions that arise each moment of our life.”² Later he has, indeed, written (Varela, Thompson & Rosch 1993): “[The enactivist approach] may be deployed in all fields of cognitive science — provided that we accept to relax the rule of considering only the achievement of precise and predefined tasks when assessing performance.”³ (This is unfortunately illustrated with the work of Brooks, which (to our mind) is not really relevant to this particular issue.)

What “positive” perspectives can we then draw to tackle functional change? The above discussion of related work already provides some clues to elaborate on existing work. We speculate that the most important point is ontogeny. AE has much focused on selection, and only marginally on the generative mechanisms themselves; however

¹ Nonetheless, they observe that a unit initially intended as “input” could be opportunistically used as a “hidden” one.

² «La plus importante faculté de toute cognition vivante est précisément, dans une large mesure, de *poser* les questions pertinentes qui surgissent à chaque moment de notre vie.» p. 90.

³ «[L’approche par l’énaction] peut être déployée dans tous les domaines des sciences cognitives — à la condition, toutefois, que nous consentions à assouplir l’exigence de ne juger les performances qu’en rapport avec des tâches précises et prédéfinies à effectuer.» p. 282.

they are probably responsible from both making functions stable and occasionally allowing novel ones to emerge (Gould 1977a; 1993 ch. 27; Saunders 1989).

6. Conclusion

This paper has presented opportunistic development as an unusual concern for robotics, which is, however, deeply rooted of what our experience of “physically embodied” mobile robots is: unexpected problems, difficulty to account for the details of a non-manufactured environment, practical importance of those details. The claim here is that “parasitic phenomena” are not only obstacles to overcome, but may be a richness to exploit in an opportunistic way.

Two examples have illustrated that unexpected phenomena might be exploited to develop a robot’s abilities opportunistically and incrementally. However, this demands to relax the importance of pre-defined intended tasks as a guide for robot development: the robot designer has to accept a possible co-evolution of functions and structures (functional change). This is opposed to the common “engineering” frame of mind of roboticists. As such, our approach still conveys concerns in basic research on autonomy, related to cognitive science, rather than applied purposes.

Artificial evolution was proposed as the most promising idea to pursue this research and develop more systematic methods to achieve the co-evolution of functions and structure. But almost everything remains to be done to really account for functional change in AE. Our current contribution has no other ambition than posing the problem.

7. References

- Braitenberg, V. 1984 *Vehicles : experiments in synthetic psychology*. MIT Press/Bradford Books.
- Brooks, R.A. 1989 A robot that walks: emergent behaviors from a carefully evolved network. *Neural Computations*, n° 1:2.
- Brooks, R.A. 1991 Intelligence without Representation. *Artificial Intelligence*, vol. 47 (submitted in 1987).
- Cliff, D., I. Harvey & P. Husbands 1996 Artificial evolution of visual control systems for robots. In Srinivisan & Venkatesh (eds), *From living eyes to seeing machines*. Oxford university press.
- Connell, J.H. 1988 A behavior-based arm controller. AI MEMO 1025, MIT.
- Dedieu, E. 1995 Contingent representation: toward a reconciliation between the functional and structural approaches to robot autonomy. PhD thesis (in French), INPG university, Grenoble, France.
- Dellaert, F. & R.D. Beer 1996 A developmental model for the evolution of complete autonomous agents. SAB’96 conference “From Animals to Animats 4”, MIT Press/Bradford books.

- Dennett, D.C. 1996 *Darwin's dangerous idea*. Simon & Schuster/Touchstone books.
- Eldredge, N. 1996 *Reinventing Darwin*. Wiley.
- Gould, S.J. 1977a *Ontogeny and Phylogeny*. Harvard University Press.
- Gould, S.J. 1977b The problem of perfection. In *Ever since Darwin*, W.W. Norton.
- Gould, S.J. 1980 The panda's thumb. In *The panda's thumb*, W.W. Norton.
- Gould, S.J. 1991 *Bully for brontosaurus*. W.W. Norton.
- Gould, S.J. 1993 *Eight Little Piggies*. W.W. Norton.
- Harvey, I., P. Husbands, D. Cliff, A. Thompson & N. Jakobi 1997 Evolutionary robotics: the Sussex approach. *Robotics and autonomous systems*, in press.
- Horswill, I.D. 1993 Specialization of perceptual processes. PhD thesis, MIT.
- Jacob, F. 1981 *Le jeu des possibles*. Livre de poche.
- Kodjabachian, J. & J.A. Meyer, 1997 Evolution and development of neural networks controlling locomotion, gradient-following, and obstacle-avoidance in artificial insects. Submitted.
- Laganey, A. 1979 *Le sexe et l'innovation*. Seuil, coll. Points, Paris.
- Lund, H.H. & Parisi, D. 1996 Pre-adaptations in Populations of Neural Networks Evolving in a Changing Environment. *Artificial Life* 2:2.
- Maes, P. 1992 Learning Behavior Networks from Experience. In F. Varela & P. Bourguin (eds), *Toward a practice of autonomous systems: proc. of ECAL'91*, MIT Press/Bradford books.
- Miglino, O., S. Nolfi & D. Parisi 1996 Discontinuity in evolution: how different levels of organization imply pre-adaptation. In Belew & Mitchell (eds.), *Plastic Individuals in Evolving Populations*. SFI Series, Addison-Wesley, in press.
- Mondada, F. 1996 Conception de structures neuronales pour le contrôle de robots mobiles autonomes. PhD thesis, EPFL, Lausanne, Switzerland, November 1996.
- Pierce, D. & B. Kuipers 1997 Map learning with uninterpreted sensors and effectors, *AI Journal*, accepted.
- Saunders, P.T. 1989 L'évolution des formes biologiques, *Intellectica* 1993/1, n° 16, initially in English in *Leonardo*, vol. 22, n° 1, 1989.
- Sims, K. 1994 Evolving 3D Morphology and Behavior by Competition. *Artificial Life IV* proceedings, R. Brooks & P. Maes (eds), MIT Press.
- Steels, L. 1994 The artificial life roots of artificial intelligence. *Artificial Life Journal*, vol. 1,1. MIT Press.
- Stengers, I. 1997 *Cosmopolitiques 6*, La Découverte, Paris.
- Stewart, J. 1993 Au delà de l'inné et de l'acquis, *Intellectica* 1993/1, n° 16.

- Varela, F. 1989 *Connaître : les sciences cognitives*. Seuil, Paris.
- Varela, F., E. Thompson et E. Rosch 1993 *L'inscription corporelle de l'esprit*. Seuil.
Translation of *The embodied mind*.
- Webb, B. & T. Smithers 1992 The connection between AI and biology in the study of behaviour. In F. Varela & P. Bourguin (eds), *Toward a practice of autonomous systems: proc. of ECAL'91*, MIT Press/Bradford books.

Adaptation on the Evolutionary Time Scale: A Working Hypothesis and Basic Experiments

Ralf Salomon and Peter Eggenberger

AI Lab, Computer Science Department, University of Zürich
Winterthurerstrasse 190, 8057 Zurich, Switzerland
FAX: +41-1-635 68 09; Email: {salomon|eggen}@ifi.unizh.ch

Abstract. In the pertinent literature, an ongoing discussion can be found about whether evolutionary algorithms are better suited for optimization or adaptation. Unfortunately, the pertinent literature does not offer a definition of the difference between adaptation and optimization. As a working hypothesis, this paper proposes adaptation as tracking the moving optimum of a dynamically changing fitness function as opposed to optimization as finding the optimum of a static fitness function. The results presented in this paper suggest that providing enough variation among the population members and applying a selection scheme is sufficient for adaptation. The resulting performance, however, depends on the problem, the selection scheme, the variation operators, as well as possibly other factors.

1 Introduction

Evolutionary algorithms (EAs) are a class of stochastic optimization and adaptation techniques that have been successfully applied in diverse areas, such as machine learning, combinatorial problems, VLSI design, and numerical optimization [16]. EAs provide a framework that consists of genetic algorithms (GAs) [3, 7], evolutionary programming (EP) [5, 6], and evolution strategies (ESs) [12, 15]. A comparison of these different methods can be found in [2].

Each of these evolutionary algorithms is designed along different methodologies. Despite their differences, all evolutionary algorithms are heuristic population-based search procedures that incorporate random variation and selection. Typically, such population-based search procedures generate offspring in each generation. A fitness value (defined by a fitness or objective function) is assigned to each offspring. Depending on their fitness, each population member is given a specific survival probability. Most evolutionary algorithms can be described in the following generic form:

- Step 0: Initialize and evaluate the fitness of each individual of the population
- Step 1: Select the parents according to a selection scheme (e.g., roulette wheel, linear ranking, or truncation selection)
- Step 2: Recombine and mutate selected parents with a specific operator
- Step 3: Go to Step 1

EP and ESs place particular emphasis on the behavioral link between parents and offspring. This link is realized by applying small, typically Gaussian or Cauchy-distributed perturbations to all parameters x_i that specify an object. Furthermore, the amount of perturbation is self-adapted as to maximize the expected progress of the fitness value in each generation. Therefore, EP and ESs are both predominantly designed for function optimization.

GAs have also been successfully applied to various optimization tasks [7, 16]. Recent results [13], however, have tempered that success in terms of efficacy. As an explanation, the research community has been reminded [4] that Holland's original research aimed at designing and implementing robust adaptive systems as opposed to developing sophisticated optimization techniques, even though GAs can be *used as* function optimizers [4].

The dilemma of current research is that even though the pertinent literature on EAs, e.g., [4, 10] and others, considers adaptation different from optimization, it does not provide any definition for adaptation or for the mentioned difference. Even though the literature on EAs does not provide a (working) definition, it seems common agreement that adaptation is concerned with time varying fitness landscapes as opposed to time invariant fitness functions typically used in optimization. This paper aims at formulating a *working hypothesis* for adaptation on the evolutionary time scale. To this end, Section 2 reviews some biologically inspired definitions, since adaptation has a long tradition in biology. It then proposes a working hypothesis from the point of view of technical applications.

Various practical tasks can be used to examine the adaptive behavior of different EAs. A common feature of most technical applications is that the object at hand is specified by a set of parameters $x_1, \dots, x_n$, with n denoting the number of parameters. In order to concentrate on the behavior of different EAs, several artificially designed test functions are used for experimentation, since complex, real-world tasks involve too many additional task-related problems. Furthermore, artificial test functions have the advantage that they can be fully understood and controlled. The experimental setup, including the used algorithms, test functions, and all parameter settings, is summarized in Section 3. Section 4 then presents the results of these basic experiments. Then Section 5 discusses related research [1], and finally, Section 6 provides a conclusion including a short discussion.

2 Adaptation: A Working Hypothesis

Adaptation has a long tradition in biology. In nature, each individual is forced to adapt to its environment. These adaptation forces originate from climate changes, limited food sources, behavioral changes of prey, the appearance of new predators, the competition among individuals of the same species, and so forth. According to McFarland [9], biologists usually distinguish between four different forms of adaptation: (1) evolutionary adaptation through genetic mechanisms in the very long term, (2) physiological adaptation to, for example, climate change or changes in food quality, (3) sensory adaptation to compensate changes in the strength of a particular stimulation, and (4) adaptation by learning to adapt

to a wide variety of different types of environmental changes. These four levels coincide with the explanation given in Merriam Webster's dictionary.

Even though the explanations given above provide a good categorization, they are not specific enough for a technical definition. Holland [8, p. 3] argues "*that adaptation, whatever its context, involves a progressive modification of some structure or structures. These structures constitute the grist of the adaptive process, being largely determined by the field of study. Careful observation of successive structural modifications generally reveals a basic set of structural modifiers or operators; repeated action of these operators yields the observed modification sequences.*" and later (p. 4), he argues that "*since a given structure performs differently in different environments – the structure is more or less fit – it is the adaptive plan's task to produce structures which perform "well" (are fit) in the environment confronting it.*" Holland's discussion proposes a close relationship between adaptation and fitness, but does not offer a conceptual difference between adaptation and plain fitness or function optimization.

A different perspective on adaptation is provided in [19, p. 105]: *Evolutionary Theory*: "*Adaptation must be defined as the origination and development of particular morphophysiological properties, whose importance to an organism is unambiguously associated with the various general or particular environmental conditions.*" and later (p. 118), it is said that "*Adaptation is the morphophysiological manifestation of the relationships between an organism and the environment recognized through their changes.*" In this view, it can be assumed that the species already exist and that their individuals have sufficient fitness to survive. Then, adaptation concerns an organism with respect to its particular environmental conditions. Accordingly, this paper proposes the following *working hypothesis* for adaptation on the evolutionary time scale.

Definition: Assume a population p that has converged to the neighborhood of the (global) optimum. Adaptation on the evolutionary time scale then refers to a fitness function in which the location of the optimum $\mathbf{x}^o$ is changing over time, i.e., all components of $\mathbf{x}^o$ are functions of the time $x_i^o = x_i^o(t)$. A population or an individual adapts to its environment, if it successfully tracks the changing location of the optimum.

By this definition, optimization and adaptation are two different tasks. The goal of optimization is to find the optimum, whereas the goal of adaptation is to track the changing location of that optimum. The definition of adaptation does not specify *why* the location of the optimum is changing.

3 Methods

This paper focuses on evolution strategies¹ and genetic algorithms. Therefore, this section first describes these algorithms in more detail and it then discusses further details of the experimental setup.

¹ For evolutionary programming, the same results can be expected as for evolution strategies, since both schemes differ, with respect to the scope of this paper, only in the self-adaptation scheme of the step-size (mutation strength) σ .

Both $(\mu \nmid \lambda)$ -ES selection schemes maintain a population of μ parents and generate λ offspring in each generation. A (μ, λ) -ES indicates that the μ new parents are selected only from the λ offspring, whereas in a $(\mu + \lambda)$ -ES, the μ parents are selected from the union of parents and offspring. For further details see [2]. The (μ, λ) -ES is currently recommended [2] for pure function optimization.

In contrast to traditional GAs, ESs encode each real-valued parameter as a floating-point number, and they apply mutation to *all* parameters simultaneously, i.e., $p_m = 1$. Mutations are typically implemented by adding $(0, \sigma)$ -normally distributed random numbers. A key feature of ESs is that they self-adapt the mutation strength σ . In its simplest form, each individual is enhanced by a step size σ , which is global with respect to the individual's parameters and local with respect to the population members. Each offspring inherits its step size from its parent(s). This step size is typically modified by taking the product of itself with a log-normally-distributed random number prior to mutation. By this means, the step size is self-adapted: those offspring survive that have the best adapted step size. For further details of different step size schemes see [2, 15]. In more elaborate forms, ESs maintain one step-size for each parameter or even maintain a whole $n \times n$ matrix with $n(n-1)/2$ independent parameters to generate correlated mutations. In all experiments reported in this paper, only one global step-size was used for each individual, and the step-sizes were always initialized with $\sigma = 0.001$.

As a representative for GAs, the breeder genetic algorithm (BGA) [11] was chosen for the following four reasons. First, the BGA is especially tailored to continuous function optimization. Second, the BGA yields superior results in the field of continuous function optimization [11, 13]. Third, the BGA encodes all real-valued parameters x_i as floating-point numbers, which has been shown to be more efficient than bit-coding schemes [14]. Fourth, the BGA uses standard parameter settings, such as a small mutation probability $p_m = 1/n$ and a high recombination probability $p_r = 0.5$, and uses an elitist (λ, μ) truncation selection scheme, which has been proven very efficient [17]; elitism means that the best individual obtained so far is preserved for the next generation. The BGA proposes the following mutation operator (see [11] for further details):

$$\delta = A \sum_{i=0}^{15} \alpha_i 2^{-i}, \quad \alpha_i \in \{0, 1\}, \quad (1)$$

where A specifies the mutation range, which might be reduced during evolution, and each α_i is set to the value 1 with probability $1/16$ and 0 with probability $15/16$. On average, the BGA's mutation operator picks only one non-zero α_j yielding $\delta = 2^{-j}$ for some j with $0 \leq j \leq 15$.

The main differences between ESs and GAs (including the BGA) are that GAs do not feature any self-adaptation of the mutation strength A and that GAs prefer small mutation rates in the order of $p_m = 1/n$, as typically recommended [2, 13, 16]. In essence, both algorithms differ in the mutation probability, i.e., $p_m = 1$ (ES) and $p_m = 1/n$ (BGA), and the self-adaptation/non-adaptation of the mutation strength. Other GA variants would use bit-string representations

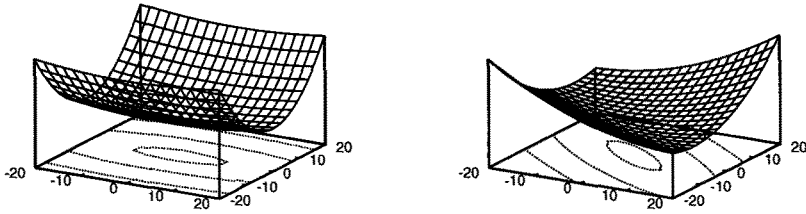


Fig. 1. 3D and contour plot of the function $f(x_1, x_2) = x_1^2 + 10x_2^2$ with (right figure) and without (left figure) a rotation of the coordinate system.

as opposed to floating-point numbers as a third difference to ESs, but bit-string representations yield worse performance under standard implementations [14].

Rather than using a complex real-world task, this paper concentrates on the following three widely-used test functions: the sphere $f_0(\mathbf{x}) = \sum_{i=1}^n x_i^2$, the quadratic function $f_1(\mathbf{x}) = \sum_{i=1}^n 2^i x_i^2$ with different eigenvalues 2^i along the coordinate axis x_i , and Rastrigin's function $f_6(\mathbf{x}) = \sum_i [x_i^2 - \cos(2\pi x_i) + 1]$. These three test functions seemed sufficient, since the task is not to discover the global optimum, but to track the location of the moving global optimum. Therefore, all individuals have been initialized around the global optimum with $(0, 0.1)$ -Gaussian-distributed noise.

It has been shown [13] that the efficacy of many GA experiments originates from the decomposability property $f(\mathbf{x}) = \sum_{i=1}^n f_i(x_i)$ of most of the widely-used test functions, which are only $O(n)$ hard, since all parameters x_i can be optimized independently of all others. For these test functions, GAs using floating point representations scale with $O(n \ln n)$ and GAs using bit coding representations scale with $O(n^k \ln n)$ with k denoting the number of bits per parameter x_i [13, 14]. Furthermore, it has been shown [13] that the performance of GAs progressively degrades, if a rotation of the coordinate system is applied. As can be seen from Figure 1, a rotation does not change the function's structure but its orientation in the search space. Therefore, this paper compares the adaptation performance on the original test functions $f(\mathbf{x})$ and the rotated versions $f(M\mathbf{x})$, where M denotes a pure rotation matrix.

In each generation, the location of the optimum, which is initially at $\mathbf{x}^o = \mathbf{0}$, is relocated by adding a speed quantity to each component $x_i^o \leftarrow x_i^o + s$ resulting in a total amount of change of $s\sqrt{n}$. The same amount s is used for all components x_i for the following reasons. First, changing some components not at all or just slightly, would mean that adaptation takes place only in a subspace. Second, changing only a few variables, would favor GAs too much, since small mutation rates $p_m = 1/n$ induce a strong search bias along the coordinate axes (see also [13]). Third, choosing a moving direction different from the $(1, 1, \dots, 1)^T$ would not lead to different results for the following two reasons. First, in the experiments not only the original test functions but also their rotated version have been used. Since the rotation angles have been uniformly chosen at random, the effective moving direction is arbitrary in the n -dimensional search space

with different effective moving speeds s_i along each coordinate axis of the rotated coordinate system. And as is discussed in Section 4, the rotation does not significantly effect the obtained performance. Second, quadratic functions with different Eigenvalues λ_i along each coordinate axis forces the population center toward a line $\lambda_1 x_1 = \lambda_2 x_2 = \dots \lambda_n x_n = c$ due to fitness selection (and not distance selection). Therefore, such functions are a challenge for ESs that use only one global step size σ .

It should be noted that the concept of tracking a moving optimum has an important consequence for the design of the GAs' mutation operator. Since GAs do not feature a self-adaptation of the mutation strength denoted by A in the BGA, the designer has to set the mutation strength to an appropriate value. In the experiments reported in Section 4, the mutation strength was set to $A = 100s$ in order to ensure small as well as large enough mutations.

In the experiments presented in the next section, the speed quantity s was varied between 0.001-10. It should also be mentioned that due to the relocation of the optimum in each generation, all individuals including the parents are re-evaluated. This unusual procedure is necessary, since it does not make sense to preserve an evaluated fitness value from one generation to another; the fitness function is changing as opposed to standard optimization tasks.

Current research on evolution strategies [2] recommends a (μ, λ) selection scheme for pure function optimization with a static fitness functions. The main advantage of choosing the new parents only from the offspring (non-elitism) is that it increases the chance of escaping from narrow, local optima due to noise or erroneous fitness evaluation. In this paper, however, a $(\mu + \lambda)$ -selection scheme has been chosen for the following reasons. First, it features elitism and therefore, makes a comparison with GAs easier, which also feature elitist selection schemes. Second, since all population members including all parents are evaluated in each generation, erroneous fitness evaluation is not a problem. Furthermore, due to the moving optimum, each population member will eventually be removed, since its fitness value starts steadily decreasing at a certain generation. In additional control experiments (see also Section 4) on the test functions described above, both schemes did not yield any significant difference.

4 Results

This section presents a summary of the most important results. All performance Figures 2 to 4 do not show any fitness values but the Euclidean distance of the population center to the current location of the moving optimum $\mathbf{x}^o(t)$. A lower limit of this performance measure would be zero, which could be obtained only if a population would know to which location the optimum is moving. In practice, this performance measure is greater than zero and depends on the moving speed s as well as the number of dimensions n . As should be obvious, increasing the number of dimensions n increases the difficulty of both the optimization and the adaptation task if the population size is kept constant as is done in this paper.

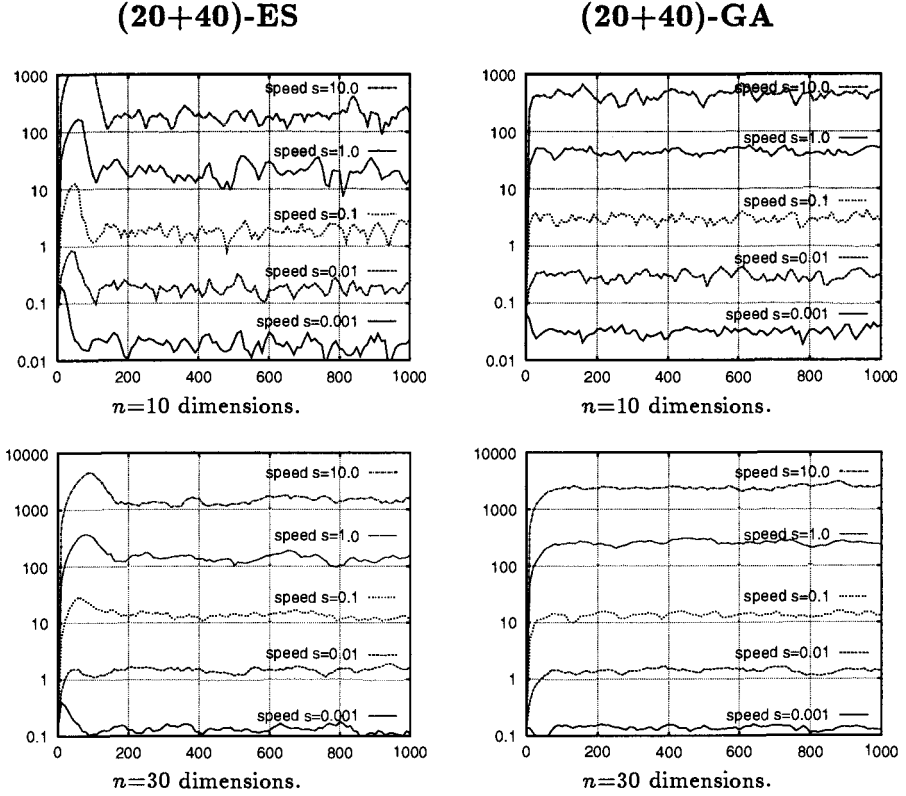


Fig. 2. Comparison of a (20+40)-ES (left column) and a (20+40)-GA (right column) when applied to the sphere $f_0(\mathbf{x}) = \sum x_i^2$ with $n = 10$ (upper row) and $n = 30$ (lower row) dimensions.

Each of the Figures 2 to 4 compares a (20+40)-ES (left column) with a (20+40)-GA (right column) when applied to functions f_0 , f_1 , and f_6 with dimensions varying between 3 and 30. From Figure 2, it can be seen that both procedures exhibit almost identical performance on f_0 . For function f_0 , no coordinate rotation has been applied, since f_0 is already rotationally invariant. It should be noted that in 30 dimensions, for example, the optimum travels a total distance of $s\sqrt{30}$ per generation with s denoting the speed constant.

Figure 3 compares both algorithms when applied to the quadratic function $f_1(\mathbf{x}) = \sum_{i=1}^n 2^i x_i^2$ with a rotation of the coordinate system. It can be seen that in this case, the ES adapts approximately half an order of magnitude closer to the moving optimum than the GA does. Compared to the rotated case, a (20+40)-GA exhibits slightly better performance when the rotation is not applied (not shown due to space limitations). From the last row of Figure 3, it can be seen that none of both algorithms is able to sufficiently track the optimum

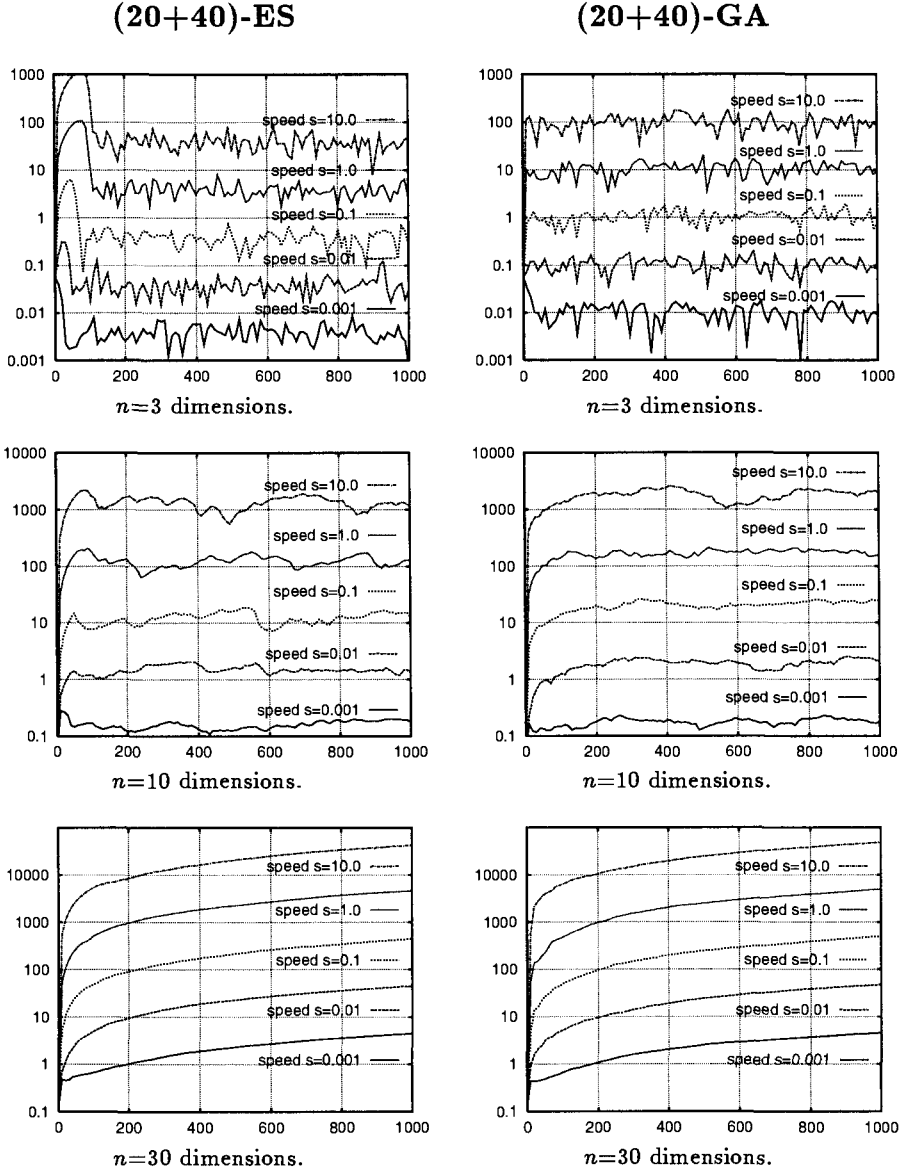


Fig. 3. Comparison of a (20+40)-ES (left column) and a (20+40)-GA (right column) when applied to the quadratic function $f_1(\mathbf{x}) = \sum_{i=1}^n 2^i x_i^2$ with $n = 3$ (upper row), $n = 10$ (middle row), and $n = 30$ (lower row) dimensions.

in 30 dimensions. It might be that due to the different eigenvalues 2^i along each parameter x_i , the population size is not sufficient for that many dimensions.

Figure 4 compares the tracking performance of both algorithms when ap-

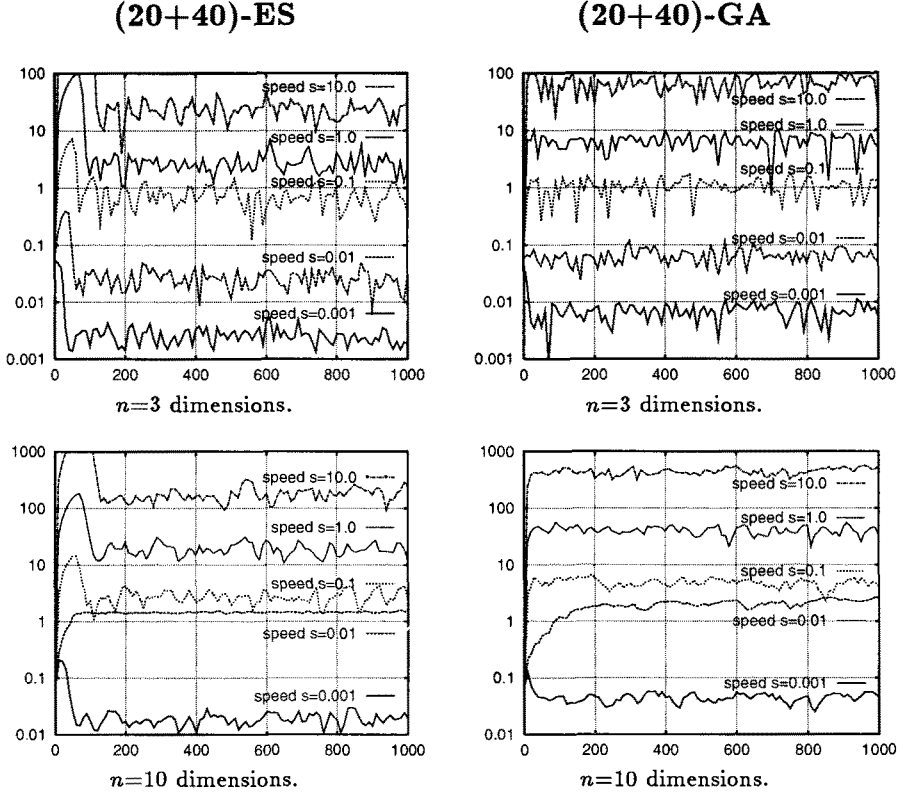


Fig. 4. Comparison of a (20+40)-ES (left column) and a (20+40)-GA (right column) when applied to Rastrigin's function $f_6(\mathbf{x}) = \sum_i [x_i^2 - \cos(2\pi x_i) + 1]$ with $n = 3$ (upper row) and $n = 10$ (lower row) dimensions.

plied to Rastrigin's function $f_6(\mathbf{x}) = \sum_i [x_i^2 - \cos(2\pi x_i) + 1]$. Again, it can be seen that the ES performs slightly better than the GA. The problem with this function is that the adaptation task turns into an optimization task for high speed values. If the moving speed is too high, the global optimum is moving far away from *all* population members from one generation to the next. Then, no population member is in the local basin of attraction of the global optimum, and the offspring have to find the new location of this basin of attraction.

Figures 2 to 4 show a peculiar behavior of the ES in the first few generation. In this time, the ES self-adapts the mutation strength σ to an optimal value that is in the order of the optimum's moving speed. This time span increases logarithmically with the speed due to the multiplicative correction of the step-size.

Figure 5 shows a comparison of a (μ, λ') and a $(\mu + \lambda)$ selection scheme when applied to the sphere with $n = 10$ dimensions and speed $s = 0.1$. In order to consider the same number of function evaluations in each generation, the number

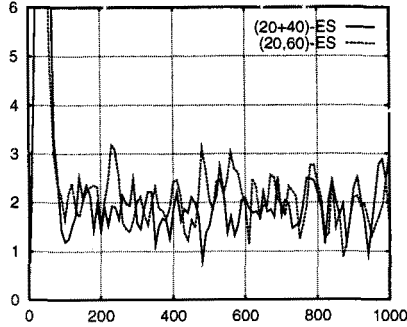


Fig. 5. Comparison of a (20+40)-ES and a (20,60)-ES when applied to the sphere with $n = 10$ dimensions and speed $s = 0.1$.

of offspring λ' of the (μ, λ') has to be set to $\lambda' = \mu + \lambda$. As can be clearly seen from this figure, both schemes yield the same statistical results.

Additional experiments have investigated the influence of recombination to the tracking behavior of both algorithms. In these experiments, both algorithms consistently yielded worse performance *with* recombination than without recombination. This result contradicts the common knowledge that recombination would be beneficial. In case of a moving optima, however, recombination cannot be beneficially exploited, since the combination of existing “good” solutions will not help in subsequent generations where the optimum is moving further.

5 Related Research

Similar research was independently done and has already been presented in [1]. Even though both papers ([1] and this paper) aim at a similar goal, the following three significant differences can be identified. First, the research presented in [1] aims at investigating the behavior of different adaptation schemes for the mutation strength in evolutionary programming. Accordingly, no other evolutionary algorithms were considered in [1]. By contrast, this paper focuses on investigating the behavior of different evolutionary algorithms, evolution strategies and genetic algorithms, with respect to their different forms of applying mutations, i.e., $p_m = 1$ and using a self-adaptation scheme for the mutation strength σ versus $p_m = 1/n$ without using any adaptation scheme for the mutation strength. Second, in [1] only the quadratic function with three parameters $x_{1 \leq i \leq 3}$ was investigated; no other test functions and no other search dimensions were considered. This paper considers three different test functions, the sphere, the quadratic function, and Rastrigin’s function, with many different properties, such as local optima and different Eigenvalues. Third, [1] has used quite different ways of moving the optimum. In his paper, the optimum was moved on a straight line, like in this paper, as well as in a circular and a chaotic way. The last two options have not been considered here, but as already discussed in Section 3,

they are covert to a large extent by considering a large variety of test functions. In addition, the results presented in [1] do not suggest a significant influence of the moving path to the resulting performance. Furthermore, has used three different update forms of the optimum's location; updating was done after 1, 5 or, 10 generations. As has been shown in [1], updating the optimum's location after 5 or 10 generations leads to a rough performance, especially for the investigated self-adaptation scheme. In [1], the self-adaptation scheme exhibits more problems, if in a circular trajectory, the update occurs after 5 or 10 generations, since the mutation strength is self-adapted for each particular parameter x_i ; these n mutation strengths have to be updated very frequently resulting in a "bumping" behavior. The update of a changing optimum after some generations has also been investigated in [18]. However, there the location of the optimum has been moved only after the entire population has been converged to the old optimum. Such task are, in the sense of this paper, rather an optimization task.

The results presented in [1] coincide with the results presented in this paper. As has been shown in the previous section, the self-adaptive mutation scheme of the ESs requires some initial time to adjust the mutation strength to the appropriate value. With this respect, GAs yield smoother results, since the appropriate mutation strength is given by the designer or by a designer-chosen heuristic. In conclusion, the research presented in [1] is complementary to the research presented in this paper.

6 Conclusions

This paper has proposed that adaptation should refer to tracking a moving optimum in dynamically changing fitness functions as opposed to function optimization as an optimum-finding task. The experiments presented in this paper have compared the tracking performance of the evolution strategies and genetic algorithms. It can be summarized that the ES performs slightly better than the GA and that in *sharp contrast* to pure function optimization (see also [13]), the application of a coordinate rotation has almost no effect on the tracking performance of both algorithms. Future research will be devoted to the investigation of different population sizes and different Eigenvalues for each coordinate axis.

From the practical point of view, ESs have the great advantage of self-adapting the mutation strength, which takes place whenever it is required. When using a GA by contrast, the designer has to find a reasonable estimate of the moving speed in order to determine the mutation strength, which is denoted by A in the BGA. This problem is not that severe, since the BGA's mutation operator yields robust behavior within a few orders of magnitude (see also [11]).

As a second, more general result, it should be noted that the particular implementation and parameter setting has much less influence on the tracking behavior than it would have on the optimization behavior. It seems essential that an algorithm provides a sufficient amount of variation and that it applies a suitable selection scheme.

References

1. Angeline, P., 1997, Tracking Extrema in Dynamic Environments, in: *Proceedings of the Sixth Annual Conference on Evolutionary Programming VI*, P.J. Angeline, R.G. Reynolds, J.R. McDonnell, and R. Eberhart (eds.) (Springer-Verlag), pp. 335-345.
2. Bäck, T., Schwefel, H.-P., 1993, An Overview of Evolutionary Algorithms for Parameter Optimization, *Evolutionary Computation* 1(1):1-23, 1993.
3. De Jong, K.A., 1975, An Analysis of the Behavior of a Class of Genetic Adaptive Systems, Ph.D. Thesis (University of Michigan).
4. De Jong, K.A., 1993, Genetic Algorithms are NOT Function Optimizers, in: *Foundations of Genetic Algorithms 2*, L.D. Whitley (ed.) (Morgan Kaufmann Publishers), pp. 5-17.
5. Fogel, D.B., 1995, *Evolutionary Computation: Toward a New Philosophy of Machine Learning Intelligence*, (IEEE Press).
6. Fogel, L.J., 1962, "Autonomous Automata", *Industrial Research*, 4:14-19.
7. Goldberg, D.E., 1989, *Genetic Algorithms in Search, Optimization and Machine Learning*, (Addison-Wesley).
8. Holland, J., 1992, *Adaptation in Natural and Artificial Systems: an Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, (MIT Press).
9. McFarland, D., 1990, What it means for Robot Behaviour to be Adaptive, in: *From Animals to Animats, Proceedings of the First International Conference on Simulation of Adaptive Behavior*, J.-A. Meyer and S.W. Wilson (eds.) (MIT Press), pp. 22-28.
10. Menczer, F., Belew, R.K., Willuhn, W., 1995, Artificial life applied to adaptive information agents, in: *AAAI Spring Symposium Series: Information gathering from heterogeneous, distributed environments*. C. Knowlton, A. Levy, S.-S. Chen, and G. Wiederhold (eds.) (AAAI).
11. Mühlenbein, H., Schlierkamp-Voosen, D., 1993, Predictive Models for the Breeder Genetic Algorithm I. *Evolutionary Computation* 1(1):25-50.
12. Rechenberg, I., 1973, *Evolutionstrategie*. (Frommann-Holzboog).
13. Salomon, R., 1996, Reevaluating Genetic Algorithm Performance under Coordinate Rotation of Benchmark Functions, *BioSystems* 39(3):263-278.
14. Salomon, R., 1996, The Influence of Different Coding Schemes on the Computational Complexity of Genetic Algorithms in Function Optimization, in: *Proceedings of The Fourth International Conference on Parallel Problem Solving from Nature (PPSN IV)*, H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel (eds.) (Springer-Verlag), pp. 227-235.
15. Schwefel, H.P., 1995, *Evolution and Optimum Seeking* (John Wiley and Sons).
16. Srinivas, M., Patnaik, L., 1994, Genetic Algorithms: A Survey, *IEEE Computer*, pp. 17-26.
17. Thierens, D., Goldberg, D., 1994, Convergence Models of Genetic Algorithm Selection Schemes, in: *Proceedings of Parallel Problem Solving from Nature 3*, Y. Davidor, H.P. Schwefel, and R. Männer (eds.) (Springer-Verlag), pp. 119-129.
18. Vavak, F., Fogarty T.C., 1996, A Comparative Study of Steady State and Generational Genetic Algorithms, in: *Evolutionary Computing: AISB Workshop*, LNCS 1143, T.C. Fogarty (ed.) (Springer Verlag), pp. 297-304.
19. Volkenstein, M.V., 1994, *Physical Approaches to Biological Evolution* (Springer-Verlag).

The Frequency Assignment Problem: A Look at the Performance of Evolutionary Search

Christine Crisan^{*} and Heinz Mühlenbein^{**}

GMD - German National Research Center for Information Technology
D - 53754 Sankt Augustin
Germany

Abstract. The performance of evolutionary search on the Frequency Assignment Problem is the topic of ongoing research. In this paper the performed evaluation is done in two steps. First, the fitness landscape to be optimized is analyzed. Secondly, the performance of different search procedures is described. The insight gained from the fitness landscape analysis is related to the performance of the different search procedures.

1 Introduction

With the perennial increase in the demand of mobile telephone services and the restricted frequency spectrum dedicated to these services, the problem of optimal usage of frequencies has become a major step in planning and operating a digital *mobile telecommunication network (MTN)*. A *MTN* divides the area to be served into cells. A cell is defined to be the geographical area around a *base station (BS)*, where the field strength received from the antenna of the corresponding *BS* is high enough to guarantee a good communication. Each base station has to be assigned a number of frequencies on which the wireless communication occurs [1]. The number of frequencies to be assigned to each base station is approximately defined by the number of users to be served in that area. This leads to an inhomogeneous distribution of frequency requirements. The assignment of frequencies to base stations must satisfy several electromagnetic compatibility constraints such as the *co-channel*, *adjacent-channel* and *co-site* constraint. The first two constraints refer to identical or, regarding the frequency spectrum, adjacent frequencies which may be used only by base stations which cannot interfere with one another. The *co-site* constraint implies that frequencies used at a certain base station must be at a predefined minimum distance with regard to the frequency spectrum.

The union of the frequency requirements from each cell can be viewed as a joint set of frequency requirements. Depending on which cell a requirement belongs to, the assignment of a frequency for that requirement has to satisfy different constraints. The *Frequency Assignment Problem (FAP)* as described in this paper consists of assigning a frequency to each requirement subject to

^{*} email: Christine.Crisan@gmd.de

^{**} email: Heinz.Muehlenbein@gmd.de

a set of constraints. Simultaneously the total number of different frequencies used by an assignment has to be minimized. The *FAP* can be formulated as a discrete optimization problem. It is a generalization of the graph coloring problem, hence NP-hard. Note that the two demands, finding frequency assignments which satisfy the electromagnetic constraints on the one hand and of frequency assignments which make use of as few as possible frequencies on the other hand, are orthogonal, which makes the problem even harder.

Modern digital cellular networks consist of more than 4000 base stations with a requirement of at least two frequencies per base station, which results in optimization problems of dimensions greater than 8000. Therefore, efficient optimization procedures have to be developed in order to find high quality frequency assignments. Several heuristic optimization methods for different instances of the *FAP* have already been proposed [2, 3, 4].

In this paper we take a bottom up approach to the problem in order to favor the design of efficient search procedures. This means that we take a look at the search space and then look at the performance of several search operators within an evolutionary search environment. In this paper we restrict the analysis of the search space to the consideration of the density of states, which is a measure of the hardness of an optimization problem [6] and take a look at the distribution of locally optimal solutions.

The paper is structured as follows: first we give a brief description of the *FAP*, then we analyze the search space. Finally we present several search operators and relate the obtained results to the knowledge gained about the search space.

2 Frequency Assignment Problems

There are many ways to define different instances of the *FAP* [5]. The special case of the *FAP* described in this paper can be described as follows.

Given are a set of requesters $R = \{r_i\}_{i=1,\dots,n}$ and a set $F = \{f_i\}_{i=1,\dots,m}$ of frequencies. For each requester r_i , $i = 1, \dots, n$ a set $D_i \subseteq F$ of frequencies which may be assigned to the respective request is given. For each pair of requesters (r_i, r_j) there is a minimum required distance given. This defines the minimal difference between frequencies which will be assigned to the requesters r_i and r_j . For each pair of requesters (r_{2k-1}, r_{2k}) , $k = 1, \dots, n/2$ which we will refer to as *parallel links*, this distance has to be exactly $d = 238$. This is the so called *equality constraint*.

Required is an assignment of frequencies to the requesters so that each requester is assigned a valid frequency and no interference constraint, i.e. minimal distance, is violated. Furthermore, the total number of different frequencies used in an assignment should be minimal. The two requirements, minimizing the interference as well as the number of frequencies used are orthogonal, which makes the problem even harder.

The *FAP* can also be described with help of an undirected weighted graph $G = (V, E, w)$, with V a set of vertices, $E \subseteq V \times V$ a set of edges and $w : E \longrightarrow \mathbb{R}$

Table 1. Description of the considered problem instances

Instance	n	# Constr.	ED	Min/Max
SCEN01	916	5548	0.0132	16/48
SCEN02	200	1235	0.0620	14/44
SCEN03	400	2760	0.0345	14/48
GRAPH01	200	1135	0.0570	18/44
GRAPH02	400	2245	0.0281	14/44
GRAPH03	200	1086	0.0545	?/44

a weighting function. For each requester r_i there is a vertex $v_i \in V$ and for each constraint (r_i, r_j) there is an edge $(v_i, v_j) \in E$. The function w assigns to each edge the minimum distance imposed by the corresponding constraint.

The considered instances are presented in Table 1. For each problem the dimension n , the total number of constraints (# Constr.), the edge density (ED) of the corresponding graph and the number of frequencies used in the best known solutions (Min) versus the total number of available frequencies (Max) are given. For the problem instance GRAPH03 the optimum is still unknown.

The *FAP* instances SCEN01, SCEN02, and SCEN03 are real-life instances which were provided by the Centre d'Electronique de l'Armement, France within the CALMA project on *Combinatorial Algorithms for Military Applications*. The instances GRAPH01, GRAPH02 and GRAPH03 were made available by the research group at the Delft University of Technology which was involved in the CALMA project. Although they were randomly generated, they have the same characteristics as the real-life instances. For a more detailed description of these instances see [9, 12].

3 Search Space Analysis

There have been many attempts to find measures which characterize a fitness landscape and give an indication about the difficulty of the problem [6, 7, 8].

In this paper we investigate two measures, the density of states and the distribution of locally optimal solutions.

First we describe the coding used, define a cost function and a neighborhood structure on the search space and show the induced metric.

3.1 The Search Space

Coding We are coding the *FAP* as an n -dimensional integer vector. A value f_i of the i th component means that frequency f_i is assigned to the request r_i , $i = 1, \dots, n$. Since for all considered instances the constraint $|f_{2k-1} - f_{2k}| = 238$, $k = 1, \dots, n/2$, has to hold, we define the domain of definition to consist only of solutions which already fulfill this requirement. The range of valid frequencies is the same for each parallel link. Since for each link there are exactly $2d_{2k}$

different assignments, where d_{2k} is the number of different possible frequencies for the requests $2k - 1$ and $2k$, the dimension of the resulting search space is $2^{n/2} \prod_{i=1}^{n/2} d_{2i}$. This coding, for which solutions violating the equality constraint are not considered, reduces the search space drastically and as we have found out leads to a faster convergence. The dimension of the search space which also contains solutions which do not satisfy the equality constraint is $\prod_{i=1}^{n/2} d_{2i}^2$.

Cost Function For a solution, i.e. frequency assignment, s , the fitness function is defined as follows

$$\mathcal{F}(s) = w_1 \cdot NOV(s) + w_2 \cdot NOF(s) + penalty(s) \quad (1)$$

with $NOV(s)$ and $NOF(s)$ the number of violated constraints and the number of different frequencies used by the solution. The penalty function favors solutions which assign some of the frequencies seldom. It is defined as follows

$$penalty(s) = \sum_{i=1}^m \sqrt{UF(f_i)} \quad (2)$$

with $UF(f_i)$ the number of times the frequency f_i is used within the solution.

Neighborhood Next we define a neighborhood structure on the search space induced by the coding. We define two solutions $s = (f_1, \dots, f_n)$ and $s' = (f'_1, \dots, f'_n)$ to be neighboring to each other *iff* they differ in exactly one parallel link assignment, i.e.

$$\exists! k \in (1, n/2) : f_{2k-1} \neq f'_{2k-1} \wedge f_{2k} \neq f'_{2k} \quad (3)$$

$$\forall j \neq k \quad f_j = f'_j \quad (4)$$

We define the neighborhood of a solution s as the set of all its neighboring solutions s' .

Metrics on the search space The coding used and the defined neighborhood induce a metric defined by the normalized Hamming distance between two solutions. The normalized Hamming distance between two solutions $s = (f_1, \dots, f_n)$ and $s' = (f'_1, \dots, f'_n)$ is defined as the total number of different assignments divided by the maximum number of different possible assignments.

$$h_d(s, s') := \frac{1}{n} \sum_{i=1}^n sgn(|f_i - f'_i|) \quad (5)$$

where sgn is the signum function. Notice that the normalized Hamming distance is always even, since only solutions which fulfill the equality constraint are considered.

3.2 Density of states

The density of states as introduced in [6, 7] is defined by the number of different solutions with a certain fitness value. Knowledge about the density of states helps with the estimation of the fitness value for the problem optimum and gives a measure of the difficulty for finding the global optimum. The capability of estimating the fitness value of the optimum with the help of the density of states concept arises from the fact that an approximation of the lowest fitness value (in case of a minimization) is given by the states for which the density is zero. Within the optimization process, the probability of still finding a better solution decreases at the same rate as the values of the density of states decrease. A rapid decrease of the density of states values points to a difficult optimization problem.

The density of states has to be estimated since the extent of the search space makes an exact computation impossible. For obtaining an approximation of the density of states we used a Metropolis algorithm. Starting with a random solution s , a new solution s' from its neighborhood is obtained, i.e. the assignment of exactly one parallel link is changed. If the fitness value of the new solution is better, s' is the new solution, otherwise an inferior solution is accepted with probability $e^{-\frac{\Delta\mathcal{F}}{T}}$, where $\Delta\mathcal{F}$ is the change in the fitness value. In this manner the algorithm converges to a Boltzmann distribution from which the density of states can be approximated [7]. So, in equilibrium the density of states observes the Boltzmann distribution

$$P_{eq}(\mathcal{F}) \sim n(\mathcal{F}) \cdot e^{-\frac{\mathcal{F}}{T}} \quad (6)$$

where $P_{eq}(\mathcal{F})$ is the equilibrium density, $\mathcal{F}$ is the energy function, T the temperature and $n(\mathcal{F})$ the density of states with the energy $\mathcal{F}$ [7].

Figure 1 shows the estimated values for the density of states for the instances SCEN01 and SCEN02. For SCEN02 there are a lot of locally optimal solutions near the global optimum. In the vicinity of the optimum the density of states decreases. The Metropolis algorithm detected configurations with a fitness value as low as 16. It estimates that there are about 1800 configurations with this fitness value. The fitness value of the global optimum is 14. For SCEN01 the distribution is different. The smallest fitness value found by the Metropolis algorithm was 22. It is estimated that there are about 2000 configurations with this fitness value. The global optimum has the fitness value 16. This shows that SCEN01 is more difficult to solve, but both problem instances are rather easy optimization problems.

3.3 Hamming Distance

Next we studied the similarity between locally optimal solutions. On the basis of the previously defined normalized Hamming distance we compute the normalized Hamming distance of a solution s with respect to other M considered solutions

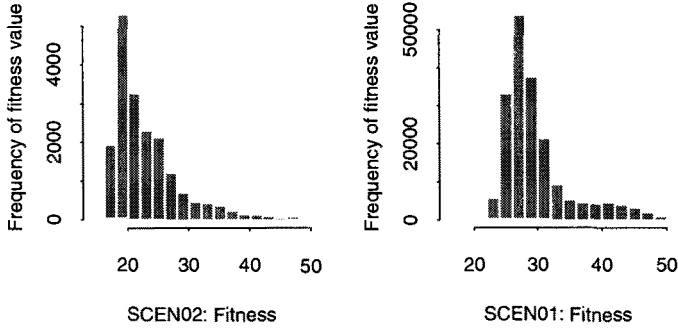


Fig. 1. Density of states

$s_1, \dots, s_M$ [11] as follows

$$h_d(s) = \frac{1}{M} \sum_{i=1}^M h_d(s, s_i). \quad (7)$$

Figure 2 shows the normalized Hamming distance of locally optimal solutions for the problems SCEN01 and SCEN02. For SCEN02 we computed 1000 global optimal solutions. For SCEN01 we computed 100 locally optimal solutions. The values of the Hamming distance are almost maximal, i.e. the solutions are very different. Even if we restrict the analysis to the best 20% of the locally optimal solutions, there is not much similarity. In fact, for SCEN01 the normalized Hamming distance of the best local optimum to the 20% best locally optimal solutions is about 0.87.

This result is very different from a corresponding search space analysis for the Traveling Salesman Problem [11]. There the normalized Hamming distance was measured as about 0.25. This means that on average two good solutions have 75% elements in common.

The analysis of the search space shows that different instances of the *FAP* can be very different with respect to the density of states. The Hamming distance space measure shows that local optima are randomly distributed in the search space. In this case the solutions do not have building blocks, so that recombination can be an ineffective search operator. This result has been confirmed by simulations. So we decided to look first at good mutation operators.

4 Comparison of Search Operators

We used an iterated (1+1) evolutionary search algorithm in order to analyze different search operators. The algorithm is shown in Figure 3. The algorithm uses just one parent, which creates an offspring. If the offspring is better than the parent, it replaces the parent. If the algorithm gets stuck in a local optimum,

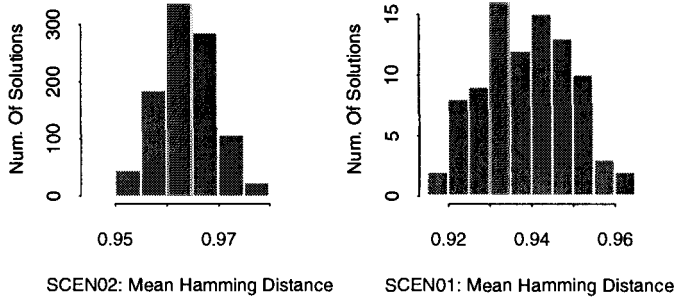


Fig. 2. Hamming Distance

it starts the search from a slightly modified point (**modify**(s)). The evaluation function is the one defined in Equation 1 with $w_1 = w_2 = 1$.

4.1 Operators

Different operators were implemented. **Random Mutation** (RM), which randomly chooses a variable for mutation and also randomly changes its assignment to a new frequency from the domain of valid frequencies. **Interference Mutation** (IM), which randomly chooses a variable for mutation, whose assignment violates at least one constraint and assigns a new frequency to it. Furthermore an **Extended Interference Mutation** (EIM) was implemented. EIM builds up a candidate list of variables which, by changing their assignment, might lead to a better solution. One variable from the candidate list is randomly chosen for mutation and assigned a new frequency. The candidate list contains the variables which violate at least one constraint and all other variables which are involved with the first variables in a constraint. Finally we developed an **Extended Interference and Frequency Mutation** (EIFM) which is similar to EIM, but the new assigned frequency is chosen among the frequencies used most often in the solution. For all defined operators, the following holds. After changing the assignment of a variable r_i , the assignment of the variable belonging to the corresponding parallel link, which is r_{i+1} or r_{i-1} , is also changed. This means that the equality constraint always holds.

All these operators perform only minimal changes on a solution, transforming it into a new solution from its neighborhood. Let $\mathcal{N}_{RM}$, $\mathcal{N}_{IM}$, $\mathcal{N}_{EIM}$ and $\mathcal{N}_{EIFM}$ be the neighborhood functions of the defined operators and $\mathcal{N}_{RM}(s)$, $\mathcal{N}_{IM}(s)$, $\mathcal{N}_{EIM}(s)$ and $\mathcal{N}_{EIFM}(s)$ their corresponding neighborhoods for a given solution

```

begin
  ITER = 0;
   $s \leftarrow \text{generate\_initial\_solution}$ 
   $fit_s \leftarrow \text{evaluate}(s)$ 
  iterate while (ITER < MAX_ITER)
    ITER++;  $num\_iterations \leftarrow 0$ ;
    modify( $s$ )
    evolutionary_search
      if ( $num\_iterations > maximum\_iterations$ )
        goto iterate
       $num\_iterations++$ 
       $s' \leftarrow \text{mutate}(s)$ 
       $fit_{s'} \leftarrow \text{evaluate}(s')$ 
      if ( $fit_{s'} < fit_s$ )
         $s \leftarrow s'$ 
      if ( $terminate\_condition$ )
         $best\_solution \leftarrow s$ 
        goto end
      else
        goto evolutionary_search
      end_evolutionary_search
    end

```

Fig. 3. Iterated Evolutionary Search Algorithm

s . Then

$$\mathcal{N}_{IM}(s) \subseteq \mathcal{N}_{EIM}(s) \subseteq \mathcal{N}_{EIFM}(s) \subseteq \mathcal{N}_{RM}(s) \quad (8)$$

All the neighborhood functions are connected, i.e. from any starting solution s an optimal solution can be reached by applying the corresponding neighborhood function. Thus, the operators may be compared with one another.

4.2 Results

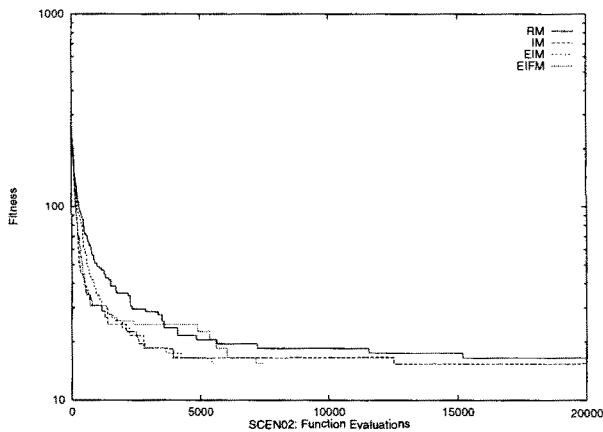
Table 2 shows the obtained results. For each operator 30 runs were performed. FE gives the mean number of function evaluation and $Best$ gives the number of utilized frequencies and the amount of interference in the best found solution. The fact that by using just RM, fairly good solutions could be obtained shows once again that a large number of locally optimal solutions exist. Nevertheless, very good solutions which are contained in the area of the search space just after the curve of the density of states decreases can not be obtained with the RM

Table 2. Obtained results

Instance	RM		EIM		EIFM		RM+1PC	
	<i>FE</i>	<i>Best</i>	<i>FE</i>	<i>Best</i>	<i>FE</i>	<i>Best</i>	<i>FE</i>	<i>Best</i>
SCEN01	1.000E+06	18/6	1.000E+06	20/1	1.000E+06	16/2	1.000E+06	18/4
SCEN02	5.935E+04	14/0	5.289E+04	14/0	1.187E+04	14/0	3.538E+05	14/0
SCEN03	1.000E+06	16/1	1.000E+06	16/0	9.866E+05	14/0	1.000E+06	16/1
GRAPH01	4.674E+05	18/0	8.574E+04	18/0	4.021E+05	18/0	1.000E+06	22/0
GRAPH02	1.000E+06	16/0	1.000E+06	14/1	9.901E+05	14/0	1.000E+06	16/0
GRAPH03	1.000E+06	30/33	1.000E+06	30/37	1.000E+06	28/38	1.000E+06	28/39

operator. IM was more efficient, but it also fails to find the optimum. EIM finds the optimum just for the problem SCEN02, which seems to be the easiest of all the problems. The best results by far can be obtained by using the EIFM operator. The operator just tunnels its way to the attractor of a good optimum by simultaneously minimizing the number of violated constraints and reducing the number of different utilized frequencies by a solution.

We also conducted experiments with a larger population size, using RM and IM combined with one of the well known crossover operators: 1 Point, 2 Point and Uniform Crossover and an elitist selection procedure. The experiments were conducted in order to gain first insights in the power of crossover for the *FAP*. Note that for the runs where Crossover was used, much more function evaluations were necessary.

**Fig. 4.** SCEN02: Fitness Progress for different Mutation operators

Figures 4 and 5 show the typical progress of the fitness (i.e. cost) of the best

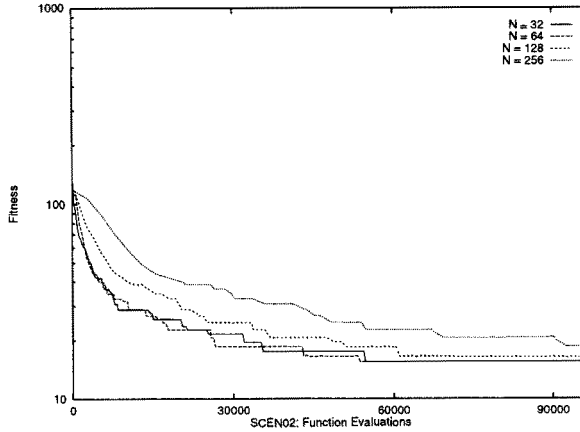


Fig. 5. SCEN02: Fitness progress for RM and 1Point Crossover

solution within a run for SCEN02. In Figure 4 different mutation operators are shown, while in Figure 5 a combination of mutation and crossover for different population sizes N is shown.

The usage of crossover makes the optimization more robust as it explores the search space more thoroughly. On its way to the optimum, local optima of the form (NOF, NOV) (see Equation 1) with $NOV = 0$ and decreasing NOF are reached. The fitness value decreases more slowly compared to the decrease when using just mutation.

The results obtained with the iterated $(1+1)$ algorithm with the EIFM operator are superior to those reported in [10], where results obtained with a standard and a meta genetic algorithms are presented. In [9] results obtained with Tabu Search are reported. There the authors only give the time needed in order to find a solution, rather than the number of function evaluations. Nevertheless, EIFM seems to find the solution more rapidly than Tabu Search, which suggests that the memory used by the Tabu Search is not necessary for these instances of the Frequency Assignment Problem.

5 Conclusions and Future Work

Evolutionary Search proves to be a powerful optimization method for solving the *FAP*. Slightly modifying a solution trapped in a local optimum and restarting the search from that point makes the search more robust. In the parts of the search space where many locally optimal solutions exist crossover shows to be a useful operator but for the *FAP* instances investigated in this paper an algorithm using only a problem specific mutation performed best.

The problem instances defined in [12] proved to be easy for our algorithm. We are now working on real world problems defined by the German mobile

telecommunication provider TMobil. These *FAP* problems have a more application oriented fitness function and the size of the problems is much larger. For Germany up to 6000 base stations have to be considered, which results in optimization problems of dimensions greater than 12000. Due to a non disclosure agreement we are currently not allowed to publish any results pertaining these data.

References

1. Balston, D.M., Macario, R. C. V. (Eds): Cellular Radio Systems. Artech House (1993)
2. Crompton, W., Hurley, S., Stephens, N.M.: A Parallel Genetic Algorithm for Frequency Assignment Problems. Proc. of IMACS SPRANN. (1994) 81–84.
3. Dorne, R., Hao, J.-K.: An Evolutionary Approach for Frequency Assignment in Cellular Radio Networks. Proc. of IEEE Intl. Conf. on Evolutionary Computation. (1995) 539–544.
4. Hao, J.-K., Dorne, R.: Study of Genetic Search for the Frequency Assignment Problem. Artificial Evolution, LNCS 1063. (1996) 333–344.
5. Hale, W.H.: Frequency Assignment: Theory and Application. Proc. of the IEEE, Vol. 68, No. 12 (1980) 1498–1573.
6. Rose, H., Ebeling, W., Asselmeyer, T.: The Density of States - a Measure of the Difficulty of Optimisation Problems. Parallel Problem Solving from Nature - PPSN IV. (1996) 208–217.
7. Asselmeyer, T., Ebeling, W., Rose, H.: Smoothing representation of fitness landscapes - the genotype-phenotype map of evolution. <http://summa.physik.hu-berlin.de/~torsten/>
8. Jones, T., Forrest, S.: Fitness Distance Correlation as a Measure of Problem Difficulty for Genetic Algorithms. Proceedings of the 6.th ICGA. (1995) 184–192.
9. Bouju, A. et al.: Tabu search for radio links frequency assignment problem. Applied Decision Technologies, London. UNICOM Conf. (1995).
10. Smith, G.D. et al.: EUCLID CALMA Radio Link Frequency Assignment Project; Technical Annexe T- 2.1:A - Supplementary Report.
11. Mühlenbein H.: Evolution in Time and Space - The Parallel Genetic Algorithm. Foundations of Genetic Algorithms. (1991) 316–337.
12. Description of the CALMA Benchmark Problems. <http://dutiosd.twi.tudelft.nl/~rlfap/>

A Critical and Empirical Study of Epistasis Measures for Predicting GA Performances: A Summary

S. Rochet¹ G. Venturini² M. Slimane² E.M. El Kharoubi²

¹ IUT,
Dép. Génie Electrique,
150, Avenue du Maréchal Leclerc,
13300 Salon de Provence, France.

² Laboratoire d'informatique,
Université de Tours,
64, Avenue Jean Portalis,
37200 Tours, France.
venturini,slimane@univ-tours.fr
Tel: (+33)-2-47-36-14-14,
Fax: (+33)-2-47-36-14-22

Abstract. Epistasis measures have been developed for measuring statistical information about the difficulty of a function to be optimized by a genetic algorithm (GA). We give first a review of the work on these measures such as the epistasis correlation. Then we try to relate the epistasis correlation to the overall performances of a binary GA on a set of 14 functions. The only relation that seems to hold strongly is that a high epistasis correlation implies GA-easy, as indicated by the GA theory of deceptiveness. Then, we show that changing the representation of the search space with transformations that improve epistasis measures does not imply the same increasing in the genetic algorithm performances. These both empirical studies seem to indicate that the generality of epistasis measures is limited.

1 Introduction

Characterizing the difficulty of functions is a key point of the genetic algorithms (GAs) theory (see a survey in (Venturini et al. 1997)). The theory of deceptiveness was probably the first one to be studied (Goldberg 1987) (Whitley 1991). It has, however, several limitations (Grefenstette 1992) like for instance dealing only with static fitness computed on the whole search space rather than dynamic fitness computed on the population. Recently, several statistical measures have been developed for characterizing the difficulty of functions for GAs, and thus for predicting the performances of the GA. The fitness/distance correlation evaluates the correlation that may exist between the distance of individuals to the closest global optimum of the fitness function f and their fitness (Jones and Forrest 1995). This measure however has the drawback of requiring the knowledge of f global optima, which is of course not realizable in practice, or not

realistic because in this case one does not need any search method to find the optimum. Other statistical measures are for instance the operator correlation (Manderick et al. 1991) which can be used in virtual GAs (Grefenstette 1995). This measure evaluates for instance the correlation that may exist between the fitness of parents and their offspring with various operators. Such operators can thus be selected according to this correlation. Finally, epistasis measures (Davidor 1991) approximate the fitness function with another function computed by considering the fitness of schemata independently from each others. This bit by bit approximation measures the dependence that may exist between bits in the representation.

In this paper, we describe with more details epistasis measures and we concentrate on the use of such measures for predicting the performances of the GA. Epistasis measures have the advantage of being easily computable from a set of points generated randomly in the search space. For this reason, they seem to be more interesting than other measures such as fitness/distance correlation. We aim here at answering one main question about epistasis measures, a question which as far as we know has not been answered yet: are these measures useful for predicting the performances of the GA in practice? We provide at least two empirical answers to this question. The first one consists in computing the epistasis of several standard functions and in running the GA on these functions. One can then try to relate the epistasis with the observed performances of the GA. As will be seen, this relation is not strong for the tested functions. Then, the second answer is provided by using epistasis measures in order to guide the search for a better binary representation that could improve the GA performances. Our tests suggest that the epistasis measure of a sample of points can be improved but that the GA performances remain at the same level.

The paper is organized as follows: section 2 provides an introduction to epistasis measures. Section 3 describes our first tests where epistasis measures are compared to GA performances on a set of 14 binary functions. Section 4 introduces the basic techniques for changing the representation in a binary GA and provides experimental results. Section 5 concludes on the usefulness of epistasis measures.

2 Overview of epistasis measures

From initial work on epistasis measures for GAs (Davidor 1991), one may define the variance of epistasis as follows: let X denote the fitness function that goes from $S = \{0, 1\}^l$ to R . To compute the epistasis variance, one must generate a set S_a of binary strings $s_1, \dots, s_n$ and evaluate the fitness of those strings $X(s_1), \dots, X(s_n)$. Then one must compute a function W that approximates X by using the information contained in the sample of n strings. A binary string of length l can be denoted by $s = b_1, b_2, \dots, b_l$ where $b_i \in \{0, 1\}$. The value of W for this string equals:

$$W(s) = [X(b_1 * \dots *) - X(* \dots *)] + [X(* b_2 * \dots *) - X(* \dots *)] + \dots$$

$$+[X(*... * b_l - X(*...*)) + X(*...*)]$$

where $X(*... * b_i * ...*)$ denotes the fitness of schema $*... * b_i * ...*$ computed on strings of S_a , i.e. the mean fitness of strings in S_a that belong to the schema. W is thus computed by assuming that the fitness function X is a simple function that is separable, i.e. fitness is assigned to bits independently. Of course, this is almost always not the case in practice. So, the difference or similarity between W and X can be viewed as respectively a complexity or simplicity measure of X . Davidor has introduced the variance of epistasis as a measure of difficulty of X . This consists simply in computing $W(s_1), ..., W(s_n)$ and then in computing the variance of $(W(s_i) - X(s_i))_{i \in [1..n]}$. This measure has, among other things, the drawback of being sensitive to the scale of X .

The epistasis variance can be normalized by the function variance on S_a (Manela and Campbell 1992), but the use of a correlation allows an even better comparison of the epistasis measures between two functions. Recently, Rochet has introduced such a measure of simplicity, based on a correlation, which has the advantage of being comparable from one function X to another and is always positive. It consists in computing the correlation between W and X . This epistasis measure has been extensively studied in (Rochet 1997). We have shown formally that epistasis measures are in fact error measures between the best linear approximation of X and X itself. Therefore such measures can be generalized to continuous representation as in (Rochet et al. 1996).

Other work has been done dealing with epistasis measures like theoretical studies (Reeves and Wright 1995). One should expect in practice that such a measure evaluates the difficulty of a function X for the GA: this comes from the fact that epistasis measures are measures of X linearity, and that the GA performs well on linear functions such as GA-easy functions (Wilson 1991). On such linear functions, the role of recombination is then optimal because the recombination of low order building block with high fitness leads to even better individuals. This is the hypothesis that we try to test in the following, from a practical point of view which should allow direct applications to real-world problems.

3 Relation to GA performances

3.1 Experimental settings

In this paper, we have used 14 binary functions as represented in table 1. The first 7 functions are standard ones in GAs evaluations as mentioned in (Whitley et al. 1995). Functions f_{11} (Ackley 1987), f_{12} (Wilson 1991), f_{13} (Goldberg et al. 1989) and f_{14} (Whitley 1991) are also standard ones in the theory of deceptiveness. The GA that we have used is a standard binary generational GA with binary tournament selection. In the following, p_{mut} , p_{cross} , $|pop|$ and G will denote respectively the probability of mutation and crossover, the population size and the number of generations. We have used one point crossover (1X).

	Function	Intervals of x_i	x_i coded on
1	$x_1^2 + x_2^2 + x_3^2$	$[-5.12, 5.11]$	10 bits
2	$100(x_1^2 - x_2^2)^2 + (1 - x_1)^2$	$[-2.048, 2.047]$	12 bits
3	$50 + \sum_{i=1}^5 (x_i^2 - 10\cos(2\pi x_i))$	$[-5.12, 5.11]$	10 bits
4	$1 + \sum_{i=1}^2 \frac{x_i^2}{4000} - \prod_{i=1}^2 \cos(\frac{x_i}{\sqrt{i}})$	$[-5.12, 5.11]$	10 bits
5	$1 + \sum_{i=1}^5 \frac{x_i^2}{4000} - \prod_{i=1}^5 \cos(\frac{x_i}{\sqrt{i}})$	$[-5.12, 5.11]$	10 bits
6	$0.5 + \frac{\sin^2(\sqrt{x_1^2 + x_2^2}) - 0.5}{1 + 0.001(x_1^2 + x_2^2)}$	$[-100, 100]$	10 bits
7	$(x_1^2 + x_2^2)^{0.25} (1 + \sin^2 50(x_1^2 + x_2^2)^{0.1})$	$[-100, 100]$	10 bits
8	$ x_1 + 2x_2 + 3x_3 + 4x_4 $	$[-100, 100]$	10 bits
9	$x_1 + 2x_2 + 3x_3 + x_4^2$	$[0, 100]$	10 bits
10	$x_1 x_2 x_3 + x_2 x_3 x_4$	$[0, 100]$	10 bits
11	Ackley's Onemax 30 bits	not applicable	n. a.
12	Wilson's non-deceptive 30 bits	n. a.	n. a.
13	Goldberg et al., fully deceptive 30 bits	n. a.	n. a.
14	Whitley fully deceptive 40 bits	n. a.	n. a.

Table 1. The 14 functions used in our tests. Note that the last four functions usually correspond to maximization problems, and we have changed them simply by $Constant - f$.

3.2 Initial epistasis computation

For each function and for sample of points of various sizes, we have firstly computed the epistasis correlation defined in section 2. The results are presented in table 2. A first comment is that the epistasis can be correctly approximated with samples of small size for functions 2, 9, 10, 11, 12, 13 and 14. For the other functions, the true and correct values converge only when the sample is large enough. Therefore, we have used in the following only the values of the last columns (2000 or 3000 points) of table 2. From this table, and if one assumes that the epistasis is correlated with the difficulty of functions for the GA, then one could be tempted to classify the functions in the following way : “easy” functions with high epistasis correlation would be functions 9, 10, and 11, “hard” functions with low epistasis correlation would be functions 1, 3, 4, 5, 6, 7, and 8, and “medium” functions with moderate epistasis correlation would be functions 2, 12, 13 and 14.

3.3 Relation to overall performances

If one considers now the performances of the GA on the 14 functions, then maybe the previous classification of functions holds too for the GA. To test this, we have run the binary GA with several parameters settings. These were $|pop| = 100$, $p_{cross} = 0.8$, $p_{mut} = 0.02, 0.05, 0.1$ and $G = 50, 100$. This amounts to 6 different sets of parameters which were tested over 10 trials. The performance of the GA is

Function	Epistasis correlation computed with				
	300 points	500 points	1000 points	2000 points	3000 points
1	0.294 (0.035)	0.248 (0.047)	0.172 (0.024)	0.125 (0.015)	0.100 (0.016)
2	0.524 (0.038)	0.527 (0.021)	0.514 (0.022)	0.503 (0.009)	0.501 (0.006)
3	0.393 (0.035)	0.304 (0.036)	0.213 (0.010)	0.157 (0.020)	0.127 (0.011)
4	0.252 (0.056)	0.197 (0.033)	0.135 (0.020)	0.094 (0.017)	0.078 (0.006)
5	0.395 (0.021)	0.309 (0.022)	0.213 (0.019)	0.162 (0.006)	0.128 (0.010)
6	0.257 (0.034)	0.178 (0.035)	0.150 (0.015)	0.095 (0.016)	0.074 (0.013)
7	0.241 (0.033)	0.196 (0.032)	0.138 (0.013)	0.088 (0.013)	0.081 (0.010)
8	0.329 (0.025)	0.262 (0.033)	0.182 (0.009)	0.139 (0.014)	0.114 (0.011)
9	0.925 (0.009)	0.936 (0.008)	0.953 (0.002)	0.960 (0.002)	0.963 (0.001)
10	0.854 (0.007)	0.865 (0.008)	0.874 (0.006)	0.877 (0.005)	0.878 (0.003)
11	0.962 (0.008)	0.974 (0.008)	0.986 (0.002)	0.993 (0.002)	0.996 (0.001)
12	0.479 (0.030)	0.471 (0.032)	0.444 (0.029)	0.447 (0.018)	0.443 (0.015)
13	0.436 (0.036)	0.434 (0.029)	0.402 (0.023)	0.380 (0.016)	0.383 (0.015)
14	0.591 (0.039)	0.580 (0.039)	0.555 (0.020)	0.543 (0.021)	0.551 (0.008)

Table 2. Epistasis correlation obtained for the 14 functions and averaged over 10 samples. Numbers in parenthesis are the standard deviations.

then measured by the obtained minimum averaged over these 10 runs. However, to compare the results and to compute a more useful performance measure, one must relate this performance to a basis which depends on each function. To achieve this, the performance of the GA on a given function is in fact related to the performance of a simple random generation of strings. Given a value of $|pop|$ and G , the GA evaluates a given number of strings, here 5000 or 10000. Thus, we have used a simple random generation of exactly the same number of strings as for the GA. This method, denoted by RANDOM in the following, consists simply in keeping the best randomly generated string. Its results have also been averaged over 10 runs.

A measure of a function difficulty for the GA can be computed by the ratio $DIFF = \frac{RAN-GA}{RAN}$ where RAN and GA denote respectively the minima found by RANDOM and the GA. This kind of measures are commonly used in combinatorial optimization in order to compare methods. If $DIFF$ is close to 1, then the GA performs much better than RANDOM. On the other hand, if $DIFF$ is negative, then the GA performs very poorly compared to RANDOM. Typical results obtained with $|pop| = 100$, $p_{cross} = 0.8$, $p_{mut} = 0.1$ and $G = 100$ are presented in table 3. From this table and according to the values of $DIFF$, one could classify the functions as follows: easy functions with high values of $DIFF$ would be functions 1, 9, 10, and 11, while hard functions with low values of $DIFF$ would be functions 7, 13 and 14. The first remark here is that functions with a very high epistasis correlation (like functions 9, 10 and 11) are indeed easy for the GA in practice. So the relation that seems to hold experimentally is “high epistasis correlation $\Rightarrow$ easy for GA”. Hopefully, this is the results which is

Function	Epistasis (3000 points)	$DIFF =$ $(RAN - GA)/RAN$	RAN	GA
1	0.100	0.760	0.074	0.018
2	0.501	0.532	0.003	0.002
3	0.127	0.357	18.095	11.636
4	0.078	0.447	0.155	0.086
5	0.128	0.328	3.736	2.509
6	0.074	0.406	0.020	0.012
7	0.081	0.255	1.288	0.960
8	0.114	1.000	0.039	0.000
9	0.963	0.694	44.045	13.464
10	0.878	1.000	0.000	0.000
11	0.996	0.667	4.800	1.600
12	0.443	0.500	6.800	3.400
13	0.383	0.243	22.200	16.800
14	0.551	0.280	45.000	32.400

Table 3. Results obtained by the GA with $|pop| = 100$, $p_{cross} = 0.8$, $p_{mut} = 0.1$ and $G = 100$ in comparison with a random search and averaged over 10 runs.

provided by the theory: linear functions are non-deceptive functions. Now, if one is looking for typical examples of low epistasis correlation and high GA performances, then functions 1 and 8 comply with this specification. So a low epistasis correlation is not a good indication about the true difficulty of the function.

Now, let us test the relation between the epistasis correlation and the functions' difficulty on the 6 sets of parameters. For this purpose, an overall measure for the 14 functions and for a given set of parameters is to compute the correlation that may exist between the epistasis measure (with 3000 points) and the measure of difficulty *DIFF*. Let us denote this correlation by $Corr(Epist, DIFF)$. For instance, in the data represented in table 3, this correlation equals 0.36. Table 4 computes this correlation for the 6 sets of parameters. This correlation seems to be increasing with the probability of mutation. It also seems to be higher at the beginning of the GA search rather than at the end. In all cases, this correlation is not very high and is not really significant.

4 Remapping the binary search space with epistasis measures

4.1 Principle

We consider in this section that it could be possible, at a low computational cost, to find a better binary representation of a search space than the initial representation, by using the epistasis measure. We thus look for a new representation of the search space where the correlation of epistasis would be closer to 1 than

$ pop $	p_{cross}	p_{mut}	G	$Corr(Epist, DIFF)$
100	0.8	0.02	100	-0.07
100	0.8	0.05	100	0.25
100	0.8	0.1	100	0.36
100	0.8	0.02	50	0.04
100	0.8	0.05	50	0.30
100	0.8	0.1	50	0.46

Table 4. Correlation between the measured epistasis correlation and the measured difficulty of functions for the 6 sets of parameters (and for the 14 functions over 10 trials).

in the initial search space. Then, using this new search space within the GA, we hope to make the function X easier and to improve the GA performances. Of course, this may not work for all functions: the problem of finding a good representation is most probably as hard as the initial optimization problem itself (Liepins and Vose 1990). What we believe is that maybe, in some cases, a better representation that increases the epistasis correlation can be easily found.

Let us consider a transformation R of the initial search space $S = \{0, 1\}^l$ into a new search space $S' = \{0, 1\}^l$. Any binary string s of S is changed to another string s' of S' with R . R must satisfy several constraints. For instance, one property to be kept absolutely is that the optima in S are changed into the optima in S' . Thus, searching for an optimum in S' will lead, once this optimum is found, directly to an optimum of S by applying R^{-1} to this optimum. R must thus be reversible/bijective, and furthermore, it must be computable in polynomial time in l in order to be realizable in practice. In order to evaluate a binary string $s' \in S'$, we use the following fitness function: $X'(s') = X(R^{-1}(s')) = X(s)$ where $s = R^{-1}(s')$. Any optimum in one of the search space will also be an optimum in the other search space. Increasing the epistasis correlation thus seems to be a good idea because it may linearize the function X . This leads to the following general algorithm:

1. Generate randomly a sample of strings $S_a \subset S$, and compute the epistasis correlation $e_{corr}(S_a)$,
2. Find a new representation R^* , such that $e_{corr}(R^*(S_a)) > e_{corr}(S_a)$, (where e_{corr} denotes the epistasis correlation),
3. Use the GA to find an optimum s'^* in $S' = R(S)$ (all strings in S' are evaluated by using the new fitness function $X'(s') = X(R^{-1}(s'))$),
4. Translate this optimum in the original search space S with the relation $s^* = R^{-1}(s'^*)$.

4.2 Remapping bits 3 by 3

The problem now is to generate transformations R from $\{0,1\}^l$ into itself which are bijective and easy to compute (see point 2 of the previous algorithm). One possibility consists in considering the bits of an individual s 3 by 3, and to remap each group of 3 bits according to the same bijective transformation r . The general transformation R is thus obtained by applying several times a simpler transformation r that works only on groups of 3 bits. r can be simply encoded in a lookup table of 8 entries, like for instance:

Input	000	001	010	011	100	101	110	111
↓	↓	↓	↓	↓	↓	↓	↓	↓
Output	101	011	001	100	000	010	111	110

Since r must be bijective, there are only $8!$ such transformations r of 3 bits to 3 bits. r is also easily reversible by considering the lookup table from outputs to inputs. These desired properties are also kept when considering R , since R is a repeated combination of a smaller transformation r . In addition, since here $l = 24, 30, 40$ or 50 , then l is not necessarily a multiple of 3, and thus the 2 (or 1) remaining bits at the end of the string are left unchanged.

4.3 Generating and evaluating transformations

Transformations R of the previous type can be generated randomly simply by performing permutations in the output row of the lookup table of r . The generated transformations can be evaluated according to their associated epistasis correlation. This requires simply to encode with R the strings generated in the sample of strings S_a . One should notice that this initial set is the same for every transformation, and since $X(R(s)) = X(s)$, one does not need to use the evaluation function on the transformed binary strings of $R(S_a)$ in order to evaluate a given transformation R . One can thus generate as many transformations as desired without evaluating new strings with X . This is a very important point which has the advantage in practice of searching for a better representation with no additional cost from the evaluation function point of view.

The best generated transformation R^* is the one which maximizes the epistasis correlation. If the epistasis correlation has not been improved, then $R^* = Identity$.

4.4 Experimental results

In this experiment, we have applied the previous algorithm to every functions of table 1. For each tested function, one can thus record the following information: the initial epistasis correlation and the initial performance of the GA without any transformation (denoted by E_{init} and GA_{init} in the following), and then the possibly improved final epistasis correlation and the final GA performance in the new search space (denoted respectively by E_{final} and GA_{final} in the following).

Function	E_{init}	E_{final}	GA_{init}	GA_{final}
1	0.112	0.638	0.0035	0.0082
2	0.505	0.723	0.0068	0.0046
3	0.160	0.394	5.7565	7.7567
4	0.091	0.190	0.0370	0.0568
5	0.152	0.521	1.4800	1.5076
6	0.097	0.121	0.0100	0.0132
7	0.097	0.119	0.7561	0.7384
8	0.145	0.233	0.0000	0.0000
9	0.959	0.959	4.0872	2.9024
10	0.875	0.875	0.0000	0.0000
11	0.993	0.993	0.0000	0.1000
12	0.439	0.909	0.2000	0.0000
13	0.386	0.919	11.0000	2.2000
14	0.555	0.636	24.0000	20.8000

Table 5. Results obtained when remapping the search space according to the epistasis correlation.

Table 5 gives the obtained results on the 14 functions averaged over 10 runs. The number of strings generated was 2000, and 20 transformations were generated randomly at each run in order to generate R^* . The parameters of the GA were $|pop| = 100$, $G = 100$, $p_{cross} = 0.8$ and $p_{mut} = 0.05$. The epistasis correlation is improved for most of the time except for functions 9, 10 and 11 where $R^* = Identity$. However, the performances of the GA are not improved with these apparently better representations, except for functions 12 and 13. Perhaps, this is due to the fact that functions 12 and 13 are defined on groups of 3 bits only and are thus very simple compared to the others. Maybe, even with a good encoding, the problem remains hard to optimize for the other functions and the improvement in performances cannot be detected.

One could argue that a problem of generalization takes place, like in machine learning: the best transformation generated with S_a improves the epistasis correlation of strings in $R^*(S_a)$, but maybe this improvement does not generalize to other strings of S . We have performed some tests that reject this hypothesis. This has consisted simply in generating another set of string S_b and checking that the transformation R^* learned on S_a also improves the epistasis correlation of transformed strings of S_b .

5 Conclusion and discussion

We have studied in this paper epistasis measures from an experimental point of view in order to check how useful these measures can be in order to predict the GA performances. We have presented two kinds of tests: trying to relate the epistasis measure to the GA overall performances evaluated in comparison with

a random search, and trying to find better binary representations that would increase the epistasis correlation of a sample of strings and possibly improve also the GA performances. Apparently, for the 14 tested functions, epistasis correlation predicts neither the GA performances nor the changes of representation with simple transformation. The only relation which seems to hold is that functions with high epistasis correlation are rather easy for GAs, and this can be related to other recent studies such as (Soule and Foster 1997).

These negative results should be tempered by several points. We have used only first-order epistasis. Maybe, second-order epistasis computed with schemata of order 2 would have given better information about the functions. Quadratic function like f_1 may have an epistasis correlation of order 2 that is closer to 1. Also, maybe our transformations are too simple: they change the search space in a good way on average but the relevant areas of the space may remain as hard as in the initial space. We believe that epistasis measures are interesting compared to others because they require low computational costs only, which is a necessary condition for bridging the gap between the theory and practice of GAs.

The immediate perspectives of this work will be presented in (Rochet et al. 1997). We try to relate also the epistasis correlation to the performances of 1X and UX crossovers. We use also more complex transformations. We have shown recently that epistasis measures can be generalized to real coded representations by using linear approximation of a function in R^l (Rochet et al. 1996). This allows us to perform similar experiments in continuous spaces.

References

- Ackley D.H. (1987), *A connectionist machine for genetic hillclimbing*, Boston, MA: Kluwer Academic.
- Davidor Y. (1991). Epistasis Variance: A Viewpoint on GA- Hardness. In *Foundations of Genetic Algorithms*. Gregory J. E. Rawlins (Ed). Morgan Kaufmann Publishers, San Mateo. pp 23-35.
- Goldberg D.E. (1987), Simple genetic algorithms and the minimal deceptive problem, *Genetic Algorithms and Simulated Annealing*, L. Davis (Ed), Morgan Kaufmann, pp 74-88.
- Goldberg D.E., Korb B. et Deb K. (1989), Messy genetic algorithms: motivations, analysis, and first results. *Complex systems* 4, pp 415-444.
- Grefenstette J.J. (1992), Deception considered harmful, *Proceedings of the second workshop on Foundations of Genetic Algorithms*, 1992, D. Whitley (Ed), Morgan Kaufmann, pp 75-91.
- Grefenstette J. J. (1995). Predictive Models Using Fitness Distributions of Genetic Operators. In *Foundation of Genetic Algorithms 3*. D. Whitley and M. Vose (Eds), Morgan Kaufmann, San Mateo, CA.
- Holland J.H. (1975), *Adaptation in natural and artificial systems*. Ann Arbor: University of Michigan Press.
- Jones T. and Forrest S. (1995), Fitness distance correlation as a measure of problem difficulty for genetic algorithms, *Proceedings of the Sixth Interna-*

- tional Conference on Genetic Algorithms, 1995, L.J. Eshelman (Ed), Morgan Kaufmann, pp 184-192.
- Liepins G. E., Vose M.D. (1990). Representationnal issues in genetic optimization. *Journal of Experimental and Theoretical Artificial Intelligence 2* Research paper, pp 101-115.
- Manderick B., de Weger M. and Spiessens P. (1991), The genetic algorithm and the structure of the fitness landscape, *Proceedings of the Fourth International Conference on Genetic Algorithms*, 1991, R.K. Belew and L.B. Booker (Eds), Morgan Kaufmann, pp 143-150.
- Manela M. and Campbell J.A. (1992), Harmonic analysis, epistasis and genetic algorithms, *Proceedings of the Second Conference on Parallel Problem Solving from Nature 1992*, R. Manner et B. Manderick (Eds), Elsevier, pp 57-64.
- Reeves C.R., Wright C. C. (1995). Epistasis in Genetic Algorithms: An Experimental Design Perspective. *Proceedings of the sixth International Conference on Genetic Algorithms*. Larry J. Eshelman (Ed), Morgan Kaufmann, San Mateo, CA. pp 217-224.
- Rochet S. (1997). Epistasis in genetic algorithms revisited. To appear in *Information Sciences*.
- Rochet S., Slimane M. and Venturini G. (1996), Epistasis for real encoding in genetic algorithms, *IEEE ANZIS'96*, V. L. Narasimhan and L. C. Jain (eds), Australia, pp. 268-271.
- Rochet S., G. Venturini, M. Slimane (1997), A critical study of epistasis measures, in preparation.
- Soule T. and Foster J.A. (1997), Genetic algorithm hardness measures applied to the maximum clique problem, *Proceedings of the seventh International Conference on Genetic Algorithms*, 1997, T. Baeck (Ed.), Morgan Kaufmann, pp 81-88.
- Venturini G., Rochet S. and Slimane M. (1997), Schemata and deception in binary genetic algorithms: a tutorial, to appear in *Control and Cybernetics*, Special Issue on Evolutionary Algorithms, M. Schoenauer and Z. Michalewicz (Eds).
- Whitley D. (1991), Fundamental principles of deception in genetic search, *Proceedings of the first Workshop on Foundations of Genetic Algorithms*, 1991, G.J.E. Rawlins (Ed), Morgan Kaufmann, pp 221-241.
- Whitley D., Mathias K., Rana S. and Dzubera J. (1995), Building better test functions, *Proceedings of the Sixth International Conference on Genetic Algorithms*, Eshelman L.J. (ed), Morgan Kaufmann publishers, pp. 239-246.
- Wilson S.W. (1991), GA-easy does not imply steepest-ascent optimizable, *Proceedings of the Fourth International Conference on Genetic Algorithms*, 1991, R.K. Belew and L.B. Booker (Eds), Morgan Kaufmann, pp 85-89.

A Priori Comparison of Binary Crossover Operators: No Universal Statistical Measure, But a Set of Hints

Leila Kallel and Marc Schoenauer

CMAP – URA CNRS 756

Ecole Polytechnique

Palaiseau 91128, France

FirstName.Name@polytechnique.fr

Abstract. The choice of an operator in evolutionary computation is generally based on comparative runs of the algorithms. However, some statistical a priori measures, based on samples of (parents-offspring), have been proposed to avoid such brute comparisons. This paper surveys some of these works in the context of binary crossover operators. We first extend these measures to overcome some of their limitations. Unfortunately, experimental results on well-known binary problems suggest that any of the measures used here can give false indications. Being all defined as averages, they can miss the important parts: none of the tested measures have been found to be a good indicator alone. However, considering together the mean improvement to a target value and the Fitness Operator Correlation gives the best predictive results. Moreover, detailed insights on the samples, based on some graphical layouts of the best offspring fitnesses, seem to allow more accurate predictions on the respective performances of binary crossover operators.

1 Introduction

The most simple and common way to compare some parameter settings (including the choice of operators) of evolutionary algorithms (EAs) is to run numerous experiments for each one of the settings, and to compare averages of on-line or off-line fitness. This trial-and-error experimental method is unfortunately time consuming. Nevertheless, the lack of efficient and robust measures of difficulty and/or adequation of settings for a given problem has made this experimental comparative method very popular since the early conferences of EAs (see e.g. [14]).

Some studies have proposed heuristical approaches to try to estimate the efficiency of different operators a priori. These methods are usually based on statistical measures computed on random samples of points of the search space. Two kinds of such methods can be distinguished :

The first set of methods do not take into account the specific characteristics of operators, but rather implicitly rely on conceptual notions that are common to all operators. For instance, the strong causality of mutations is assumed in [10], where a distance in the genotype space is used to assess correlation length, and in [8], where the correlation between fitness and distance to optimum is described as a good predictor of EA performances. In the same line, the respectfulness

of operators toward schemata [6, 12] or epistasis [4] is used to characterize the suitability of the representation and its associated operators.

The second kind of methods directly address the issue of operator performance relative to the fitness landscape. In the early days of evolution strategies, Rechenberg [13] derived the famous 1/5 rule by theoretical calculation of the expected progress rate toward the optimum for a zero mean Gaussian mutation, on the sphere and corridor models in $\mathbb{R}^n$, in relation to the probability of a successful mutation.

More recently, Weinberger [15] introduced the notion of fitness operator correlation, for asexual operators, as the correlation between the fitnesses of the parents and their iterated offspring. Manderick [10] adapted this correlation measure to crossover operators. He devoted attention to the correlation between mean parental fitness and mean offspring fitness, and indicated the possible usefulness of this measure to assess the role of an operator on the NK-landscapes and the TSP problems.

However, as noted by Altenberg [1], such fitness operator correlation suffers from theoretical limitations: a maximal correlation is possible even in the absence of any improvement of fitness from parents to offspring. Altenberg further suggested that the performance of a genetic algorithm is related to the probability distribution of offspring fitness indexed by their parents' fitness that result from application of the genetic operator during the GA. Unfortunately, Altenberg noted that empirical estimation of the fitness probability distribution could be problematic if the searched surface has multiple domains of attraction.

Grefenstette [7] used the probability distribution of offspring fitnesses indexed by the mean parents fitnesses to yield a first-order linear approximation of the suitability of a genetic operator. Another approach was suggested by Fogel and Ghozeil [5]: to compare different selection schemes and operators, they use the mean improvement of offspring, and concludes that the uniform intermediate recombinations yields better results (higher mean improvement) than the one-point crossover, for all three selection methods they tested, on some two-dimensional functions in the real-valued framework.

This paper is concerned with statistical a priori measures that can be used to assess crossover efficiency: All the above measures (fitness operator correlation, probability distributions, and mean improvement) have been tested separately. As far as we know, no comparison has been made of their efficiency on the same problem. The goal here is to achieve such a comparison on different binary problems from the GA literature. The different selection schemes will not be discussed here, and for all experiments linear ranked-based selection will be used.

The paper is organized as follows. Section 2 presents in detail and discusses the statistical measures briefly introduced above. A modified version of the fitness operator correlation is introduced to address some of its limitations. Mean improvement method is modified to only account for strictly positive improve-

ments. Section 3 experimentally compares the estimates of those different measures on various test cases in the light of actual results of GA runs, demonstrating the inadequacy of the fitness operator correlation for most problems, and the potentialities of the statistical measures based on best offspring fitnesses. These results yield a better understanding of the domain of validity and the limitations of each statistical measure. Moreover, a detailed look at some samples appeals for on-line measures: Further research directions are sketched in the last section, in light of the obtained results.

2 Comparing Crossover operators

This section details some of statistical measures that have been proposed in previous work, and discusses some of their limitations. This leads to proposing alternative measures.

Consider the problem of optimizing a real-valued fitness function $\mathcal{F}$ defined on a metric search space $\mathcal{E}$ and suppose that there exists one global optimum of $\mathcal{F}$. The common basis of all statistical measures described below is the use of a sample of random individuals in $\mathcal{E}$. The operators to be assessed are then applied on all (or part of) the sample.

2.1 Mean Fitness Operator Correlation

Manderick & al. [10] define the Mean Fitness Operator Correlation (M-FOC) as follows:

From a sample of n couples of individuals (x_i, y_i) , let (f_{pi}) be the mean fitness of couple i , and (f_{ci}) the mean fitness of their two offspring with crossover OP. The *M-FOC coefficient* of the operator OP is defined by:

$$\frac{\frac{1}{n} \sum_{i=1}^n (f_{pi} - \bar{f}_p)(f_{ci} - \bar{f}_c)}{\sigma_{F_p} \sigma_{F_d}}, \quad (1)$$

where $\bar{f}_p, \bar{f}_c, \sigma_{F_p}$ and σ_{F_d} are respectively the means and standard deviations of parents' and offspring mean fitness.

This coefficient measures the extent to which the mean fitness of a couple of parents can give some information about the mean fitness of their offspring. A very high correlation between parents and offspring (M-FOC coefficient close to 1) indicates that offspring mean fitness would be almost proportional to parental mean fitness. No correlation (M-FOC coefficient close to 0) shows that no information about offspring mean fitness can be gained from the parents mean fitness. GAs using high-correlated operators are expected to achieve best performances as demonstrated in [10] on the NK-landscapes and the TSP problems.

Unfortunately, suppose there is a quasi-linear relationship between the mean fitnesses of both sets of parents and offspring. The M-FOC correlation will be maximum (1), whatever the slope of that linear relation. In particular, if that slope is less than 1, no offspring fitness can be better (in mean) than its parents': the correlation can be maximal when no improvement is to be expected! However

unrealistic this particular case can be, it nevertheless shows the limits of M-FOC as a measure of operator's efficiency.

In that line, as noted by Fogel and Ghoeil [5], M-FOC cannot be used on problems that yield zero mean difference between parents and offspring fitnesses, as in the case of linear fitness functions with real-valued representations. The correlation is always maximal, and cannot reflect the high dependency of their convergence rate on the the standard deviation of the Gaussian mutation [13].

2.2 Best Fitness Operator Correlation

To address the above-mentioned limitations of M-FOC, we propose to study the correlation between the best (rather than mean) parent and offspring fitnesses. The definition of the B-FOC coefficient is the same as in equation 1, except that $\bar{f}_p, \bar{f}_c, \sigma_{F_p}$ and σ_{F_d} are replaced by the means and standard deviations of the best fitnesses of parents and offspring. This coefficient should avoid some of the ambiguities pointed out in the preceding subsection.

2.3 Mean Improvement

Fogel and Ghoeil [5] use 1000 couples of parents to compare different selection schemes and operators. To generate each couple, 100 random individuals are drawn. The couple is then chosen among them either randomly, or as the two best individuals, or as one random and the best. The mean improvement of the best offspring (of each couple) over the best individual of the initial 100 individuals is then computed. Such mean improvements are finally averaged over the 1000 couples. As noted in [5], one limitation of this approach is its computational cost (102.000 fitness evaluations). The following definitions are a tentative to achieve the same goals at lower cost: Consider a sample of N random individuals as $N/100$ disjoint subsets of individuals. Among each of these 100 size subsets, 100 couples of parents are selected randomly (selection scheme is presented in 3.1). Two cases are considered:

Mean Improvement over the best parent

In the same line as Fogel and Ghoeil's second procedure (computing the improvement over the best parent), we propose to compute the improvement of the best offspring of each couple over its best parent. The mean improvement of N couples of parents is computed as follows:

$$\text{MI-P} = \frac{1}{N} \sum_{i=1}^N \max(0, \mathcal{F}(\text{best offspring}) - \mathcal{F}(\text{best parent}))$$

Mean Improvement to target

Fogel and Ghoeil's first procedure amounts to computing improvement of the offspring over a target set to the best fitness of the current set of 100 individuals. Rather, we define a target $\mathcal{T}$ as the median of the set $\{f_i\}_{i=1, \frac{N}{100}}$ where f_i is the fitness of the best individual among those of the i th -100 size- subset generated. The mean improvement to target is then computed as follows:

$$\text{MI-T} = \frac{1}{N} \sum_{i=1}^N \begin{cases} \max(0, \mathcal{F}(\text{best offspring}) - \mathcal{T}) & \text{if } \mathcal{F}(\text{best offspring}) > \mathcal{F}(\text{best parent}) \\ 0 & \text{otherwise} \end{cases}$$

2.4 Probability of Improvement

The above definitions of MI-P and MI-T do not consider the number of offspring that actually are source of improvement. In order to separate the mean improvement from the probability of improvement, the latter is defined as the fraction of offspring actually having a better fitness than their parents (PI-P) or the target (PI-T). Two new measures are then defined, termed MI-P+ and MI-T+, by counting in the definitions of MI-P and MI-T only those offspring.

3 Experimental Validation

3.1 Experimental conditions

From now on, the search space will be the binary space $\{0,1\}^n$ ($n = 900$ for all problems, except the Royal Road), and the underlying EA some standard binary GA: population size 100, linear ranked-based selection, crossover rate of 0.6, mutation rate of $1/n$ per bit, elitist generational replacement.

Two operators are compared: The uniform crossover (UC) and the one-point crossover (1C). For each of them, the statistical measures are computed either on samples of 10000 individuals, with one crossover per parent, or on samples of 200 individuals, each couple generating 100 offspring (by repeating 50 crossovers operations). The parents are selected with a linear ranking selection applied to each disjoint subset of 100 individuals. In both cases, the computational cost is 20000 fitness evaluations per operator.

The goal of these experiments is to compare operators under the same conditions of initialization and selection. Therefore neither the selection scheme (as in [5]) nor the initialization procedure (as in [9]) will be modified.

The stability of all the following statistical measures have been checked carefully. The relative difference between measures on UC and 1C, has been found stable over 10 trials in most cases. However, all unstability cases are marked with a “*” in the tables.

3.2 Test cases

Onemax problems

In the well-known *onemax* problem, the fitness function is simply the number of 1's in the bitstring. Hence, the fitness is equal to n minus the distance to the global optimum. In order to avoid structural bias, modified versions of the onemax problem were designed: The fitness function is defined as the Hamming distance to a fixed bitstring. That bitstring is chosen to have a certain number

O of 1's, randomly placed in the bitstring. Such problem is termed the $(O, n-O)$ -*onemax* problem. Different values for O were experimented: (900, 0), (800, 100) and (450, 450).

Gray-coded Baluja functions F1 and F3

Consider the functions of k variables $(x_1, \dots, x_k)$, with $x_i \in [-2.56, 2.56]$ [2]:

$$F_1(\mathbf{x}) = \frac{100}{10^{-5} + \sum_{i=0}^{k-1} |y_i|}, y_0 = x_0 \text{ and } y_i = x_i + y_{i-1} \text{ for } i = 1, \dots, k-1$$

$$F_3(\mathbf{x}) = \frac{100}{10^{-5} + \sum_i |.024 \star (i+1) - x_i|}$$

They reach their maximum value of 10^7 at point $(0, \dots, 0)$. The Gray-encoded versions of F_i , with 100 variables encoded on 9 bits each are considered.

The 4 peaks problem

In the FourPeaks problem [3], the fitness is the maximum of the length of the sequence of 0's starting at the first bit position and the length of the sequence of 1's ending at the last position, plus a reward of n if both sequences are larger than a given threshold T . There are two global optima, made of a block of 0's followed by a block of 1's, of lengths T and $n-T$ or $n-T$ and T . However, there are two local optima, the all 1's and all 0's bitstrings, from which it is difficult for a GA to escape. The most difficult instance of that problem (i.e. $T = n/4$) is used here.

The Ugly problem

The Ugly problem [16] is defined from an elementary deceptive 3-bit problem ($F(x) = 3$ if $x = 111$; $F(x) = 2$ for x in $0 \star \star$, and $F(x) = 0$ otherwise). The -deceptive - full problem is composed of 300 concatenated elementary deceptive problems. The maximum is the all 1's bitstring.

The Royal Road problem

The 64-bits Royal Road problem used here was conceived to study into details the combination of features most adapted to GA search (laying a *Royal Road*). A precise definition as well as an analysis of the unexpected difficulties of this problem can be found in [11].

3.3 Results

For all problems above, the statistical measures defined in section 2 have been computed, and their predictions compared to actual GA runs.

Average on-line results of the runs (over 21 runs) are presented in Figure 1, and the statistical measures for both uniform (UC) and one-point (1C) crossovers are presented in tables 1 and 2. Each table presents FOC and MI-P coefficients with one crossover per parent, as well as MI-T coefficients for both 1 and 50 crossovers per parent. Only one of the three onemax problems is presented since the two others gave almost the same results.

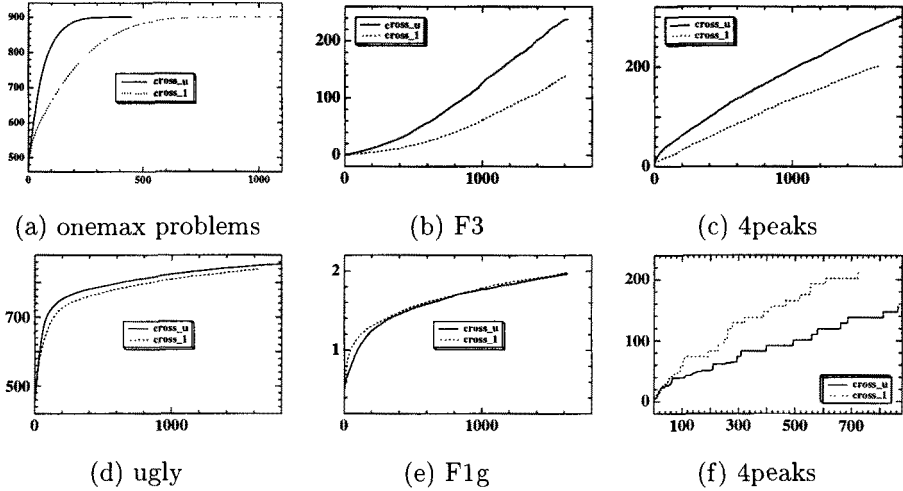


Figure 1: *Average on-line results of 21 GA runs on different problems (fitness $\times$ generation): influence of the crossover operators.*

Large unstability has been found for MI-T with the Royal Road problem. This is due to the highly discontinuous fitness values it yields (observed values are limited to 1, 5.2, 5.4, 21.6). Hence, when computing MI-T on few individuals, this can amount to contradictory results for different samples. More generally, MI-T has been found slightly unstable. This is not surprising when considering that very low probabilities are involved (lines PI-T in the tables). However, when considering the case of 50 crossovers per parent, the unstability does no longer alter the significance of the differences between MI-T coefficients of operators. A last general remark is that all MI coefficients should be considered relatively to the actual fitness range in the sample.

FOC Coefficients The only case where higher M-FOC coefficient and better convergence coincide is obtained with the F1g problem! On the 4peaks, Ugly and F3g problems the reverse situation is observed: Better on-line performances (Figure 1) are obtained for the operator with lower M-FOC values (Tables 1 and 2). On the opposite, runs for the Royal Road problem with either UC or 1C are similar (the 99% T-test fails) until generation 500, though, their respective M-FOC coefficients are quite different (Table 2).

On the same problems, B-FOC give similar indications than M-FOC, and hence suffer the same defects. On the Onemax problems, where the M-FOC always equals 1, the B-FOC coefficient reaches higher values for the 1C, while the runs (Figure 1) using UC are far better than those using 1C.

Hence neither M-FOC not B-FOC seems an accurate predictor of relative performance of operators, at least under linear ranking selection.

	onemax problems		F3g		4peaks	
	UC	1C	UC	1C	UC	1C
M-FOC	1	1	0.88	0.99	0.7	0.93
B-FOC	0.7	0.8	0.6	0.8	0.6	0.99
MI-P	3.4(0.03)	2.7(0.03)	0.01	0.008	0.4	0.002
MI-P+	7(0.08)	6(0.06)	0.02	0.016	2	2
PI-P	0.23	0.23	0.26	0.25	0.1	0.003
MI-T	0.07(0.012)	0.06(0.016)	0.00026	0.00021	0.1	0.002
MI-T+	5.6(0.3)	5.6(0.5)	0.02	0.02	2.2	1.8
PI-T	0.006	0.005	0.006	0.005	0.005	6e-5
MI-T (50)	3.2(1)	1.2(0.7)	0.01	0.004	0.6	0.02
MI-T+ (50)	8(1)	6(1)	0.028	0.025	2.7	1.7
PI-T (50)	0.02	0.018	0.018	0.016	0.01	0.0001

Table 1: Mean(standard deviation) of statistical a priori measures for different problems, on a 10000 individuals sample, 2 offspring for each parent with the uniform and one-point crossover operators. Bold fonts indicates measures on a 200 individuals sample, and 100 offspring for each parent. The + concerns strictly positive improvements only.

	ugly		Flg		royal	
	UC	1C	UC	1C	UC	1C
M-FOC	0.74	0.99	0.2	0.7	0.18	0.92
B-FOC	0.5	0.8	0.1(0.01)	0.6(0.005)	0.18	0.91
MI-P	5.9(0.07)	3.6(0.05)	0.05(7e-4)	0.03(4e-4)	0.19	0.019
MI-P+	12.3(0.1)	7.7(0.1)	0.09(9e-4)	0.05(8e-3)	4.2	4
PI-P	0.27	0.23	0.3	0.28	0.02	0.002
MI-T	0.08(0.01)	0.07(0.008)	7e-4(1e-4)	5e-4(1e-4)	e-3(8e-4)	2e-4(4e-4)
MI-T+	6.6(0.5)	6.8(0.5)	0.05(2e-3)	0.05(5e-3)	1.5(1.2)	0.9(2.3)
PI-T	0.006	0.005	0.007	0.006	0.0004	0.0001
MI-T(50)	5(2)	1.3(0.8)	0.05(0.02)	0.02(7e-3)	0.07(0.07)	0.01(0.02)
MI-T+(50)	10(2)	8(1.5)	0.07(0.01)	0.06(0.01)	2.5(2.8)	0.6(4)
PI-T(50)	0.018	0.016	0.016	0.015	4e-4(1e-4)	4e-4(8e-4)

Table 2: Mean(standard deviation) of statistical a priori measures for different problems, on a 10000 individuals sample, 2 offspring for each parent with the uniform and one-point crossover operators. Bold fonts indicates measures on a 200 individuals sample, and 100 offspring for each parent.

Mean Improvement to Parents If we consider the Flg problem, the results of the runs (figure 1), show that the one-point crossover average run is better than the uniform one until generation 200 (confirmed with a 99 % confidence T-test). But mean improvement to parents values (Table 2), suggest the opposite. A close look at the plot of best parents $\times$ offspring (Figure 2), shows that UC

yields better improvement than OP only at low fitness values of (best) parents. The situation is reversed for higher fitnesses. This situation could have been predicted by the Best FOC results: a 0.7 correlation for the 1C means that the best offspring's fitness is almost proportional to its best parent's fitness, while the 0.2 value for the UC means on the otherhand that well-fitted offspring can be obtained whatever the (best) parent's fitness. The misleading results of improvement to parents are so no longer surprising. It seems that MI-P values can be compared reliably when the B-FOC coefficients are similar, as it is the case of the F3g and the three onemax problems.

For the Ugly and Royal Road problems, the situation is similar to that of F1g problem. Average improvements to parents are very different in favor to UC, whereas the runs show identical performances: the T-test with 99 % confidence finds no difference between average runs until generation 30 (Ugly) and 500 (Royal Road). Again, at the opposite of 1C, the UC yields good fitness offspring from low fitness parents, as illustrated by the B-FOC coefficients.

Note however that the reverse is not true: different B-FOC values do not necessarily imply false MI-P previsions, as illustrated by the 4peaks problem.

Considering the probabilities of improvement (PI-P and MI-P+) for the F1g and Ugly problems does not improve the results: the same bias as for MI-P is observed, for the same reasons (different B-FOC coefficients).

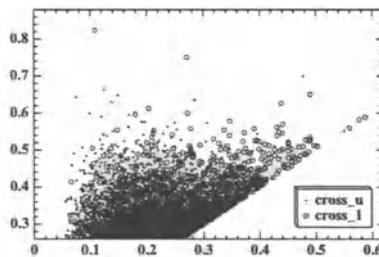


Figure 2 : *Fitness of best offspring $\times$ fitness of best parent, for the offspring that outperformed their parents, on the F1g problem.*

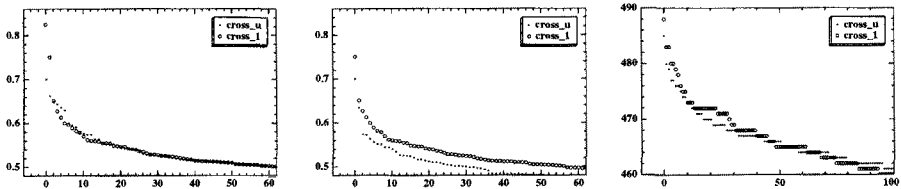
Mean Improvement to Target The case of 1 crossover per parent is found to be rather unstable and often involves very low improvement probabilities, which does not help comparison. Note for example that for the onemax problems (table 1), the observed differences of MI-T, with one crossover per parent, are very small (around 0.01) relatively to the range of fitness values of the problem ([400,500]). Hence only the case of 50 crossovers per parents (bold fonts in tables) will be considered in the following.

When considering the improvement to target, the dependency on the parents' fitness vanishes. So no bias is induced by the dissymmetric repartitions of improvement to parents. Nevertheless, for F1g and Ugly problems, MI-T values (Table 2) are still indicating some superiority of UC, whereas the opposite actually happens during the GA runs for F1g, and Ugly runs are similar until

generation 30 (Figure 1). The main reason for that is probably that averaging the improvements hides both the evolution of their differences and the best fitted offspring obtained: For the F1g problem, when getting to higher parents' fitness, the 1C gets better improvement to target than the UC as can be seen for instance on the plot (best parent $\times$ best offspring) of Figure 2.

Considered together with the probability of improvement, some additional information about the sample repartition can be obtained. In fact, IP-T does take into account the whole set of generated offspring, while MI-T does only take into account the set of best offspring of each parent. For the problematic cases of F1g and Ugly, MI-T+ values are still favoring the wrong operator, but with much smaller differences than MI-T, and IP-T values are close for both operators. MI-T+ still suffers from the bias of averaging, and does nevertheless hide the most important information about offspring, which is their maximal fitnesses, and how they evolve when higher parents are considered.

Further directions Other measures, not presented here, have been computed for both UC and 1C operators; such as mean fitness values of all offspring, or of all offspring that outperformed their parents. None of these have been found of any help to find about operators suitability. Rather, the most important factor is found to be the 'maximal potential' of an operator, i.e. how fit are the best offspring it can give birth to.



(a) F1g :whole sample (b) F1g: Half sample (c) Ugly: whole sample
Figure 3: *Fitnesses of best offspring $\times$ their rank, with one crossover per parent, for uniform and one-point crossovers: The relative position of curves changes when only the top half of the sample is considered for offspring generation.*

To illustrate this, consider the plots of fitness repartition of offspring as a function of their rank according to their fitness. The case of F1g problem presented in Figure 3-a shows the best offspring of the whole sample. One cannot say which curve is better: 1C yields higher fitness offspring than UC, but UC gives a higher number of well-fitted offspring. (Remember that the all the considered measures suggest the superiority of UC, and that the runs are ordered the other way round.) But consider now Figure 3-b, which plots the same repartition of best offspring, but when considering only the best half of the parents in the sample. The curve for 1C gets largely better than UC. Figure 3-c presents the case of the ugly problem: curves of 1C and UC are similar (and so are the runs), while all the previous measures favor the UC.

This plot was found efficient in predicting the runs order for all the considered problems. The plot is almost unchanged when considered with the top half of the sample.

A crucial factor for operator comparisons seems to be their ability to generate high values, even rarely, or even for the best of parents. And this can be quite different from any mean value on the whole sample. The mean as a measure does not give any information about the most likely direction of evolution. This calls for alternative measures, as well as for on-line dynamic measurement.

4 Summary and conclusion

Some statistical measures to allow assessment of a priori suitability of a genetic operator have been presented and experimented to compare 1-point crossover and uniform crossover in the binary framework.

These experiments have first confirmed the inadequacy of fitness operator correlation, which was either unable to detect differences, or even predicting the inverse of actual results of GA. FOC only indicates whether well-fitted offspring are obtained from well-fitted parents or not. However, such information can be useful in addition to other measures.

Average improvement to parents (MI-P) can be very misleading for two reasons:

1- The dependency on the parents fitnesses can induce big differences (in MI-P) while the fitness distribution of offspring -independently of their parents- is the same for different operators.

2- It suffers from the loss of information induced by any measure based on averaging.

Those remarks address both cases of 1 and 50 crossovers per parents, which generally gave the same tendencies, the case of 50 crossovers, showing much higher differences.

When considering average improvement to target (MI-T), only the case of 50 crossovers per parents gave stable and significant differences relatively to the fitness range of each problem. MI-T does no longer suffers from the first bias above but still meets the second:

MI-T, even with 50 crossovers per parents, fails on both F1g and Ugly problems to predict the best crossover. In our opinion, this is due to the second bias of average measure. In fact, on the F1g problem, the relative position of offspring fitness distribution for UC and 1C operators changes when considering only fit parents. Moreover, the averaging does not capture any information about the fittest offspring of each operator. This information, is found to be crucial to compare crossovers.

So none of the proposed measures was found satisfactory alone, though MI-T appears the most useful – especially when B-FOC values are similar. To design

more efficient measures, we think averaging should be avoided, and best fitness offspring should be emphasized. Whatever the measure, the most important factor seems to be the way it evolves when getting to higher fitness parents: Further work will have to design online measures.

References

1. L. Altenberg. The schema theorem and Price's theorem. In L. D. Whitley and M. D. Vose, editors, *Foundations of Genetic Algorithms 3*, pages 23–49, San Mateo, CA, 1995. Morgan Kaufmann.
2. S. Baluja. An empirical comparizon of seven iterative and evolutionary function optimization heuristics. Technical Report CMU-CS-95-193, Carnegie Mellon University, 1995.
3. S. Baluja and R. Caruana. Removing the genetics from the standard genetic algorithms. In A. Prieditis and S. Russel, editors, *Proceedings of ICML95*, pages 38–46. Morgan Kaufmann, 1995.
4. Y. Davidor. Epistasis variance: A viewpoint on representations, GA hardness, and deception. *Complex systems*, 4:369–383, 1990.
5. D. B. Fogel and A. Ghozeil. Using fitness distributions to design more efficient evolutionary computations. In T. Fukuda, editor, *Proceedings of the Third IEEE International Conference on Evolutionary Computation*, pages 11–19. IEEE, 1996.
6. D. E. Goldberg and M. Rudnick. Genetic algorithms and the variance of fitness. *Complex Systems*, 5:266–278, 1991.
7. J. J. Grefenstette. Predictive models using fitness distributions of genetic operators. In L. D. Whitley and M. D. Vose, editors, *Foundations of Genetic Algorithms 3*, pages 139–161. Morgan Kaufmann, 1995.
8. T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In L. J. Eshelman, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 184–192. Morgan Kaufmann, 1995.
9. L. Kallel and M. Schoenauer. Alternative random initialization in genetic algorithms. In Th. Bäck, editor, *Proceedings of the 7th International Conference on Genetic Algorithms*. Morgan Kaufmann, 1997. To appear.
10. B. Manderick, M. de Weger, and P. Spiessens. The genetic algorithm and the structure of the fitness landscape. In R. K. Belew and L. B. Booker, editors, *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 143–150. Morgan Kaufmann, 1991.
11. M. Mitchell and J.H. Holland. When will a genetic algorithm outperform hill-climbing? In S. Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms*, page 647, 1993.
12. N. J. Radcliffe and P. D. Surry. Fitness variance of formae and performance prediction. In L. D. Whitley and M. D. Vose, editors, *Foundations of Genetic Algorithms 3*, pages 51–72. Morgan Kaufmann, 1995.
13. I. Rechenberg. *Evolutionstrategie: Optimierung Technischer Systeme nach Prinzipien des Biologischen Evolution*. Fromman-Holzboog Verlag, Stuttgart, 1973.
14. J. D. Schaffer, R. A. Caruana, L. Eshelman, and R. Das. A study of control parameters affecting on-line performance of genetic algorithms for function optimization. In J. D. Schaffer, editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 51–60. Morgan Kaufmann, 1989.

15. E.D. Weinberger. Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological Cybernetics*, 63:325–336, 1990.
16. D. Whitley. Fundamental principles of deception in genetic search. In G. J. E. Rawlins, editor, *Foundations of Genetic Algorithms*. Morgan Kaufmann, 1991.

Evolvable Hardware and Robotics

The Dynamical Nightwatch's Problem Solved by the Autonomous Micro-Robot Khepera

A. Löffler*, J. Klahold, and U. Rückert

Paderborn University, Heinz Nixdorf Institute, System and Circuit Technology,
Fürstenallee 11, D-33102 Paderborn, Germany

E-mail: loeffler,klahold,rueckert@hni.uni-paderborn.de
<http://hni.uni-paderborn.de/fachgruppen/rueckert>

Abstract. In this paper, we present the implementation, both in a simulator and in a real-robot version, of an efficient solution to the so-called dynamical nightwatch's problem on the micro-robot Khepera. The problem consists mainly in exploring a previously unknown environment while detecting, registering and recognizing light sources which may dynamically be turned on and off. At the end of each round a report is requested from the robot. Therein we made use of an agent-based approach and applied a self-organizing feature map in order to refine some of the behaviour generating control-modules.

1 Introduction

1.1 Motivation

Our basic motivation to consider the implementation of behaviour generating control structures in mobile autonomous systems is represented by the following question: "What is the most complex task a robot is able to carry out, thereby coping with its limited resources, e.g. the available sensor sources, finite energy supply and limited processor power? The encountered consequences may be a restricted perception of its environment, runtime constraints and the problem of severe real-time demands." To be able to successfully envisage this kind of problem, we strongly believe in adopting the following two-step program: firstly, to implement software solutions making recourse to concepts of neural information processing; secondly, to replace single software modules by resource-efficient microelectronic components. In this framework, the contents of the present paper corresponds to the first point, i.e. the implementation of a software solution to a hard robotics' problem. Hereby, the micro-robot Khepera (see Fig.1) serves as an exemplary model for a mobile autonomous system.

Processing the sensor data of a mobile robot appropriately seems still to be a difficult and largely unsolved problem [1]. Hence, it remains an in general challenging task to consistently implement a set of behaviour generating control

* Supported by DFG-Graduiertenkolleg "Parallele Rechnernetze in der Produktionstechnik", GRK 124/2-96.

modules based on the different sensor sources of the system in question. Considering the micro-robot Khepera in its basic configuration, there are three main sources of sensor data which may be used to implement such modules:

<i>source of sensor data</i>	<i>task of the respective control-module</i>
(1) the infra-red proximity sensors	→ exploration of the environment
(2) the ambient light sensors	→ detection and registration of light sources
(3) the wheel-based step counters	→ basic positioning system.

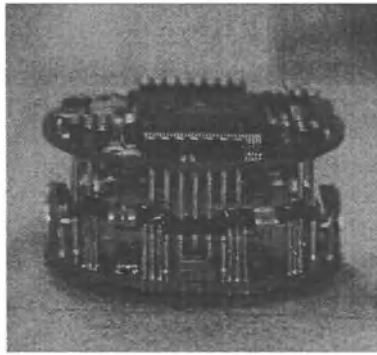


Fig. 1. Khepera is a micro-robot which has a diameter of 55mm, a height of 30mm and weighs 70g. The processor system contains a Motorola 68331 with 256Kbyte RAM and 256Kbyte ROM. Khepera perceives its environment using 8 infrared distance- and light-sensors; moreover incremental encoders placed on each motor axis are available for step counting.

In the next paragraph, we define an appropriate problem, i.e. one which includes all three of the above mentioned tasks.

1.2 The Dynamical Nightwatch's Problem

Imagine the town's nightwatch strolling over central square at late evening, looking for light sources, e.g. street lamps, lighted windows, etc., thereby avoiding obstacles, following walls (because he is a bit short-sighted) and turning round at dead ends. Having found the first light source, he fixes his report-board at the wall next to it to write down everything he will have encountered when he will arrive again at this location. In fact, he notes whether he has found a new lantern or recognized an old one and its current state (on/off) since the last stop at the board. Probably he would try to guess his position on the square by counting his steps and using already registered light sources as local landmarks to verify his counting.

The micro-robot Khepera should be able to perform an analogous task when transferred to a kind of suitable toy-world.

1.3 Design Principles

Animals may be considered as biological analogies to mobile autonomous robots. Other than the latter, they constantly had (and still have) to prove their fitness by surviving a natural selection process. Thus, it seems profitable to exploit their evolutionary optimized strategies for robot design, in this context especially those which deal with information processing. In order to do this, we adopted the following design principles inspired by concepts of neural information processing thereby making recourse to the so called agent paradigm [2–4] and some earlier works in cybernetics [5] :

- simple modules cope with simple tasks (e.g. obstacle avoidance, edge following, etc.)
- the control-modules are organized in parallel, i.e. no explicit hierarchical structure is implemented, and their results are simply superpositioned. In case of this not being possible, a data/event-driven decision is made (e.g. in dead ends, the turning-round-module takes control).
- no global goals are explicitly given to the robot
- arising complex behaviour is thus a result of the interaction of the basic modules
- adaptive algorithms, e.g. neural networks, are used to incorporate acquired informations about the environment.

2 Exploration

Enabling Khepera to explore its environment, we had to ensure free motion, i.e. avoiding obstacles, leaving substructures of the environment and not getting stuck in dead ends or similar structures. This was attained by implementing the following three basic control-modules:

2.1 Obstacle Avoidance

The obstacle avoidance was implemented as in Braitenberg's vehicle IIIb [5] using the very simple neural concept of cross-inhibition. This means that if one of Khepera's infra-red sensors detects an obstacle, the opposite wheel will be slowed down proportionally to the sensor input. By this way the obstacle is avoided (see Fig.2).

2.2 Edge Following

Khepera tries to keep the distance to the wall, i.e. the value of the outmost sensor, constant (see Fig.3). Note that no explicit distance to the encountered wall

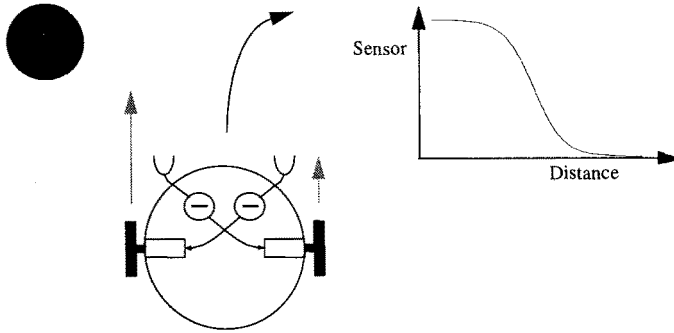


Fig. 2. A sketch of the robot avoiding a cylindrical obstacle (left) and an infra-red sensor vs. distance-to-light source characteristic are shown (right).

is be kept which allows a more flexible behaviour of the robot than otherwise be possible, especially in narrow corridors and similar structures. When a wall is detected for the first time (and avoided by means of the obstacle avoidance module), the outmost sensorvalue is registered. Afterwards, the robot will turn away from the wall, if this sensorvalue increases, and respectively will turn towards it, if it decreases. This algorithm is particularly simple, easy to implement and robust. Edge following allows to leave substructures of the environment quicker than it would be possible otherwise.

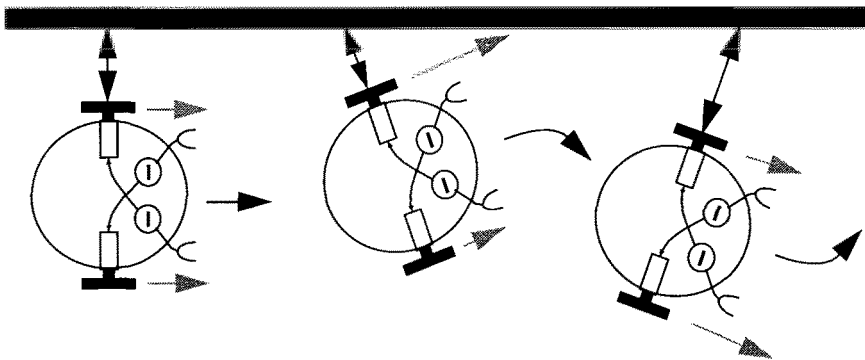


Fig. 3. Visualisation of the simple edge-following algorithm where three different situations might occur: a wall is detected (register sensorvalue_memory), turn away from wall (if sensorvalue > sensorvalue_memory), turn towards wall (if sensorvalue < sensorvalue_memory).

2.3 Turning

If a situation occurs in which Khepera is not able to leave a small area (symbolized by the circle) after some tries, it simply turns round to where it came from (see Fig.4). This module ensures that the robot does not get stuck in dead end like structures.

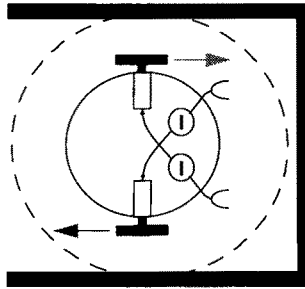


Fig. 4. The turning procedure.

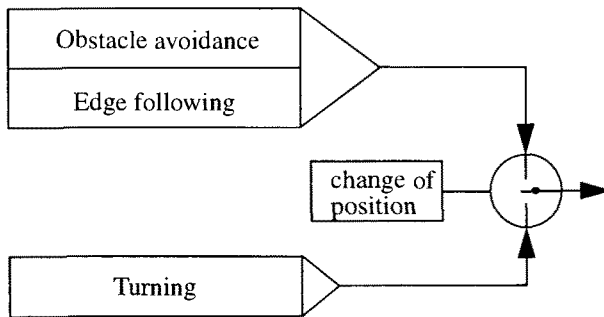


Fig. 5. The interplay of the three control modules ensuring constant exploration.

The interplay of the three modules is shown in Fig.5. A module, recognising the change of the position of Khepera, decides whether obstacle avoidance and edge following or the turning procedure get active. This means that if the robot is not able to change its position sufficiently during some 50 steps, it will turn around thereby suppressing the two other exploration modules; otherwise a superposition of the normal obstacle avoidance/edge following algorithms is used.

3 Positioning

An at least locally reliable positioning system had to be necessarily at basis of the light-source mapping process. For this purpose, we used the step counting functionality of Khepera. The position vector of Khepera consists of an x -value, y -value and the direction-indicating angel α . The new position (x, y, α) is calculated from the old one (x_0, y_0, α_0) using the values n_r and n_l given by the incremental encoders placed on each motor axis. The table below shows how the new position is calculated:

Table 1. Calculation of the change of position

				
$ n_l = n_r $	n_r S	n_r S	$n_r - n_l$ T	$n_r - n_l$ T
$ n_l < n_r $	n_l S	n_l S	$-2 * n_l$ T	$-2 * n_l$ T
	$n_r - n_l$ C	$n_r - n_l$ C	$n_r + n_l$ C	$n_r + n_l$ B
$ n_l > n_r $	n_r S	n_r S	$2 * n_r$ T	$2 * n_r$ T
	$n_r - n_l$ C	$n_r - n_l$ C	$-(n_r + n_l)$ C	$-(n_r + n_l)$ C

Straight:

$$x = x_0 + n \cdot \Delta l \cdot \cos \alpha$$

$$y = y_0 + n \cdot \Delta l \cdot \sin \alpha$$

Turn:

$$\alpha = \alpha_0 + \frac{n}{2 \cdot r} \cdot \Delta l$$

Curve:

$$\alpha = \alpha_0 + \frac{n}{2 \cdot r} \cdot \Delta l$$

$$x = x_0 + r \cdot (\cos \alpha - \cos \alpha_0)$$

$$y = y_0 + r \cdot (\sin \alpha - \sin \alpha_0)$$

4 Sensor Calibration

Adapting an one-dimensional self-organizing feature map [7, 8] by training with a real robot data set, we managed to calibrate the ambient light-sensors (see Fig.6), i.e. to transform the coarse grained information of Khepera's angle-to-light-source into a fine grained one; this also helped to overcome sensor-response variations due to fabrication tolerances. Calibration of the ambient light sensors means to classify the sensor vectors by an one-dimensional Kohonen feature map. Each neuron is active for a certain class of sensor vectors giving a direct information about Khepera's angle-to-light source.

The charts (Fig.7) show the sensor data, respectively used to learn and to test the neural network. Before usage, the data were standardized by a scaling process, which increased the accuracy considerably.

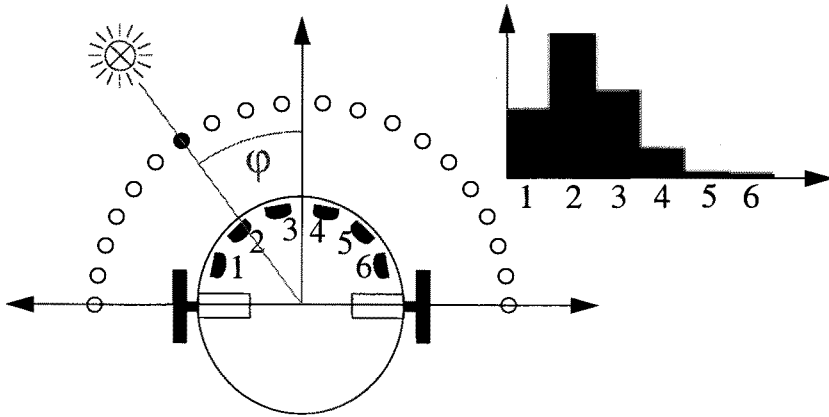


Fig. 6. The picture on the left shows that every neuron belongs to a specified angel. Each neuron is trained to react on a special characteristic of sensor values (e.g. the neuron represented by the filled circle reacts on the sensor vector given in the bar chart on the right).

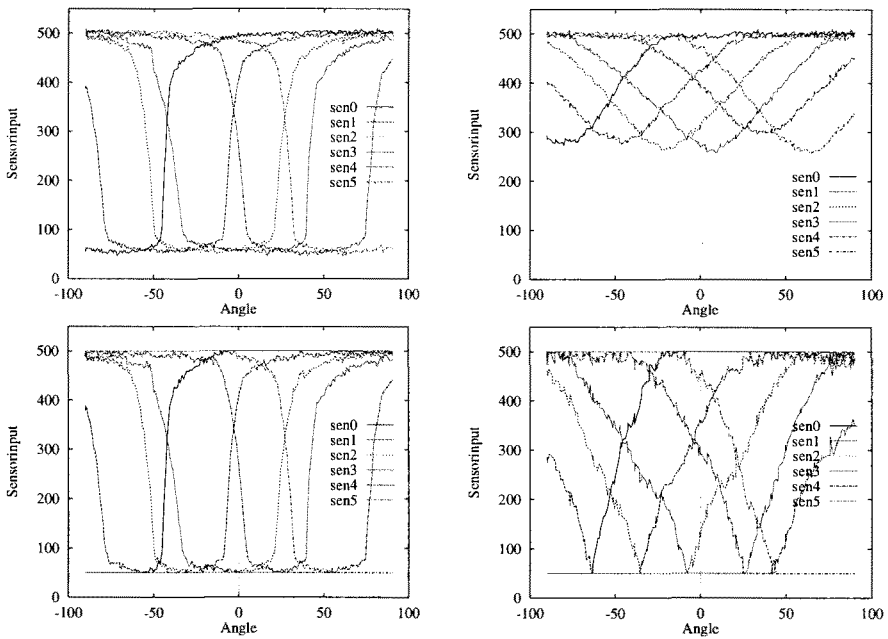


Fig. 7. The sensor data sets used to train and test the neural net; upper left: learning data, upper right: test data, lower left: scaled learning data, lower right: scaled test data.

The network consists of 60 neurons. It was trained in 20 cycles with an incremental change of the neighbourhood-width from 10 to 1 and a learning rate from 0.6 to 0.1. The result was an average error of 1.52 and a maximum one of 6.66 degrees at the recall process. The test at double distance caused an average error of 6.52 and a maximum one of 21.89 degrees. The results are graphically depicted in Fig.8.

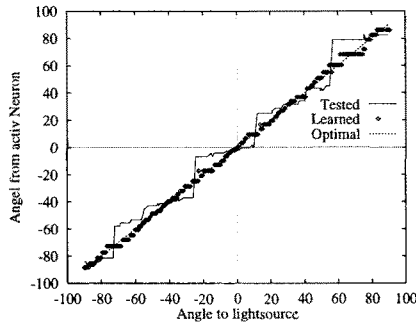


Fig. 8. The result of the adaptation process is shown in this chart. The dotted line serves as a referential, the bold dots correspond to the data being learned and the strait line belongs to data registered at the double distance with respect to the learning set.

5 Mapping Procedure

The procedure of the position-determination of detected light-sources was implemented as follows (see Fig.9). If the robot detects a light-source (1), Khepera approaches it (2) until one sensor value exceeds a certain threshold (optimal range for the neural network is reached) (3). Then Khepera registers its position vector (x_1, y_1, α_1) , turns by 90 degrees and moves in a strait line (4) to a second turning point (5). Afterwards, the robot reapproaches the light-source (6) until the optimal range is reached again (7). With the new position vector (x_2, y_2, α_2) and the old one, the location of the target can be calculated.

In this section, the conditions for changing the state of a light source are explained (see Fig.10). If Khepera penetrates the inner circle (in the Manhattan distance) of a previously discovered light source without detecting it, the state of the light source will be changed from on to off. If a “new” light source is registered within the outer circle of a previously discovered one, they are recognized as being identical. Eventually, the internal position of the robot is readjusted by a comparison of the previously and the currently registered location of the light source.

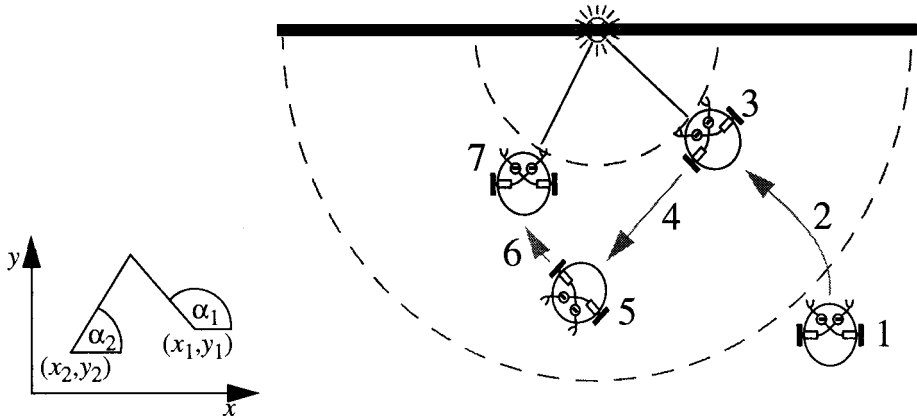


Fig. 9. A schematic view of the successive steps of the mapping procedure (right) and a sketch of the triangulation process using the registered data (left), hence determining the detected light source's location.

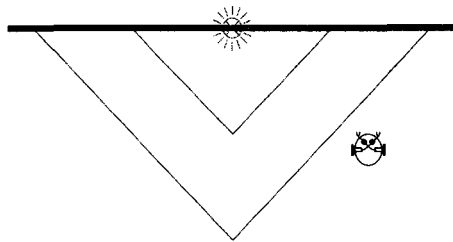


Fig. 10. Proportional view of Khepera and the two circles (in the Manhattan distance) relevant for changing the state of a previously discovered light source.

6 Report

We developed a simple code to enable the robot to report its findings during the last round using the two LEDs of Khepera.

The signals have the following meaning (legend: ● on, ○ off):

- ● New light source found
- ○ Previously discovered light source recognized
- ● Previously discovered light source turned off
- ● Previously discovered light source turned on again

7 Results

This paragraph is dedicated to the obtained results for both the simulator solution to the dynamical nightwatch's problem and the real-robot implementation (see Fig.11, Fig.12) using the obtained paths of the robot in the respective environments as an evaluation benchmark. We state a very satisfactory behaviour of the simulator solution whereas the real-robot version unveils to be feasible only locally. Note that the compiled C-code of the whole program has 87Kbyte, hence we are using only about one third of the RAM capacities of Khepera's processor.

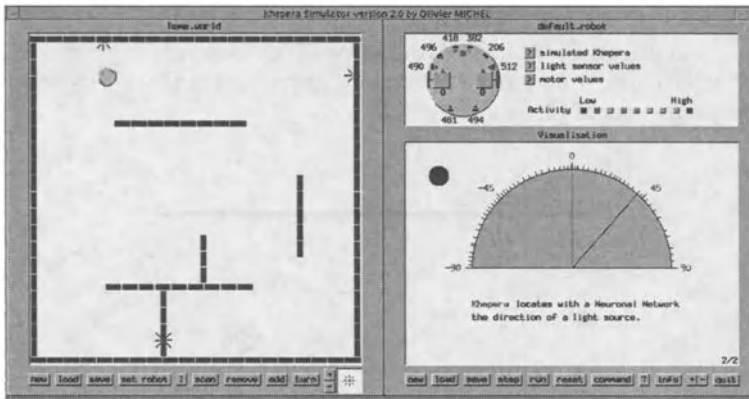


Fig. 11. In the visualisation window of the Khepera simulator [6], the robot's current angle-to-light-source is depicted.

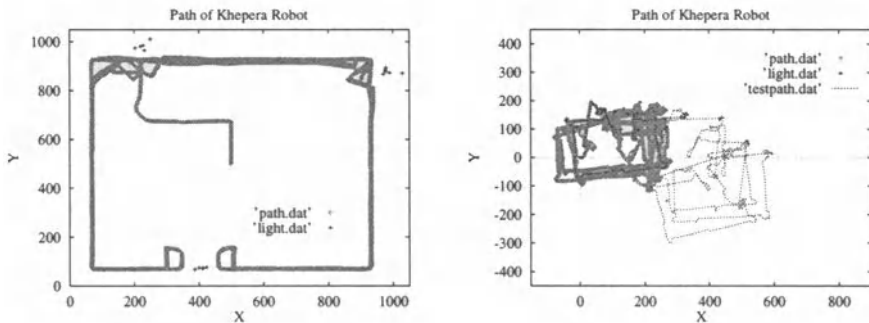


Fig. 12. The left gnuplot shows the path of the simulated Khepera. Note that the small location errors are solely caused by the remaining inaccuracies of the neural network. The right gnuplot shows the path of a real Khepera wherein the bold line represents the path with and the dotted line one without readjustment of the position using the already discovered light source as a landmark.

8 Conclusion and Future Work

Inspired by concepts of neural information processing and the agent paradigm, we implemented a solution to the dynamical nightwatch's problem, both on the simulator and on the real Khepera. In particular, we used an one-dimensional self-organizing feature map for sensor calibration attaining maximum accuracy in average. The simulator solution proofed to be globally reliable, whereas the real-robot implementation had some short-comings due to step-counting errors inducing inaccuracies into the position determination procedure. This was partly cured by taking into account the already registered light sources as local landmarks in order to readjust the current position. The results showed to be satisfactory locally, i.e. when either the robot's environment was restricted to a certain area or the runtime was limited.

Nevertheless, some further research on the topic of navigation is to be done. We hope to enhance the robot's performance considerably by incorporating associative memory and neural classifiers into the control structure. Moreover, it is also envisaged to replace some of the software modules by resource-efficient microelectronic devices as described for example in [9, 10].

References

1. S. Geva, J. Sitte and H. Sira-Ramirez, When are tasks "difficult" for learning controllers?, 1994 World Congress on Computational Intelligence WCCI, July 1994, Orlando, Florida, USA. Proceedings of the IEEE International Conference on Neural Networks, Volume IV, pp.2419-2423
2. P. Maes, Modeling Adaptive Autonomous Agents, Artificial Life Journal, edited by C. Langton, Vol 1, No. 1 & 2, MIT-Press, 1994
3. R. A. Brooks, Intelligence without Representation, Artificial Intelligence, 47, pp. 139-159, 1987
4. R. A. Brooks, Intelligence without Reason, Computers and Thought lecture, Proceedings of IJCAI 91, Sydney, Australia, 1991
5. V. Braitenberg, Vehicles: Experiments in Synthetic Psychology, MIT Press/Bradford Books, 1984
6. O. Michel, Khepera Simulator, version 2.0, 1996, <http://diwww.epfl.ch/lami/team/michel/khep-sim/>
7. K. Malmstrom, L. Munday, J. Sitte, A Simple Robust Robotic Vision System using Kohonen Feature Mapping, Proceedings of the 2nd IEEE Australia and New Zealand Conference on Intelligent Information Systems, pp. 135-139, 1994
8. T. Kohonen, Self-Organization and Associative Memory, Springer-Verlag, 2nd edition, 1988
9. A. Heitmann, J. Malin, C. Pintaske, U. Rückert, Digital VLSI Implementation of a Neural Associative Memory, 6th International Conference on Microelectronics for Neural Networks, Evolutionary & Fuzzy Systems, 24.-26. September, Dresden, Germany, 1997
10. S. Rüping, M. Porrmann, U. Rückert, SOM Hardware-Accelerator, WSOM'97: Workshop on Self-Organizing Maps, June 4-6, pp. 136-141, Espoo, Finland, 1997

CoDi-1Bit : A Simplified Cellular Automata Based Neuron Model

Felix Gers¹ Hugo de Garis¹ Michael Korkin²

¹ATR, Human Information Processing Laboratories
2-2 Hikari-dai, Seika-cho, Soraku-gun, Kyoto 619-02, Japan.
tel. +81-774-95-1079 ,fax. +81-774-95-1008 , flx@hip.atr.co.jp &
degaris@hip.atr.co.jp, <http://www.hip.atr.co.jp/~flx>, &
<http://www.hip.atr.co.jp/~degaris>

² Genobyte Inc. 1319 Spruce St., Suite 210, Boulder, CO, 80302, USA.
tel. + 1 303 545 6790, fax. + 1 303 545 9667, korkin@genobyte.com,
<http://www.genobyte.com>

Abstract. This paper presents some simplifications to our recently introduced “CoDi-model”, which we use to evolve Cellular Automata based neural network modules for ATR’s artificial brain project “CAM-Brain” [11]. The great advantage of CAs as a modeling medium, is their parallelism, which permits neural system simulation hardware based on CoDi to be scaled up without loss of speed. Simulation speed is crucial for systems using “evolutionary engineering” technologies, such as ATR’s CAM-Brain Project, which aims to build/grow/evolve a billion neuron artificial brain. The improvements in the CoDi model simplify it sufficiently, so that it can be implemented in state of the art FPGAs (e.g. Xilinx’s XC6264 chips). ATR is building an FPGA based Cellular Automata Machine “CAM-Brain Machine (CBM)” [13], which includes circuits for neural module evolution and will simulate CoDi about 500 times faster than MIT’s Cellular Automata Machine CAM-8 currently used at ATR.

Keywords

Cellular Automata, Evolutionary Engineering, Evolvable Hardware, Neural Networks, Genetic Algorithms, Genetic encoding, Artificial Brains, Cellular Automata Machine (CAM-8), CAM-Brain Machine (CBM).

1 Introduction

ATR’s CAM-Brain Project aims to build/grow/evolve brain-like-systems, into which human knowledge can be inserted as initial conditions. The brain-like structures are based on cellular automata (CA) which grow inside cellular automata machines (CAMs). Evolutionary engineering techniques, combined with huge computing power, may then be able to evolve such brain-like systems with a level of functionality similar to that of biological brains. The expectation is that

it will be much easier to investigate the behavior of engineering-based brains than biological brains, and that the engineering solutions will reveal insights into the working mechanisms of the evolved systems, which might then be rediscovered in biological brains.

In the CAM-Brain Project, the CAs act as the medium in which to conduct the growth, and in a second step, the neural signaling, of brain-like structures.

To apply evolutionary engineering techniques to the creation of huge neural systems, e.g. an artificial insect brain, we need a simulation tool that has to satisfy two conditions. First of all, it must be possible to simulate most of the important characteristics of biological neural networks. Secondly, the evolution times must be manageably short. The bottle-neck in evolutionary algorithms is the fitness measurement time, i.e. the time needed to simulate the systems. Hence, in order to develop what would presently be considered large-scale systems, on the order of a billion neurons, we believe that scalability by massive parallelism is essential.

With CAs as a simulating tool, the simultaneous demands of generality and scalability can be met. Artificial neurons and neural networks can be modeled at any level of detail, because CA cells can symbolize a molecule in a biological cell membrane, a whole neuron, or any structure in between. Since CAs are only locally connected, they are ideal for implementation on purely parallel hardware. CAs are both structurally flexible and inherently parallel.

However, for parallelization to be advantageous, it is essential that the rules for the underlying CA be simple enough to be stored locally. Indeed, Our CoDi CA based model [11] first grows and then executes neural structures by means of a set of rules small enough to be embodied in large-scale parallel hardware by means of reconfigurable VLSI circuits – Field Programmable Gate Arrays (FPGA's). For simulation purposes, we have executed the model on MIT's Cellular Automata Machine CAM-8 with a 16 bit look-up-table.

This paper summarizes the main ideas and features of the CoDi model, and introduces some simplifications to it that lead to the "CoDi-1Bit" model and its implementation in ATR's FPGA based CAM-Brain Machine (CBM) [13]. The CBM, which is a Xilinx XC6264 chip based hardware device which can evolve 3D CoDi based neural net modules including a complete Genetic Algorithm.

The remainder of this paper consists of the following. Section 2 describes our CoDi-1Bit cellular automaton model's growth and neural signalling phases. Section 3 presents results from some experiments in which simulations of small-scale structures were effected on the CAM8 machine. Section 4 describes some features of the CAM-Brain Machine (CBM), the FPGA machine which at time of writing is under construction. Section 5 discusses the results achieved, and previews the main orientations of our future research.

2 General Description of the CoDi CA

When designing our CA-based neural networks using the new "CoDi-1Bit" model, our objective was to implement them directly in evolvable hardware, e.g. FP-

GAs. Therefore, we tried to keep the CAs as simple as possible, by having a small number of bits to specify the state, keeping the CA rules few in number, and having few cellular neighbours. At this stage, no learning algorithm has been included into the CA, as we rely on evolutionary learning based on a Genetic Algorithm (GA).

We use a von Neumann neighbourhood [2], i.e. a cell in a 3D space looks at its 6 nearest neighbors and at its own state. It should be possible to extend our model to more than 3 dimensions, since the general approach taken is independent of the dimensionality of the CA-space.

We use two different CAs for the growth phase and the signaling phase in our neural networks. The cell interactions are similar in both. The main difference between the two phases, and hence the two CAs, is that the type of a cell can only change during the growth phase, e.g. a blank cell can change to an axon or a dendrite cell. (See later sections for details.) Cell types do not change during the signaling phase, although cells do change their “activity values” and exchange neural signals. (See later sections for details).

Since cell-to-cell interactions are similar in the growth and signalling phases, they can be incorporated into the same CA with a dynamic “phase switch”. This might be useful when a future learning algorithm needs to grow new connections, depending on the activity in already existing axonal or dendritic connections. In the following three subsections (2.1- 2.3) we describe how the CoDi CA can be implemented as a partitioned CA, the CoDi cell-to-cell interaction model itself and the genetic coding, i.e. the initialization, of the CoDi CA.

2.1 Implementation of CoDi as a Partitioned CA

The states of our CAs have two parts, which are treated in different ways. The first part of the cell-state contains the cell’s type and activity level and the second part serves as an interface to the cell’s neighbourhood by containing the input signals from the neighbors. Characteristic of our CA is that only part of the state of a cell is passed to its neighbours, namely the signal and then only to those neighbours specified in the fixed part of the cell state. This CA is called “partitioned” [11], because the state is partitioned into two parts, the first being fixed and the second is variable for each cell (see figure 1).

The advantage of this partitioning-technique is that the amount of information that defines the new state of a CA cell is kept to a minimum, due to its avoidance of redundant information exchange.

For a CA implementation on a look-up-table (LUT) machine such as MIT’s CAM-8 or in FPGAs, it is crucial to minimize the amount of cell-to-cell communication, to keep the number of states in the CA manageable.

This is a major advantage for the implementation of CAs in parallel hardware. Large LUTs or complex programs cannot be stored locally, and hence are non-parallel. Using the partitioning-technique, improves the practicability of a neural network simulated directly in parallel hardware.

The next few subsections detail the new CoDi model and its implementation as a partitioned-CA.

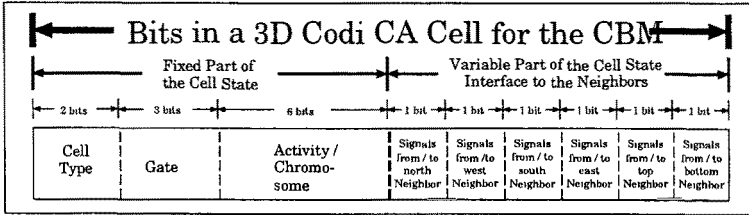


Fig. 1. State representation in the CoDi-1Bit model. During the growth phase 6 of the bits are used to store the chromosome's growth instructions. The same 6 bits are later used to store the activity of a neuron cell during the signaling phase.

2.2 CoDi Cell Interaction

The CoDi model is named after the essential features of its axonal and dendritic cell behavior, i.e. to “collect” or to “distribute” the cellular values from/to the CA cells’ neighbors. The CoDi model works with four basic types of CA cells, neuron body, axon, dendrite and blank.

- Blank cells represent empty space. They do not participate in any cell interaction during the signaling of the neural network.
- Neuron bodies consist of one CA cell. The neuron body cells collect neural signals from the surrounding dendritic cells and apply an internally defined function to the collected data. In the CoDi model the neurons sum the incoming signal values and fire after a threshold is reached. This behavior of the neuron bodies can be modified easily to suit a given problem. The output of the neuron bodies is passed on to its surrounding axon cells.
- Axonal cells distribute data originating from the neuron body.
- Dendritic cells collect data and eventually pass it to the neuron body.

These types of cell-to-cell interaction cover all kinds of cell encounters. Axonal and dendritic cells do not need to know anything about the cell type of their neighbors. They behave in the same way towards any type of neighbor. Therefore it is not necessary to distinguish between the different cases of cell-to-cell interactions, e.g. axon-neuron, axon-dendrite, dendrite-dendrite etc, so that designing separate rules for each case is unnecessary. The two basic interactions cover every case, and they can be expressed simply, using a small number of rules.

To make axonal and dendritic signal trails (which can be dense in the CA-space) without blockers or sheath cells as in earlier models [4], the four basic cell types need further specification. To the 2 bits which specify the basic cell type, we added 3 more bits to set the “gate” of a 3D CA cell. (See figure 1.) A neuron cell uses this gate to store its orientation, i.e. the direction in which the axon is pointing.

In an axon cell, the gate points to the neighbor from which the neural signals are received. An axon cell accepts input only from this neighbor, but makes its own output available to all its neighbors. In this way axon cells distribute information. The source of information is always a neuron cell. Dendritic cells collect information by accepting information from any neighbor. They give their output, (e.g. a Boolean OR operation on the binary inputs) only to the neighbor specified by their own gate. In this way, dendritic cells collect and “sum” neural signals, until the final sum of collected neural signals reaches the neuron cell.

We will see that each axonal and dendritic cell “belongs” to exactly one neuron cell. This configuration of the CA-space is guaranteed by the preceding growth phase (section 2.3). Figure 2 shows the signaling phase with the CoDi-model. Each neuron is given two dendritic trees and two axonal trees. One of the axonal trees distributes inhibitory signals of the neuron, the other distributes excitatory signals. A neuron has two dendritic trees to maximize the amount of information that can be passed to the neuron body in each time step.

We use binary neural signals which can fill a trail contiguously. The direction of motion is given by the underlying signal trails.

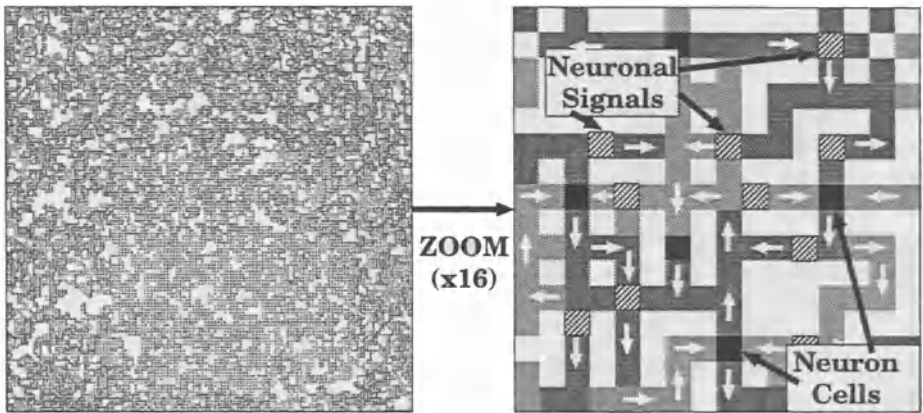


Fig. 2. (Left) A CA-space with $256 \cdot 256$ cells in the signaling phase with the CoDi-model. (Right) A zoom (x16) with five neuron cells (black) with two dendrites and two axons each. The arrows inside the axonal (dark grey) signal trails and dendritic (light grey) signal trails indicate the direction of information flow during the signaling phase. In the growth phase the signals in the dendritic trails travel the other way, because they are emitted by the neuron. To improve the visualization of neural trails, the underlying chromosome is gridded with a checker-board pattern of “grow-straight” instructions. This causes the neural trails to grow on a 2-cell grid.

Synapses The CoDi model does not use explicit synapses, because dendrite cells that are in contact with an axonal trail (i.e. have an axon cell as neighbor) collect the neural signals directly from the axonal trail. This results from the behavior of axon cells, which distribute to every neighbor, and from the behavior of the dendrite cells, which collect from any neighbor.

However axon-axon and dendrite-dendrite interactions of cells that do not belong to the same neuron are not possible. This is because axon cells take their information only from the neighboring axon cell that is closer to the “source” neuron on their axonal trail. Dendritic cells behave in the opposite way, by passing signals to the dendritic neighbor that is closer to the neuron or to the neuron cell itself. This configuration of the CA-space results from the neural growth phase.

These two simple behaviors of axon and dendrite cells (i.e. collect and distribute), which are exact opposites, allow axonal and dendritic trails to be dense in the CA-space, without interacting in an unintended way. The CoDi-model remains functional even if every cell in the CA-space is used.

The strength of a neuron-neuron connection (a synapse) is represented by the number of their neighbouring axon and dendrite cells. The exact structure of the trails and the position of the axon-dendrite neighbor pairs determine the time delay and strength (weight) of a neuron-neuron connection. This principle implies that a single neuron-neuron connection can consist of several synapses with different time delays with independent weights.

2.3 Genetic Encoding, Initialization and Growth of the CoDi CA

Biological Systems have to self-assemble. The genetic encoding of biological brains is indirect, sophisticated and incomplete. There is not enough genetic information in vertebrates to encode the detailed connectivity of their brain circuits.

However, these characteristics of biological brain growth need not apply to electronic brains. Electronic circuits do not have to grow. They can be copied into hardware as initial settings. The genetic encoding can be direct. For example, a neural circuit can be stored as a spatial pattern in bitmap format. The genetic information can be complete, so that a given chromosome always produces exactly the same neural circuit and vice versa.

For the CoDi model, we have kept the biological feature of growing a system. We chose a genetic encoding that is indirect, but very different from biological genetic encoding. We wanted our encoding to be one to one, so that a given chromosome leads to exactly one neural network. In this way we ensure that a chromosome has a unique fitness value and hence do not have to spend time calculating an average fitness for each chromosome.

The advantage of a grown system is that the structural integrity of the basic components (in our case the neurons) is preserved in the presence of mutations (or alternations in the chromosome). All non-blank cells in the CA-space can be assigned to exactly one neuron, (the neuron from which they were grown). This is the case for any chromosome. Though the connectivity of the neural network

may be unsuitable for a given problem after the growth according to a arbitrary chromosome, a consistent structure of the neural network is always guaranteed. The importance of this feature becomes apparent when we compare a) the effect of genetic operators on a neural system that grows and b) neural systems that are mapped directly to their chromosomes. In the first case a mutation leads to a small, local change in the direction of a signal trail. In the case of direct mapping, a mutation and the resulting local change of a cell, which may belong to a signal trail of a complex axonal tree, can lead to disconnection of a large amount of circuitry. This circuitry would then be disconnected from its source neuron, and loose the integration into the basic structure of the neural network model. The situation gets worse, if we look at the way new neural connections come into being. The probability that a new and useful signal trail evolves if all cells in the CA-space are mutated independently (as is the case for direct mapping), are very small. However a mutation in a grown neural network (for example from “turn right” to “split right”) leads to the growth of a new neural signal trail that is consistently integrated into the structure of the source neuron and hence the whole network. This feature of a grown system reduces the search space for possible CA-patterns to the patterns that represent consistent neural structures. The selection of an indirect genetic encoding is a logical consequence of the decision to grow our networks.

The CoDi model uses a distributing chromosome to encode its structure [11]. The chromosome is initially distributed throughout the CA-space, so that every cell in the CA-space contains one instruction of the chromosome, i.e. one growth instruction, so that the chromosome belongs to the network as a whole. This technique has several advantages compared to previous approaches [4].

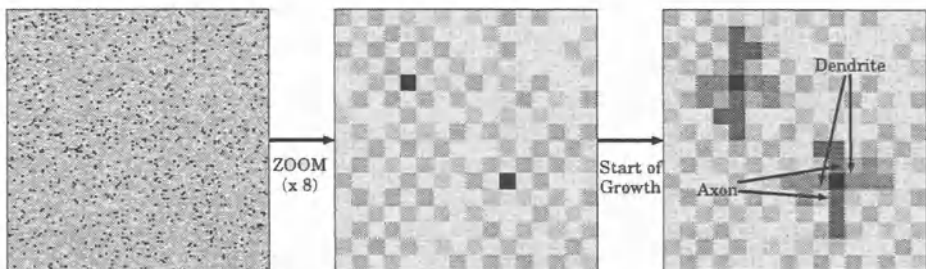


Fig. 3. (Left) An initial pattern of $128 \cdot 128$ 2D CA cells with a neuron-density of 5%. The (black) neuron cells are randomly positioned. (Middle) A zoom (x8) with two neuron cells on a random (2-cell gridded) chromosome. (Right) The beginning of the growth phase, after three CA-steps.

The neuron bodies can be positioned arbitrarily in the CA-space. (See figure 3.) Growth signals are passed to the direct neighbors of the neuron cell according to its chromosome information. The blank neighbors, which receive a neural

growth signal, turn into either an axon cell or a dendrite cell. (See figure 3 (right).) In the earlier CoDi model [11] the growth signals (and signals in general were 2 bits wide), so that they included information containing the cell type of the cell that was to be grown from the signal. To reduce the Codi model to one bit signals we had to find a different way to pass this information. The solution is to work with an oscillating CA, i.e. the growth signals are time multiplexed. In a given time-step only axons pass their growth signals and all blank cells that receive signals interpret them as “grow an axon” signals, whereas in the next time-step dendrites grow. Different ratios of axon to dendrite time-steps are possible, e.g. 3 axonal time-steps on one dendritic time-step.

The cells just grown, then receive the next growth signal from the neuron body cell, which sends them continuously. To decide in which directions axonal or dendritic trails should grow, the grown cells consult their chromosome information which encodes the growth instructions. These growth instructions can have an absolute or a relative directional encoding. An absolute encoding masks the 6 neighbors (e.g. directions) of a 3D cell with 6 bits. The cell contiguous to the face (direction) whose bit is set, will itself become an axon cell or dendrite cell at the next clock tick. For a relative encoding, 5 bits are sufficient, as the direction from which the initial growth signal comes is known to a cell (be stored in its gate field). The growth information can be interpreted relative to this direction, so one bit less information is needed. The absolute encoding has the advantage of being easier to formulate in logic equations and we use it in the FPGA design of the CBM (see section 4). A relative encoding saves one bit in the cell state, which can be crucial for LUT machines. To avoid a 17 bit LUT we used the relative encoding on the CAM-8 (which offers a 16 bit LUT). We will use the relative encoding to illustrate the growth phase with a distributed chromosome.

5 bits of chromosome information in each cell offer $2^5 = 32$ different growth instructions, which are : (“block growth”, 5 simple turns, 10 simple splits, 10 3-direction splits, 5 4-direction splits and “grow in all directions”), see figure 4). Using its chromosome instructions and knowing from which direction the growth signals come, a cell can determine the neighbors to which it transmits the incoming growth signals. (See RHS of figure 4.) If, for example, a growth signal comes from the left neighbor to a cell whose chromosome instruction is “split right”, it will pass the growth signal to its right and bottom neighbors. After a cell is grown, it accepts growth signals only from the direction from which it received its first signal. This “reception direction” information is stored in the “gate” position of each cell’s state.

Figure 5 shows two cut-outs of neural networks at the end of a growth phase. The growth phase stops when growth is saturated, i.e. when no more trails (as specified by the instructions of the network’s chromosome) can be formed.

In most of our figures, we have used gridded neural structures, in order to render the axonal and dendritic trails more visible. To force neuron trails to grow on a 2-cell grid, the underlying chromosome is initialized in a “checker-board” pattern. The “black” squares of the checker-board are filled with “grow-straight”

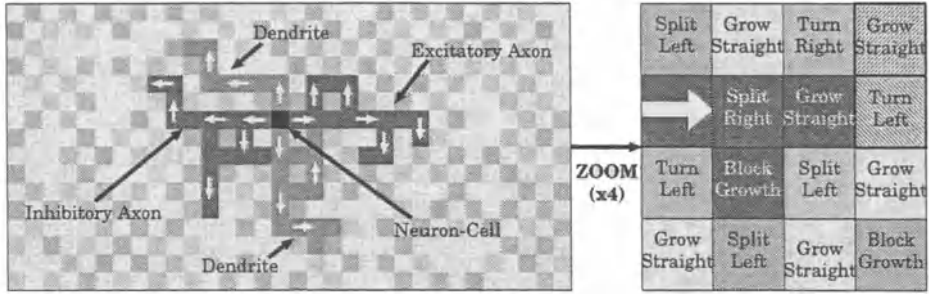


Fig. 4. (Left) A neuron in the CoDi-model with two dendrites and two axons. The arrows inside the axonal and dendritic signal trails indicate the direction of information flow during the growth phase. The neuron grows on a random (2-cell gridded) chromosome in the CA-space. (Right) A signal trail grows according to the underlying chromosome instructions. The trail continues to grow over the two striped cells in the next two time steps. The branch of the trail that leads downward is blocked by the blocker instruction in the chromosome.

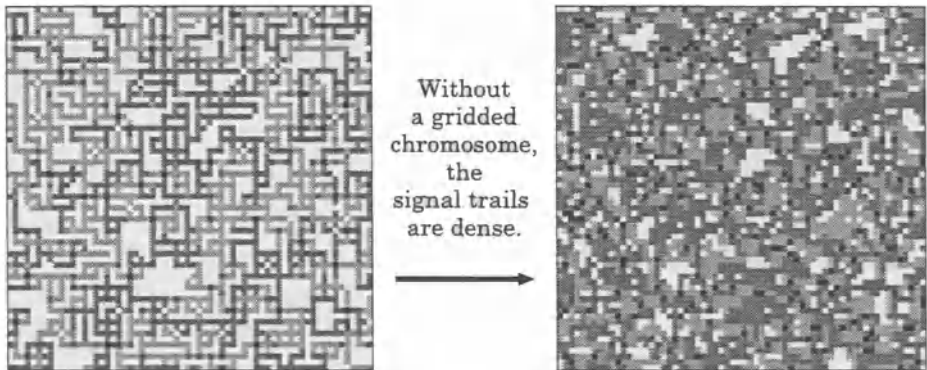


Fig. 5. Two cut-outs of neural networks at the end of the growth phase. (Left) A network grown from a gridded chromosome. (Right) A network grown without gridded chromosome instructions.

instructions and the white squares are filled with arbitrary growth instructions. Since gridding is only a visualization trick, an unconstrained evolutionary algorithm can grow trails which are dense (the RHS of figure 5 shows an example of an ungridded neural network).

Beside the “turns”, “splits” etc. instructions, there are “block-growth” instructions in the chromosome. They cause a cell in an axonal or dendritic trail to block the passage of the incoming growth signals to its neighbors, so the

branch of the trail stops growing. With a high density of blocker instructions in a chromosome, an area in the CA-space can be transformed into a more or less impenetrable “wall”. Figure 6 shows two “walls” with different densities of

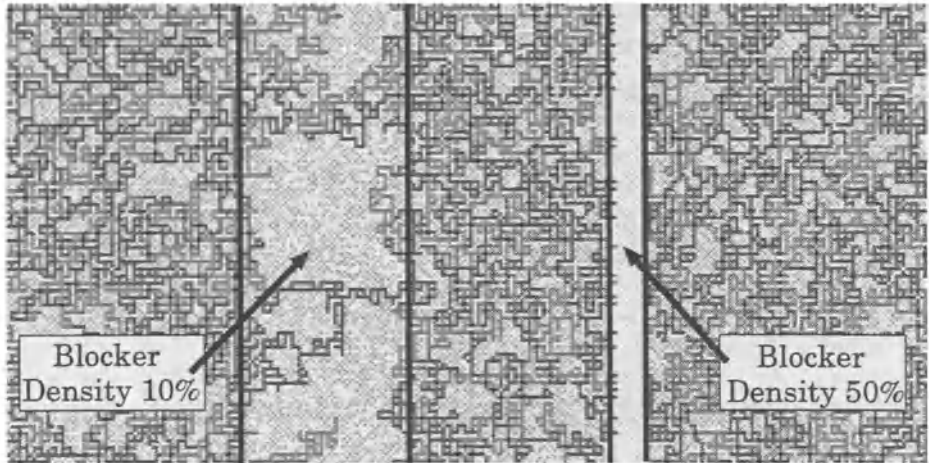


Fig. 6. The use of “walls” to modularize neural circuits in the new CAM-Brain model. The density of blocker instructions on the chromosome determines the probability of trail-growth through a wall and hence the penetrability of the wall.

blocker instructions in the two regions. A combination of walls can partition a neural network into more or less independent modules.

The grown circuitry and the chromosome are connected locally, so that a local learning algorithm can make local changes in the phenotype and the genotype of the network. This enables the use of Lamarckian evolution (with local learning algorithms) in addition to Darwinian evolution.

The distributed chromosome technique of the new CAM-Brain model makes maximum use of the available CA-space and enables the growth of any type of network connectivity. The local connection of the grown circuitry to its chromosome, allows local learning to be combined with the evolution of grown neural networks.

2.4 CoDi on the CAM-Brain Machine (CBM)

Why is the simplicity of our CA-model important? Our long term aim is to build CAs to model brain-like structures. Biological brains, the highly optimized results of millions of years of evolution, are obvious sources of inspiration for our work.

Even the tiny brains of insects consist of about a million neurons with complex connectivity. In order to model a 3D neuron with CAs in a way that preserves the essential functionality of real neurons, we estimate that we need at least a cube of 10^3 CA cells. So the minimum size of CA-space to model an insect brain is 10^9 cells. To update a billion CA cells at acceptable speeds will require the use of parallel hardware. The ideal case for CA updating is to give each CA cell its own processor, so that the whole CA-space is updated, in parallel, in one clock cycle.

To build parallel hardware with 10^9 independent CA processors (for a small mammalian brain it would be 10^{12} CA processors), the components must be very simple in their internal logic, with very little memory. Large look-up-tables (LUTs), as are common in traditional CA implementations, will be impossible. Hence, for a CA to be executable on parallel hardware, the rules must be few enough to be expressible within the limited number of logic gates per CA processor inside the parallel hardware. For this reason we tried to keep our new CAM-brain model as simple as possible.

With the CoDi-1Bit model, illustrated in this paper, it is possible to implement one CA cell with about 150 Xilinx XC6264 "functional units". This inspired us to build our CAM-Brain Machine (CBM)[13].

3 Evolvability Simulation Results

This section presents some results which were undertaken to show that the CoDi-1Bit model is actually evolvable. It also briefly discusses representation issues.

Three experiments were performed to show that it would be worthwhile to invest in the financial cost to build a fully hardware implemented model of the CoDi-1Bit design in the CAM-Brain Machine (CBM). Note that the emphasis here is NOT on the intrinsic interest of the evolved functionality, but rather the evolvability of the model.

3.1 The Comparator

The first software simulation experiment (on a Sun Sparc 10 workstation) aimed to evolve a binary comparator, i.e. two 3 bit numbers were input at the face of a cube of $16*16*16$ CA cells. At the opposite face was an output whose binary value should be 1 if the input signal A was greater or equal to the input signal B, otherwise 0. The signal phase was run over 50 clockcycles, with only the output from the final 20 clocks being used in the fitness definition (to allow time for the signal to circulate through the circuit). If the desired output actually occurred, the fitness value incremented a point. The total points over all 64 cases (8 cases of A, 8 cases of B) were divided by $64*20$. The population size was 10. The chromosome length was about 20K bits. The fitness rose to about 90 percent after about a day's run, and then very slowly increased, but our human patience ran out. We were not prepared to run the experiment for days. We came to the conclusion *that to truly test the evolvability of the CBM, we would need a CBM.*

3.2 The Timer

The next two experiments were "single case" types, so that the fitness measurement time would be within human patience limits. The second experiment was to evolve a timer, i.e. the output signal should be 0 for the first 40 clocks (of 100 total), the next 30 clocks should have a 1 putput, and the remaining 30 clocks should be 0. The inputs were the same 6 inputs as before, but all set at 1. The same output point was used. The 16*16*16 cube was wrapped around on the side edges, but not at the top and bottom. The fitness definition was chosen so that if a desired 1 occurred, the fitness value incremented 5 points, otherwise, incremented zero points. If a desired 0 occurred, the fitness incremented by 1 point. The total over the 100 clocks was divided by $((40+30)*1)+(30*5) = 220$. After about 100 generations, the desired output evolved perfectly, i.e. fitness was 100 percent. This first success showed that the CoDi-1Bit model could do at least this much.

3.3 The Full Sinus Wave

The third and final experiment undertaken was to evolve a sinusoid output. The inputs were the same as in the second experiment. There was a 2 bit output with the following representation. If the first bit B1 was a 1 and the second bit B2 was a 0, then a fictitious counter was incremented by 1. If B1=0 and B2=1, then the counter was decremented by 1. Other cases left the counter unchanged. An arbitrary sinusoid amplitude and wavelength target shape was chosen, that the counter is supposed to follow as closely as possible, with the counter value being the vertical "y" axis, and the clock count being the horizontal "x" axis. The circuit should evolve so that the outputs B1 and B2 change over time, such that the counter gives the appropriate "y" values over time.

The fitness value was defined to be the inverse of the sum of the squares of the differences between the desired y values and the actual counter values. An incremental or stepwise evolutionary approach was taken that de Garis developed in 1990 [de Garis 1990], namely that the evolution used several intermediate fitness definitions, where the resulting population of GA chromosomes evolved with fitness definition FD1, became the starting generation of chromosomes with fitness definition FD2, etc. Often this approach gives better results than using only one fitness definition. In the case of the sinusoid, at first a quarter wavelength was evolved (where the target sinusoid had an amplitude of 20 and wavelength of 200 and the whole curve shifted up by 30. The fitness measurement started after 20 clocks, at which point, the counter was set initially at the value 30. After about 100 generations, the evolved output followed the desired target within a few percent. The resulting population became the starting population for a second evolution phase where the aim was to evolve a half sinusoid. Again, after a day, and several hundred generations, the actual curve followed the target curve within a few percent. Two further steps were used, to evolve a three-quarter wave and a full wave. The three-quarter wave evolved as well as the previous two, but the full wave did not quite get the actual curve to return to the "base line"

(set at 30). It looked as though the limits of the evolvability of the 4K CA cell module had been reached. An alternative fitness definition which weighted the later clocks was attempted, but had no effect. Fig.7 shows the result of the full sinusoid, both target and actual output.

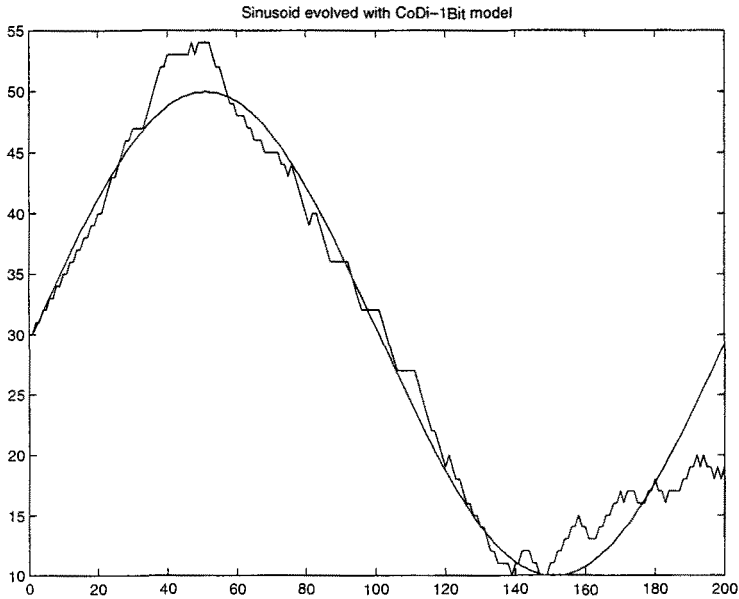


Fig. 7. Sinus wave generator evolved by CoDi model simulation on CAM-8

These few results were thought to be enough to show that the CoDi-1Bit model is evolvable enough to proceed with the construction of the CAM-Brain Machine.

3.4 Representation Issues

Finally, something needs to be said about the representation of the input and output signals. Of course, one is free to interpret these binary inputs and outputs as one chooses, e.g. as binary numbers, as biological "spikes" in a convolution/deconvolution model, as incrementers/decrementers, etc. The issue of representation is an important one which needs to be given a lot more thought in future CBM work.

4 CBM Architecture

The CAM-Brain Machine (CBM) is a special purpose hardware device based on Xilinx's programmable hardware (FPGA) chips (XC6264). The CBM is not just a faster version of the CAM-8 machine. It is based on the CoDi-1Bit neural model and introduces a number of highly specialized features. In this respect, CBM is not intended for general purpose cellular automata applications, such as hydrodynamic modeling, lattice gas simulation, or thermal stress analysis. The CBM consists of the following five major blocks :

- Cellular Automata Module
- Genotype/Phenotype Memory
- Fitness Evaluation Unit
- Genetic Algorithm Unit
- Module Interconnection Memory

Each of these blocks are discussed in detail below.

4.1 Cellular Automata Module

The cellular automata module is the hardware core of the CBM. It is intended to accelerate the speed of brain evolution through a highly parallel execution of cellular state updates. The CA module consists of an array of identical hardware logic circuits or cells arranged in a 3D structure of two modules of $16*16*16$ (4096 cells) or one module of $16*16*32$ (8192 cells). Cells forming the top layer of the module are recurrently connected with the cells in the bottom layer. A similar recurrent connection is made between the cells on the north and south, and the east and west vertical surfaces. Thus a fully recurrent toroidal cube is formed. This feature allows a much higher axonal and dendritic growth capacity by effectively doubling each of the three dimensions of the block.

4.2 Universal Cell

Each hardware logic cell contains circuitry to implement several different CBM functions, neuron, axon, dendrite and blank cell. Depending upon a particular chromosome of the neural module, each of these "universal" cells will perform one of the 4 possible functions in a given module during the signaling phase and may change its type from blank to an axon or a dendrite type during the growth phase. A blank cell can also be initialized to function as a neuron before the beginning of the growth phase.

4.3 CA Module Implementation in Hardware

The CA module is implemented with new Xilinx FPGA devices XC6264. These devices are fully and partially reconfigurable, feature a new co-processor architecture with data and address bus access in addition to user inputs and outputs,

and allow the reading and writing of any of the internal flip-flops through the data bus. An XC6264 chip contains 16384 logic function cells, each cell featuring a flip-flop and some Boolean logic capacity, capable of toggling at a 220 MHz rate. Logic cells are interconnected with neighbors at several hierarchical levels, providing identical propagation delay for any length of connection. This feature is very well suited for a 3D CA space configuration. Additionally, clock routing is optimized for equal propagation time, and power distribution is implemented in a redundant manner. To implement the CA module, a 3D block of identical logic cells is configured inside each XC6264 device, with CoDi specified 1-bit signal buses interconnecting the cells. Each XC6264 device, available in a QFP304 (quad flat pack) package, has up to 180 input/output pins. The number of available pins limits to 180 the maximum number of external connections to be made when interconnecting several XC6264 chips to form the 4096-cell space. Another limitation is imposed by the amount of logic needed to implement all the necessary functions in the CoDi cells. With these limitations in mind, the best arrangement for each XC6264 part is a block of $4*4*8$ cells (total of 128), which results in 160 externally connected cell surfaces, 4 surfaces of $4*8$ and 2 surfaces of $4*4$ cells. To assemble a 4096-cell module, 32 XC6264 chips are needed, configured as $2*4*4$ chips on 4 boards (dual surface mounted). The amount of hardware logic available for each CA cell is $16384/128 = 128$ Xilinx function blocks. This amount is sufficient for implementing all necessary CoDi model functions for growth and signaling, cell interconnection, and other requirements. All cells exchange signals and perform their respective functions synchronously at the rising edge of the master clock running at 30 MHz. The clock rate of 30 MHz (33 ns cycle) permits eleven layers of logic between reregistration to be used for CA cell circuitry design. (Each logic layer's propagation delay is 2.5-3 ns). During every clock cycle (33 ns), all 4096 cells are updated. This yields a speed increase of 614.4 times over the CAM-8 speed for the 1-bit 3D CoDi cells update rate of 200 Mcells/s. This number demonstrates "raw" speed increase, which is somewhat reduced by the module reconfiguration time. (See below).

4.4 Power Consumption and Heat Management

Dynamic power consumption of the module hardware is proportional to the clock rate, the number of flip-flops toggling simultaneously, and the capacitive loading of all the connections inside and outside the FPGAs. Static power consumption is expected to be negligible. Calculations yield a dynamic power consumption of the order of 4 watts for each XC6264 device. With a total of 32 FPGAs, the power consumption of the module is 128 watts (25.6 amperes). It is expected that the rest of the CBM circuitry will additionally require 100 watts. Given the estimated power dissipation, thermal management of the device can be accomplished using standard forced airflow techniques.

4.5 Module Inputs and Outputs

Although the module architecture is fully recurrent, some of the recurrent connections between top and bottom, east and west, etc., are reserved for external inputs and outputs. There are 8 cells on each surface of the module reserved to be used as external inputs to the module (a total of 48) and another 8 cells on each surface to be used as external outputs (a total of 48). Input and output locations are evenly spread over each surface of the module cube. In order to establish connections to external signals, an input cell must be evolved as either a dendrite or a neuron pointed inside. An output cell must be evolved as an axon or a neuron, pointed outside.

4.6 Genotype and Phenotype Memory

There are two modes of CBM operation, namely evolution mode and run mode. In the evolution mode, memory space is used to store the chromosome bitstrings of the evolving population of modules (module genotype). For a module of 4096 cells there are 20480 bits of memory needed. For each module the genotype memory also stores information concerning a maximum of 128 neurons locations and orientations inside the module. This includes, X, Y, Z coordinates (12 bits), gating code (3 bits), input functions (excitatory/inhibitory) (5 bits), giving a total of 20 bits per neuron. Thus, the total chromosome memory requirement for one module is $20480 + 20 * 128 = 2880$ bytes.

In run mode, memory is used as phenotype memory for the evolved modules. The phenotype data describes grown axonal and dendritic trees and their respective neurons for each module. For phenotype storage there are 6 bits required per cell, i.e. for gating (3 bits), cell type (2 bits), and signal value (1 bit). Neuron cells additionally require 4 bits for the accumulator value stored. Phenotype data is loaded into the CA module to configure it according to the evolved function. Every 1024 cell board on a 4 board module will have 8 Mbytes of its own genotype/phenotype memory, with four 32-bit data buses for parallel access. 32 Mbytes of memory can store over 10 thousand modules at a time. This amount is sufficient to evolve 10 thousand modules at high speed, or to run a simulated brain with one million neurons. A large memory will be based in the main memory of the host computer (Pentium-Pro) connected to the CBM through a PCI bus, capable of transferring data at 132 Mbytes/s. Genotype/phenotype memory is connected to the hardware CA module and is used for rapid reconfiguration of the neural module by loading new chromosome data (or phenotype data) into the hardware registers of each cell through the XC6264 data bus access. Thus the fast hardware for the CA module is time- multiplexed between multiple neural modules in a large brain.

4.7 Hardware Reconfiguration Speed

The module reconfiguration speed is limited by the speed of the XC6264 data bus access and the width of the memory data bus. Each XC6264 device is capable

of accessing 32 internal flip-flops at a 25 MHz rate, which results in 150 million 5-bit cell reconfigurations per second. Each of the 2-chip sections on each of the four boards has its own memory bus. During parallel access to 16 FPGA's a rate of 2.4 billion cell reconfigurations per second is achieved. Thus, a 4096-cell FPGA hardware module can be reconfigured in 1.7 microseconds. There is a two-stage pipelined configuration register in every cell which permits the loading of the configuration data of a new module (phenotype and genotype) while running a previous module. At the end of the signaling or growth phase, the new configuration is instantly available. This feature guarantees continuous running of the CA module at full speed with practically no interruptions for reloading.

4.8 Overall CBM Speed Estimate

The goal of the CBM architecture is to exploit the full theoretical speed of parallel cell update, equal to 122.88 billion cells per second, or 614.4 times faster than the CAM-8. In order to achieve this, each module must be run for at least the time needed to load a module's configuration data (1.7 microseconds) which is equivalent to 51 update cycles. Also, a dual buffered set of signal FIFOs (see sections 3.3 and 3.5) will be provided to avoid interruptions of input/output signal data reloading. However, with some unavoidable overhead, the overall speed is expected to be around 110 billion cell updates per second.

4.9 Fitness Evaluation Unit

When a useful module is being evolved, each instance of a module must be evaluated in terms of its fitness for a targeted task. During the signaling phase, each module generates a sequence (an array) of 48 output signals, or vectors, which is compared with a target array in order to guide the evolutionary process. This comparison gives a measure of performance, or fitness, of the module. Fitness evaluation is supported by a hardware unit which consists of an input vector array stack, a target vector array stack, and a fitness comparator. Both stacks are 6-byte wide SRAMs (FIFOs) storing up to 2048 input and target vectors each to support the signaling phase of up to 2048 cycles. During each clock cycle an input vector is read from its stack and fed into the module's inputs. At the same time, a target vector is read from its stack to be compared with the current module output vector by the fitness evaluation unit. The fitness comparator computes a Hemming distance between each output vector and a corresponding target vector, and accumulates the result for the whole duration of the signaling phase. At the end of the signaling phase, a final measure of the module's fitness is instantly available. Multiple target and input arrays are stored in the host computer memory.

4.10 Genetic Algorithm Unit

To evolve a module, a population of 100 modules is evaluated by computing every module's fitness measure, as described above. The ten best modules are

then selected for further reproduction. A hardware support will be provided to store and update a "current best ten" list. This list is an array of the current ten best module numbers. Each number is an address of the module chromosome in the chromosome memory. After each generation of modules, the ten best are mated and mutated to produce 100 offspring modules to become the next generation. Mating and mutation is performed by the host computer software on the chromosome memory. Because this process can be performed in parallel with the module evaluation, and only takes place once in a generation, it is expected that there will be no significant slowdown in the evolutionary process.

4.11 Module Interconnection Memory

In order to support the run mode of operation, which requires a large number of evolved modules to function as an artificial brain, a module interconnection memory is provided. It consists of an output vector array stack, similar to the input array and target stack, and a larger memory for inter modular pathway storage. When each module of a large brain is configured in the CA hardware core (by loading its phenotype), an input stack is loaded with an array of input vectors. These vectors are previously stored output vectors recorded during the signaling phase of other modules, connected to this module. A large module interconnection memory will store the connection map and the current state of signals "traveling" between modules. There will be software and hardware support provided for combining output arrays from up to 8 modules, to be used as inputs for one module. In addition, an externally connected set of inputs and outputs is provided in hardware, which will allow the reception of signals from external sensors and the sending of signals to external effectors for robotic control. When running a simple million-neuron brain which contains ten thousand 100-neuron modules, the CBM is capable of updating each module for 100 cycles at the rate of about 27 times per second, allowing real-time control of robotic devices.

5 Conclusions

This paper presented a simplification of the CoDi model (called "CoDi-1Bit") for the evolution of neural structures, based on cellular automata. We reduced the signals to one bit with an oscillating cellular automata. This simplification made it possible to implement CoDi-1Bit with about 150 "functional units" of a Xilinx XC6264 chip per CA cell. This led to the development of the CAM-Brain Machine (CBM), whose architecture is also described. Hence, with the new CoDi model, it will be possible to evolve neural structures directly in hardware.

References

1. Toffoli, T. & Margolous, N. '*Cellular Automata Machines*', MIT Press, Cambridge, MA , 1987.
2. von Neumann, J. '*Theory of Self-Reproducing Automata*', ed. Burks A.W. University of Illinois Press, Urbana , 1966
3. Codd, E.F. '*Cellular Automata*', Academic Press, NY 1968.
4. de Garis, H. '*CAM-BRAIN: The Evolutionary Engineering of a Billion Neuron Artificial Brain by 2001 which Grows/Evolves at Electronic Speed Inside a Cellular Automata Machine (CAM)*', in '*Towards Evolvable Hardware*', Springer, Berlin, Heidelberg, NY , 1996.
5. Lloyd, S. '*A Potentially Realizable Quantum Computer*', Science **261**, 1569-1571 1993.
6. Koza, J.R. '*Genetic Programming: On the Programming of Computers by the Means of Natural Selection*', Cambridge, MA, MIT Press , 1992
7. Koza, J.R. & Bennet, F.H. & Andre, D. & Keane, M.M. '*Toward Evolution of Electronic Animals Using Genetic Programming*', ALife V Conference Proceedings, MIT Press , 1996.
8. Sipper, M. '*Co-evolving Non-Uniform Cellular Automata to Perform Computations*', Physica D **92**, 193-208 1996.
9. Carter, F. L. '*Molecular Electronic Devices*', North-Holland, Amsterdam, NY, Oxford, Tokyo , 1986.
10. Gers, F. A. & de Garis, H. '*Porting a Cellular Automata Based Artificial Brain to MIT's Cellular Automata Machine 'CAM-8'*', SEAL'96 Conference Proceedings S7-3 , 1996.
11. Gers, F. A. & de Garis, H. '*CAM-Brain: A New Model for ATR's Cellular Automata based Artificial Brain Project*', ICES'96 Conference Proceedings S7-5 , 1996.
12. Margolus, N. '*Crystalline Computation*', (Preprint).
13. Korkin M. & de Garis H. & Gers F.A. & Hemmi H. '*CBM (CAM-Brain Machine) : A Hardware Tool which Evolves a Neural Net Module in a Fraction of a Second and Runs a Million Neuron Artificial Brain in Real Time*', Genetic Programming Conference, July 1997, Stanford, USA , 1997.

Million Module Neural Systems Evolution

The Next Step in ATR's Billion Neuron Artificial Brain ("CAM-Brain") Project

Hugo de GARIS(1) Lishan KANG (2) Qiming HE (2),
Zhengjun PAN (2) Masahiro OOTANI (3)
Edmund RONALD (4)

(1) ATR Human Information Processing Research Laboratories,
2-2 Hikaridai, Seika-cho, Soraku-gun,
Kansai Science City, Kyoto-fu, 619-02, JAPAN.
degaris@hip.atr.co.jp <http://www.hip.atr.co.jp/~degaris>

(2) Key State Laboratory of Software Engineering,
Wuhan University, Wuhan 430072, PRC.
kang, heqm, zjpan@rjgc.whu.edu.cn <http://www.rjgc.whu.edu.cn>

(3) Electrical and Electronic Engineering Department,
Toyohashi University of Technology,
1-1 Hibarigaoka, Tenpaku-cho, Toyohashi-shi, Aichi-ken, 441, JAPAN.
ohtani@icg.eee.tut.ac.jp <http://icg.eee.tut.ac.jp/~ohtani>

(4) Centre de Mathematiques Appliquees
Ecole Polytechnique, 91 128 Palaiseau, France.
eronald@cmapx.polytechnique.fr

Abstract. This position paper discusses the evolution of multi-module neural net systems, where the number of neural net modules is up to ten million (i.e. an "artificial brain"). ATR's "CAM-Brain" Project [de Garis 1993, 1996] has progressed to the point where it is technically possible (using a new FPGA (Field Programmable Gate Array) based evolvable hardware (EHW or E-Hard) system to be completed by the spring of 1998 [Korkin & de Garis 1997]) to begin to evolve and build an artificial brain containing 10,000 neural net modules. This development raises the prospect that within a few years these numbers will rapidly increase. This paper introduces some issues that such massive system-building will generate. The immediate question is "What should we evolve?" This paper presents some suggested evolvable system targets containing N neural net modules, where $N = 100; 1000; 10,000; 100,000; 1,000,000; 10,000,000$ with an emphasis on the $N = 100$ case, for purposes of illustration. The issues involved are not only of a conceptual and evolutionary engineering nature, but (when N is large) economic, managerial and even political as well.

Keywords : Million Neural Net Module Artificial Brains, Evolvable Hardware (EHW, E-Hard), Evolutionary Engineering, Artificial Brain Architectures, CAM-Brain Machine (CBM), Kitten Robot (Robokoneko), Cellular Automata, Neural Networks, Genetic Algorithm, "J-Brain Project".

1 Introduction

1.1 Background

This is a position paper, which states the visions of ATR's Brain Builder Group (BBG) for the next few years. The present aim of our project is to build massively parallel evolvable hardware which will achieve the scale necessary for real-time intelligent perception and control - such hardware we call "artificial brains".

These artificial brains will be too large and too complex to program explicitly - hence we aim to evolve them. The evolutionary approach is inspired by our past experiments with an artificial creature (Lizzy) which we hand-assembled from neural-net modules. However, Lizzy's modules were evolved with the aid of hand-crafted fitness functions (see below). We argue that the crafting of fitness functions and the assembly of evolved modules will be done by a new breed of software professionals called Evolutionary Engineers (EEs).

As a logical model for scalable parallel hardware, we have adopted a 3-dimensional Cellular Automaton paradigm, which is at present moving from simulation to physical prototypes. In 1998, the BBG will possess new hardware which will make the building of 10,000 neural net module artificial brains practical. Hence the time is ripe to start thinking about what might be interesting to build with 100,000; 1,000,000 etc modules.

1.2 ATR's "CAM-Brain Project"

This initiative [de Garis 1993, 1994, 1995, 1996; Gers & de Garis 1996, Korkin & de Garis, 1997], is a project funded by Japan's ATR, running from 1993 to 2001, that aims to build a billion neuron artificial brain by 2001, using cellular automata based neural net modules. Owing to the simplicity of the cellular model, the state of each cell can be stored in a few bytes of RAM (2 bytes in the CoDi 1-bit model [Gers, de Garis and Corkin 97]. Modern workstations easily accept 1 GigaByte of RAM, easily allowing the simulation of networks with ten million artificial neurons, albeit at software speeds. The next step in the CAM-Brain Project is to find a way to evolve neural network modules quickly.

To solve this problem, the Brain Builder Group at ATR is building an electronic device called a "CAM-Brain Machine (CBM)" which grows/evolves the neural circuits (neurons, axons, dendrites and synapses) at electronic speeds inside Xilinx's new Field Programmable Gate Array (FPGA) XC6264 chips [Xilinx 1996]. The CBM implementation was subcontracted in December 96, and a proof-of-concept prototype has already been delivered, with 2 gate-arrays and PCI bus host connectivity; The full version should be delivered in the first quarter of 98. It will be able to evolve several hundred generations of a Genetic Algorithm (GA) population of a 100 neural circuits in a fraction of a second, i.e. a theoretical upper limit of hundreds of thousands of evolved modules a day. Each neural net module consists of some 6000 cellular automata cells in a 3D space, containing roughly 100 artificial neurons. For details see [Korkin & de Garis 1997].

Of course such astronomical module evolution speeds will not be obtainable in practice, because the real bottleneck will shift from the former slowness of software evolution speeds to the human thinking time required to conceive the inter-module circuitry, and the fitness definitions of the various modules contained in the circuit. (At this stage of the research, the individual modules are evolved, whereas the connections between modules and the overall multi-modular circuitry is user (i.e. humanly) specified. Simultaneous multi-modular evolution is a topic for future research. Interestingly, the CBM will allow many thousands of modules to evolve simultaneously in the work station RAM, so this option can be explored in 1998.

The CBM includes programmable hardware into which human programmers can compile the fitness definition for each module. Thus the fitness of each evolving circuit will be measurable at electronic speeds. The CBM can also function as a successor to the machine currently used by ATR's Brain Builder Group to update the cellular automata (CA) cells which form the basis of the evolvable neural medium [de Garis 1994]. At present, ATR uses MIT's "CAM-8" machine, which can update 200 million CA cells a second. However the chips and methodology of this machine date from 1988. The CBM will be able to update about 100 BILLION cells a second, i.e. about 500 times faster, as Moore's law would predict. Once the evolution of all the neural modules is complete (using the CBM), their evolved CA circuits will be downloaded into a gigabyte of workstation memory to be updated at run time by the CBM. Current estimates are that the CBM will be able to update each CA cell in the RAM at a rate of about 2500 times per second, which we believe sufficient for real time robot control experiments.

In parallel with the testing of the capabilities of the CBM, we intend to design a kitten robot called "Robokoneko" (which is Japanese for "robot kitten") which will be life size, with four legs, a tail, two small TV cameras for eyes, two microphones for ears, an on-board battery and a two-way radio antenna to communicate sensor data to, and movement instructions from, the offline CBM and workstation. "Robokoneko" should weigh less than 2 kilos, so that it can romp around the lab at (hopefully) real life kitten speeds, displaying a repertoire of kitten like behaviors, which should be possible because its brain will consist of 10,000 evolved neural net *modules*.

The CBM will make the creation of million-module artificial brains realistic within the next few years, provided of course, that there are enough human "evolutionary engineers" (EEs) to think up all the fitness definitions and inter module circuit designs in a reasonable time, a requirement which will necessitate the creation of human "fitness (definition) teams" - e.g. the "vision" team, the "legs" team, the "behavioral control" team, etc. Therefore, the time is ripe to start thinking about what such million-module systems might do. This paper provides some suggestions concerning what kinds of artificial brains (with increasing complexities and capabilities) might be interesting to evolve over the next few years. Of course, once the CBM hardware is ready, ideas on what to evolve will develop quickly, especially when other research centers buy the hardware from ATR. The existence of the hardware will stimulate development of the appropriate theories and methodologies. It often happens in the history of science that a new technology or new tool transforms a specialty, e.g. consider the impact that the

invention of the telescope had on astronomy, or the impact of the electron microscope on cell biology etc.

The remainder of this paper consists of the following. Section 2 presents in some detail a 100 module "artificial brain" architecture, which gives an idea of the kind of thing envisioned by the authors. Section 3 presents some suggestions on what might be interesting to evolve for larger and very much larger systems. Finally, in section 4, some comments on the longer term future of brain building are given.

2 A 100 Module System

The aim of this section is to give a fairly concrete example of a multi-module system. The ideas used in this example may prove useful when the time comes to build 1000; 10,000; etc module systems.

de Garis's PhD thesis and other papers [de Garis 1991a,b, 1992] details the simulation of an animat lizard called LIZZY. This 3D quadruped is shown in Fig. 1. It possessed 3 basic behaviors, namely - eat, don't be eaten, and mate. These behaviors were generated and controlled by evolved neural net modules assembled into humanly specified control circuits. (Perhaps at a later stage of research it might be possible to evolve the inter-module connections). Actually, with the workstations of that period, only the detectors and motion controllers were evolved (for reasons of simulation speed - every time a new neural net module was added to the software, the speed of LIZZY's motion on the computer screen decreased). (For a movie of Lizzy's motions, see de Garis's web site, and for full details of Lizzy, see de Garis's PhD thesis, Ch. 7, also on his web site).

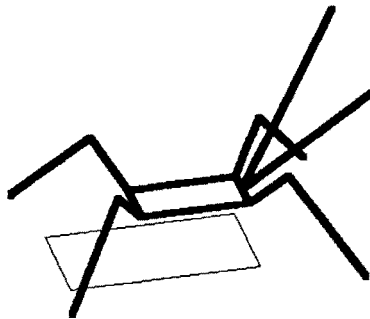


Fig. 1 LIZZY : An Artificial Creature

This section presents a more complete architecture of LIZZY, including the evolved control modules, to illustrate how it is possible to build artificial nervous systems (artificial brains) using evolved neural net modules. This approach will form the basis for future attempts to build million module artificial brains. These modules can be classified into three broad categories, namely - detector, decision, and motion. In broad terms, the detectors feed their output to the decision

modules (which can be combined to form "production rules"), and the outputs from the production rule circuits determine which behaviors to switch on.

LIZZY's antenna are capable of detecting the frequencies of sinusoid signals emanating from an arbitrary fixed point (using evolved frequency detectors). LIZZY is capable of detecting three kinds of creature in its environment - a prey (which, in the simulation model, emits a high frequency), a mate (which emits a middle frequency), and a predator (which emits a low frequency). Each antenna at its tip contains an average signal strength detector. At the base of the two antenna is a signal strength difference detector, which allows Lizzy to orientate towards a prey or a mate, or away from a predator. In the model, the amplitude of the sinusoid signals emanating from the fixed point drops off linearly with distance.

Lizzy's behavioral repertoire is as follows. If a prey or a mate is nearby (as indicated by the frequency detector), Lizzy orientates toward the signal source, walks until it is close, then stops and eats or mates (i.e. the front or back legs move up and down). If the frequency detector shows a predator, Lizzy orientates away from the source and walks away until the signal strength is weak.

There are 5 evolved motions, i.e. one module per motion.

- a) Walk straight ahead
- b) Turn left
- c) Turn right
- d) Peck at food (raise/lower front legs)
- e) Mate with partner (raise/lower back legs)

These motions are switched on and off by enabling signals coming from production rules of the form -

IF (A&B&C) => motion 2, where A,B,C are conditions.

These production rules and motions are created by evolving neural net modules. In greater detail, a Lizzy production rule might look like the following -

IF[(prey)&(SS(L)>SS(R))&(SS<K1)&(SSD>K2)] => Turn-L

i.e. if the source emits a high frequency (i.e. prey = a small animal = food) & the signal strength at the left antenna (SS(L) = signal strength detector (left)) is greater (i.e. closer to the source) than the SS(R) & the signal strength < constant K1 (i.e. weak signal = animal is far away) & signal strength detector difference SSD = |SS(L) - SS(R)| > constant K2 THEN turn left.

In other words, if its a prey, and the left antenna is closer to the prey and the prey (signal) is too far way to eat (so that Lizzy has to move) and Lizzy is not oriented toward the prey, then turn left. This is just common sense.

Other rules are -

IF[(prey)&(SS(R)>SS(L))&(SS<K1)&(SSD>K2)] => Turn-R

which is the right equivalent of the above rule.

IF[(prey)&(SS<K1)&(SSD<K2)] => Walk

(i.e. if prey and far away and aligned with prey => walk)

IF[(prey)&(SS>K1)] => Eat (peck at prey)

(i.e. if prey and very close, i.e. the signal strength is high => eat)

Four similar rules exist for mating -

IF[(mate)&(L>R)&(SS<K1)&(SSD>K2)] => Turn-L

IF[(mate)&(R>L)&(SS<K1)&(SSD>K2)] => Turn-R

IF[(mate)&(SS<K1)&(SSD<K2)] => Walk

IF[(mate)&(SS<K1)] => Mate (with partner)

and similarly for a predator -

IF[(predator)&(L>R)&(SS>K3)] => Turn-R

IF[(predator)&(R>L)&(SS>K3)] => Turn-L

IF[(predator)&(SS>K3)&(SSD<K2)] => Walk (Run)

When a motion is switched on, it continues for time "T" (CA clock cycles), to avoid rapid (useless) switching between behaviors (a notion called "behavioral commitment"), so a timer module is needed.

It is usually easier to evolve a single-function module than a multi-function module. However, sometimes a multi-function module is needed, as will be shown below. The neural net modules evolved in [de Garis 1990, 1991a,b, 1992] were all fully connected (binary fraction bit strings representing the connection weights were concatenated to form the GA chromosomes).

For more details on neural net module ("GenNet") evolution, see [de Garis 1990] or consult de Garis's PhD thesis downloadable from his web site). However, when using cellular automata based neural nets (as is the case for the CAM-Brain Project), the connectivity is less, but still dense enough for good evolvability [de Garis 1996]. Extensive connectivity leads to more complex dynamics, hence greater functionality.

Module Descriptions and Fitness Definitions

Having given an idea of the type of control rules which govern Lizzy's behaviors, we now show how the modules used in such circuits can be evolved.

Frequency Detector

Use as input, 3 sinusoids of different frequencies (low, middle, high), e.g. wavelength = 20, 30, 40 clock cycles, and many amplitudes. Evolve 3 separate modules, one for each frequency detection. When the non desired frequency is input, a low output is desired (where output signal values range from 0.0 to 1.0). This output should be high if the input is the desired frequency.

(For details on the actual evolution of Lizzy's frequency detection modules, see de Garis's PhD thesis on his web site). The 3 frequencies are input sequentially (with a variety of amplitudes), and the numerical output values stored over a reasonable number of clock cycles. (The CA are synchronous). A fitness definition meeting these specifications could be the following. (Outputs range from 0.0 to 1.0).

$$\text{FITNESS} = \text{reciprocal}((\text{sum over } i \text{ clock cycles for non desired frequencies } (\text{OUT}_i - 0.0) * (\text{OUT}_i - 0.0)) + \text{weighting factor } (\gamma > 1) * (\text{sum over } i \text{ for desired frequency } (\text{OUT}_i - 0.8) * (\text{OUT}_i - 0.8)))$$

Note that it is useful to extract the outputs after a certain settling delay of the signals.

Signal Strength Detector (SS)

The output should be proportional to the average signal strength coming in (i.e. if the input signal is of the form $A \sin(\Omega * T_i)$, then the SS output should be roughly $A^2/2$). So input a series of sinusoids with many amplitudes and many frequencies, with the following fitness definition. (Again, see de Garis's PhD thesis for details).

$$\text{FITNESS} = \text{reciprocal}(\text{sum over frequencies } (\text{sum over amplitudes } "j" (\text{sum over clock cycles } "i" (\text{OUT}_i - A_j * A_j / 2) * (\text{OUT}_j - A_j * A_j / 2))))$$

Signal Strength Difference Detector (SSD)

An SSD is used to orientate Lizzy relative to the signal source, due to the signal strength differences between signals received at the left and right antenna. An SSD takes two SS outputs as inputs and outputs their difference. To evolve it, input combinations of real numbers, i.e. the tensor product of [0.1, 0.2, 0.3, ... 0.8, 0.9] with itself.

$$\text{FITNESS} = \text{reciprocal} (\text{sum over clock cycles } "i" (\text{sum over } j = 0.1 \text{ to } 0.9 (\text{sum over } k = 0.1 \text{ to } 0.9 ((\text{OUT}_{ijk} - \text{modulus}(j - k)) * (\text{OUT}_{ijk} - \text{modulus}(j - k))))))$$

Other Modules

Other modules evolved include comparators, and gates, or gates and timers, maximum position detectors, saturators (amplify a signal to maximum).

Putting It All Together

Now that a number of individual modules have been defined, both in terms of function and fitness, they can be put together into humanly specified functional intermodular circuits. Fig. 2 shows how the following production rule can be implemented using various modules.

IF [(prey)&(SS(L)>SS(R))&(SS<K1)&(SSD>K2)] =>Turn-L

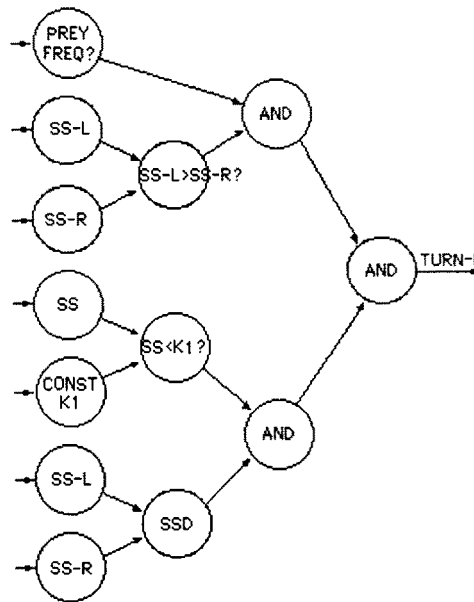


Fig. 2 A Production Rule Circuit

Fig. 3 shows how Lizzy's 5 behaviors can be switched on and off. Only one motion control can be sent to the legs at a time, e.g. if the WALK signal is strongest, then the module MAX-POSN module for WALK will go high and trigger WALK's Timer module (T), making WALK last for T cycles. When WALK's Timer is high, it sends saturated signals to the other MAX-POSN modules, switching them off, until WALK's Timer goes low, T clock cycles later. Then another (or the same) motion is high and a similar process occurs. In the action bus, output signals sum at intersections, which allows for a smooth transition between leg motion types.

The above discussion of LIZZY is only an example of the type of thinking involved in making multi-module neural architectures. We believe that new professions will be created, namely the "Evolutionary Engineer (EE)", whose job will be to invent neural module functions and fitness definitions, and "Brain

Architect (BA)", whose job will be to design artificial brains. In a large scale brain building project, it is likely that top level designers will be the BAs, and they will pass down their high level specifications to lower level EEs who will evolve the actual modules with CBMs. With the creation of the world's first CBM by the end of the century, the need for BAs and EEs will soon arise.

3 Larger Systems

This section is more speculative. It presents some early ideas on what kinds of "large N" systems might be interesting to evolve/build. We begin with $N = 1000$, and increase each time by an order of magnitude, up to 10,000,000. It also discusses some of the personnel, management and political issues involved, as the scales of the projects increase.

The $N = 1000$ Case

The above LIZZY architecture gives an idea of how artificial nervous systems (or, if there are enough modules - artificial brains) can be assembled from evolved neural net modules. By simply adding to LIZZY's behavioral repertoire, one can quickly increase the number of modules to 1000. LIZZY could be made to behave like a toy kitten, so that it could jump, chase its tail, emit simple cries, run at different speeds, etc. A hard working evolutionary engineer (with a CBM) could probably build a 1000 module creature alone (in simulation), but if the modules control a physical robot, maybe a team of two people (i.e. with one of them being a roboticist) could do this work in a year.

The $N = 10,000$ Case

With ten thousand modules, one can begin to experiment with vision and hearing. Simple artificial retinas could be built with some post retinal processing. Maybe some memory could be added. This seeing and hearing creature could avoid objects, approach or flee from slow or fast moving objects respectively, pick up things, etc. At this number of modules, two people would be stretched to do the whole project in a few years. Probably a small team of several evolutionary engineers, a roboticist, and a general programmer would be needed (plus a CBM of course) - say, four people at the least. In fact, the main task of the authors for 1997 will be to create a 10,000 module architecture to control the kitten robot "Robokoneko", as mentioned earlier in this paper. As of February 1997, this work has not yet started, so the many challenges of generating a 10,000 module system design are not yet familiar to us, although this will quickly change once the CBM is ready (the kitten robot as well) by the end of 1997. Our papers in 1998 (the year in which the CBM, Robokoneko and a 10,000 module architectural plan are integrated) will probably have a very different flavor compared to this one.

The N = 100,000 Case

With one hundred thousand modules, more serious versions of creatures with memory, vision, motion generation and detection, hearing, simple comprehension, and multi-sensor interaction can be built. At this number of modules, one needs to begin thinking seriously about the management and personnel planning aspects of such a project. Probably about a dozen or more people would be needed to finish such a project within a few years, a figure within the reach of many universities and smaller companies. Probably most examples of artificial brains will be of this size, given the realities of university and medium sized companies research budgets.

The N = 1,000,000 Case

For a million module system, the management and personnel demands become large. For example, if one makes the assumption that on average, it takes a (human) evolutionary engineer (EE) one hour to conceive and compile the fitness definition of a module (and link the module to a global inter module architecture), then how many EEs would be needed for a 2.5 year, million module, artificial brain research project? Assuming an 8 hour day, a 40 hour week, a 50 week year, i.e. a 2000 hour year, the project would need 500 EE-years, spread over 2.5 years, hence 200 EEs would be needed. This number could be afforded by a large company, so one can expect companies to start building large artificial brains before the year 2000.

Of course, the above figure is based only on the fitness definition creation times. A brain builder project would need to consider many other factors (e.g. sensor development, robotics problems, coordination of human groups, neural signal pathway specification and routing, etc.), which would add to the project time, personnel numbers and costs. Nevertheless, with a suite of CBMs (i.e. each with fraction-of-a-second module evolution times), the complete construction of a million module artificial brain becomes quite realistic within 5 years, say by the end of 2001. It will be interesting to read this paper 5 years from the time of writing (February 1997) to see how far off the mark we were, if at all.

Suggested examples of million module systems might be, artificial kitten pets for children and the aged, robot "guide dogs" to help blind people cross the road, household cleaner robots, etc. These systems would include quite elaborate artificial retinas, and post retinal processing, memory processing, sound generation, even early speech.

The N = 10,000,000 Case

One immediately jumps to the 2000 personnel range. This is a major national or international project, and too big for all except the biggest companies. de Garis dreams that Japan will start a "J-Brain (J = Japanese) Project" (2001-2005), to build a ten million module (billion neuron) artificial brain, once the technologies and methodologies have matured, based on experience gained from

the building of smaller brains. Large national projects of this kind would compare with America's NASA project to put a man on the moon.

Japan's "J-Brain Project" would attempt to build the world's most intelligent artificial brain. In the process, the project would create a whole new industry, where the brain-like computer market would eventually be worth a trillion dollars a year, and would bring tremendous international prestige to Japan, which so far has had an international "uncreative copycat" image. Every household would want a home cleaner robot that could do the vacuuming, the shopping, emptying the garbage, washing the car, etc.

In fact, at the time of writing (February 1997), one of Japan's major scientific research funders, the STA (Science and Technology Agency) has budgeted 20 TRILLION (20,000,000,000,000) yen over a 20 year period, starting in the fall of 1997, to finance three areas of brain research, namely basic neuro-science, neuro-medical-science, and brain engineering. At roughly 20 million yen per researcher per year (salary, equipment, tax etc), that's 50,000 researchers a year. de Garis hopes to persuade the STA to finance the "J-Brain Project" (a mere 2000 researchers)!

Japan is not the only country interested in brain building. Officials of America's scientific research funders NSF and DARPA have both asked de Garis to talk to them about CAM-Brain, which he did in April 1997. The Chinese are also interested - three of the authors and brain stormers of this paper are Chinese. ATR and Wuhan University collaborate closely. Wuhan University hopes to become the major center for brain building research for the whole of China.

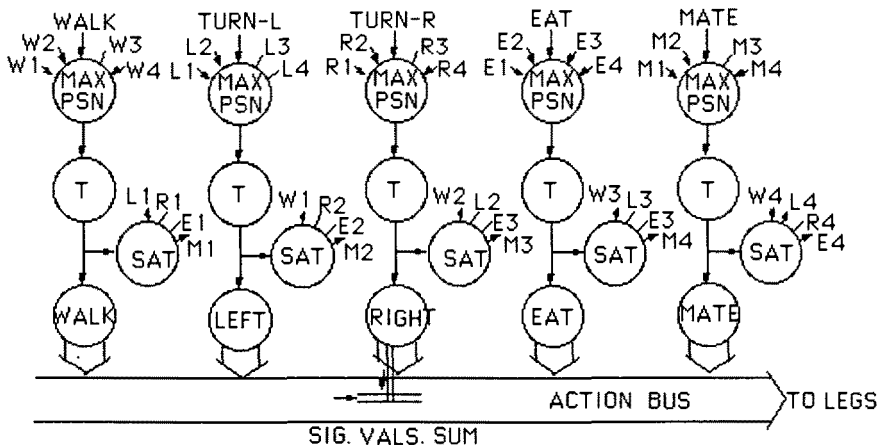


Fig. 3 Motion Selection Circuit

4 Comments

With million module systems and larger, people can begin to test serious models of biological brain function. As electronic technology improves (in our exciting electronic era of what de Garis calls "massive Moore doublings", i.e. where the size of the progressive increments in electronic speeds and densities is becoming enormous, with a 4 GIGAbits experimental memory device announced in early 1997), it will be possible to evolve more biologically realistic neural circuits, so that brain building and brain science can come closer together and benefit from each others advances.

However, ten million module artificial brains are only the beginning. Molecular scale electronics (e.g. single electron transistors (SETs), molecular electronic devices (MEDs), quantum dots (QDs), etc) will mean that the heat generation problem (which arises when conventional, irreversible, register-clearing, computing techniques destroy information) at molecular level will have to be overcome. If not, molecular scale circuits will reach temperatures of exploding dynamite. The only way to go will be to reduce the heat (virtually to zero) by using "reversible computation" techniques [Feynman 1996].

Heatless computers will allow electronics to use 3D, size-independent circuits, with 1 bit per atom, so in theory, one could have self assembling asteroid size computers with 10 to power 40 components. These huge numbers absolutely dwarf the human brain's pitiful tens of billions of neurons.

Brain builder technology in the late 21st century will threaten humanity's status as dominant species, with profound global political consequences. (See the "Cosmist" essays on this topic on de Garis's web site, or [de Garis 1996b]).

References

Note : de Garis papers can be found at site :- <http://www.hip.atr.co.jp/~degaris>

[de Garis 1990] "Genetic Programming : Building Artificial Nervous Systems Using Genetically Programmed Neural Network Modules", Hugo de Garis, in Porter B.W. & Mooney R.J. ed., Proc. 7th. Int. Conf. on Machine Learning, pp 132-139, Morgan Kaufmann, 1990.

[de Garis 1990b] "Genetic Programming : Modular Evolution for Darwin Machines", Hugo de Garis, Int. joint Conf. on Neural Networks, January 1990, Washington DC, USA.

[de Garis 1991a] "Lizzy: The Genetic Programming of an Artificial Nervous System", Hugo de Garis, Int. Conf. on Artificial Neural Networks, June, 1991, Espoo, Finland.

[de Garis 1991b] "Genetic Programming, Artificial Nervous Systems, Artificial Embryos, and Embryological Electronics", Hugo de Garis, in "Parallel Problem Solving from Nature", Lecture Notes in Computer Science, 496, Springer Verlag, 1991.

[de Garis 1992] "Artificial Nervous Systems : The Genetic Programming of Production Rule GenNet Circuits", Hugo de Garis, Int. Joint Conf. on Neural Networks, November 1992, Beijing, China.

[de Garis 1993] "Neurite Networks : The Genetic Programming of Cellular Automata based Neural Nets which Grow", Hugo de Garis, Int. Joint Conf. on Neural Networks, October 1993, Nagoya, Japan.

[de Garis 1994] "An Artificial Brain : ATR's CAM-Brain Project Aims to Build/Evolve an Artificial Brain with a Million Neural Net Modules inside a Trillion Cell Cellular Automata Machine", Hugo de Garis, New Generation Computing Journal, Vol. 12, No.2, Ohmsha & Springer Verlag.

[de Garis 1995] "The CAM-Brain Project : The Genetic Programming of a Billion Neuron Artificial Brain by 2001 which Grows/Evolves at Electronic Speeds inside a Cellular Automata Machine", Hugo de Garis, Int. Conf. on Artificial Neural Networks and Genetic Algorithms, April 1995, Ales France.

[de Garis 1996] "CAM-Brain : ATR's Billion Neuron Artificial Brain Project : A Three Year Progress Report", Hugo de Garis, Int. Conf. on Evolutionary Computation, May 1996, Nagoya Japan.

[de Garis 1996b] "Cosmism : Nano-Electronics and 21st Century War", Nanotechnology Magazine, July, 1996, also on de Garis's web site under "Essays".

[Feynman 1996] "The Feynman Lectures on Computation", R.P. Feynman, Addison Wesley, 1996.

[Gers & de Garis 1996] "CAM-Brain : A New Model for ATR's Cellular Automata Based Artificial Brain Project", Felix Gers & Hugo de Garis, Int. Conf. on Evolvable Systems, October 1996, Tsukuba, Japan.

[Korkin & de Garis 1997] "CBM (CAM-Brain Machine) : A Hardware Tool which Evolves a Neural Net Module in a Fraction of a Second and Runs a Million Neuron Artificial Brain in Real Time", Micahel Korkin & Hugo de Garis, Genetic Programming Conference, July, 1997, Stanford, USA.

[Xilinx 1996] "Xilinx Data Manual 1996".

Representations, Fitness Functions and Genetic Operators for the Satisfiability Problem

Jens Gottlieb and Nico Voss

Technische Universität Clausthal, Institut für Informatik,
Erzstraße 1, D-38678 Clausthal-Zellerfeld, Germany.
{gottlieb,nvoss}@informatik.tu-clausthal.de

Abstract. Two genetic algorithms for the satisfiability problem (SAT) are presented which mainly differ in the solution representation. We investigate these representations – the classical bit string representation and the path representation – with respect to their performance. We develop fitness functions which transform the traditional fitness landscape of SAT into more distinguishable ones. Furthermore, new genetic operators (mutation and crossover) are introduced. These genetic operators incorporate problem specific knowledge and thus, lead to increased performance in comparison to standard operators.

1 Introduction

The satisfiability problem (SAT) is the first problem proved to be NP-complete [GJ79] and can be stated as follows. Given a boolean function $f : \mathbb{B}^n \rightarrow \mathbb{B} = \{0, 1\}$, the question is: Does there exist a variable assignment $x = (x_1, \dots, x_n) \in \mathbb{B}^n$ with $f(x) = 1$?

In this paper we assume without loss of generality that f has conjunctive normal form $f = c_1 \wedge \dots \wedge c_m$ with each clause c_i being a disjunction of k_i literals, i.e. (positive) variables and negated variables. Furthermore, we suppose $k_i = k$ for all $i \in \{1, \dots, m\}$ and for a constant k .¹ Our goal is to find a variable assignment $x \in \mathbb{B}^n$ that satisfies all clauses.

It is challenging to develop heuristic methods for this problem because unless $P = NP$ there does not exist an exact algorithm with polynomial time complexity for SAT. As there is a growing interest in evolutionary computation techniques in the last years, it seems to be natural that many researchers apply genetic algorithms (GAs) to SAT and 3-SAT [DJS89, Fra94, Hao95, Par95, FF96, EH97].

We investigate two different solution representations of SAT, one of them being the classical bit string representation. The other representation is the path representation which emphasizes the satisfaction of clauses. For each representation we introduce new fitness functions which contain more heuristic information than the traditional approaches. Moreover, we propose problem specific mutation and crossover operators.

¹ Note that even for $k = 3$ (3-SAT) the problem is NP-hard, while all instances with $k = 2$ (2-SAT) are solvable in polynomial time [GJ79].

This paper is organized as follows. After a review of related work in Sect. 2, Sect. 3 introduces the ingredients (fitness function, mutation and crossover) of the genetic algorithm based on the bit string representation. Also, some computational results are presented. Section 4 deals with the path representation, its fitness function and genetic operators, and presents the obtained results. Finally, the conclusions and possible directions for future research are given in Sect. 5.

2 Related Work

De Jong and Spears [DJS89] are the first who have applied GAs to solve SAT. They use the bit string representation with two-point crossover and standard mutation, and do not assume f to be in conjunctive normal form. Thus, they introduce a fitness function with range $[0, 1]$ that recursively evaluates a given expression.

Frank [Fra94] reports that the use of hillclimbing before the other genetic operators significantly enhances the solution quality for 3-SAT problems. Furthermore, a specific parallel GA with interacting sub-populations seems to be inferior to a “normal” GA with only one population.

Hao [Hao95] proposes a (non-standard) representation that emphasizes the local effects of the variables in the clauses and that is strongly related to our path representation. Each clause is assigned a partial variable assignment that satisfies this clause; all these assignments form the chromosome. Some of these “local” variable assignments may be inconsistent and thus, the goal is to find a chromosome without any inconsistencies. Hao employs fitness functions that guide the GA search into regions with only few inconsistencies. He presents a special bit mutation that ensures local consistency, and local search operators.

Park [Par95] checks the effect of crossover and mutation in GAs with bit string representation for 3-SAT. He assumes a conjunctive normal form and uses the number of satisfied clauses as fitness function. He reports similar performance of uniform and two-point crossover, but comes to the conclusion that a GA with standard mutation alone is more effective than a GA including crossover.

Another bit string based approach is proposed by Fleurent and Ferland [FF96]. They use standard mutation and a heuristic crossover operator based on uniform crossover that exploits information about clauses that are not satisfied by both parents. Our crossover operators pursue a similar idea to make use of problem specific knowledge. Fleurent and Ferland report good results for GAs incorporating local optimization.

Eiben and van der Hauw [EH97] apply adaptive GAs based on the bit string representation to 3-SAT. They use standard mutation and investigate multi-parent operators but conclude that a GA using a population of size one (and thus, no crossover) yields sufficient good results. They employ an adaptive penalty function to guide the search towards solutions satisfying yet unsatisfied clauses. Their approach is noteworthy for its generality as it is principally applicable to any constrained problem, e.g. the graph coloring problem [EH96].

Besides the GA approaches many local search algorithms can be found in literature. Selman et al. [SLM92] report good results for their GSAT procedure. Many enhancements of this algorithm are proposed with respect to escaping from local optima [SKC94] or heuristic weights of clauses [Fra96, Fra97]. Gu [Gu94] gives an overview of other optimization algorithms for SAT. The most prominent exact algorithm originates from a method proposed by Davis and Putnam [DP60].

3 The Bit String Representation

3.1 Representation and Basic Fitness Function

The most obvious way to represent a solution of SAT is a bit string of length n where every variable x_j corresponds with one bit. As genetic algorithms use the fitness function to guide the search into promising regions of the search space it is very important to design a well suited fitness function. Obviously, the simplest approach is to take the given boolean function f as fitness function. The main drawback of this approach is that unless the GA has found a solution all individuals would have the same fitness value 0. Hence, the GA gets no information from f and consequently, the GA search degenerates to pure random search which is very ineffective.

In what direction of the search space should we guide the GA search? We must define a fitness function that is able to distinguish individuals x with $f(x) = 0$. Such a function should have higher values if the distance to an optimum is getting lower. Hence, it is reasonable to use $fit_B(x) = \text{“number of satisfied clauses”}$ as *basic fitness function*, see [Par95, EH97]. Note that an individual x with maximum fitness value solves our SAT instance, and that the range of fit_B is $\{0, \dots, m\}$, where m is the number of clauses of f . We interpret this problem formulation as a maximization problem, i.e. the goal is to find a solution with maximum objective (fitness) value. Hence, we use the word solution for an individual although it need not be a solution of the original SAT instance.

3.2 Refining Functions

The fitness landscape induced by the basic fitness function consists of at most $m + 1$ different heights. Due to the large search space there are many solutions having the same fitness value. Thus, any GA using this fitness function is not able to distinguish between such solutions. This would be acceptable if all these solutions have the same chance to lead to improved solutions or even the global optimum. But this need not be the case for most instances of SAT.

We define *refining functions* with range $[0, 1)$ which capture heuristic information about the solution quality. These functions are added to the basic fitness function yielding a fitness function with range $[0, m + 1)$. The reason for the restricted range $[0, 1)$ of the refining functions is that the main distinction between two individuals should remain the number of satisfied clauses. Hence, the

heuristic information contained in the refining function becomes most influential if some individuals have the same value with respect to fit_B .

Let p_j and n_j be the numbers of positive and negative occurrences of variable x_j in f , respectively. If $p_j > n_j$ holds it is a reasonable rule of thumb to set $x_j = 1$: This could cause a higher number of satisfied clauses and more alternatives for other variables in the clauses satisfied by x_j . Together with the corresponding argument for the case $p_j < n_j$, this idea leads to the first refining function

$$ref_{B1}(x) = \frac{1}{n} \sum_{j=1}^n \frac{x_j p_j + (1 - x_j) n_j}{1 + p_j + n_j}$$

which puts emphasis on satisfied clauses. The second refining function

$$ref_{B2}(x) = \frac{1}{n+1} \sum_{j=1}^n \frac{1}{(1 - x_j) p_j + x_j n_j + 1}$$

is based on the same idea, but emphasizes the number of unsatisfied clauses. Thus, ref_{B2} has increasing values for a decreasing number of unsatisfied clauses for a given variable. The third refining function

$$ref_{B3}(x) = \frac{1}{n} \sum_{j=1}^n \frac{x_j p_j + (1 - x_j) n_j}{(1 - x_j) p_j + x_j n_j + p_j + n_j + 1}$$

is simply a combination of ref_{B1} and ref_{B2} .

Whereas the first three refining functions are based on information about the variables in the conjunctive normal form, the fourth refining function uses information on the clauses in f . Therefore, let $s_i(x) \in \{0, \dots, k\}$ be the number of variables x_j that satisfy c_i . The higher $s_i(x)$ for an assignment x , the more variables contained in c_i may be changed without violating the constraint that x must satisfy c_i . As these changes could result in a higher number of satisfied constraints, we define the last refining function as

$$ref_{B4}(x) = \frac{1}{k(m+1)} \sum_{i=1}^m s_i(x) .$$

Adding one of these refining functions to the basic fitness function changes some plateaus in the fitness landscape into a more distinguishable landscape containing additional small hills. This should give more information to the GA to direct the search into promising regions of such plateaus and to leave such plateaus towards regions of higher quality. It is important to use the refining function together with the basic fitness function, as was confirmed by experiments. Otherwise the main information of any solution (the number of satisfied clauses) is lost and the search would probably be guided into regions of suboptimal solutions.

3.3 Mutation and Crossover

The standard mutation for bit strings (which changes each bit with the same probability) may destroy good parts of the chromosome and is not able to guide the search into regions of higher quality. Hence, we propose a mutation operator that tries to improve a solution while preserving its promising parts.

Our operator M_B changes only bits corresponding to variables which are contained in at least one unsatisfied clause. M_B checks all unsatisfied clauses in random order. The contained variables are changed with probability $p_M \cdot \text{fit}_B(a)/m$ which helps avoiding premature convergence as the mutation probability is increased if the distance of the solution a to the optimum gets lower. Note that every bit is changed at most once by one application of M_B . This operator is motivated by the observation that in the current solution at least one of the variables in each unsatisfied clause must be changed to obtain an optimal solution. The WSAT procedure of Selman et al. is based on the same idea [SKC94].

```

procedure MUTATION  $M_B$  (parent  $a \in \mathbb{B}^n$ )
  set  $\bar{a} = a$ 
  for  $c_i \in \{c_1, \dots, c_m\}$  do    {random order}
    if clause  $c_i$  is not satisfied by  $a$  then
      for variables  $x_j$  contained in clause  $c_i$  do
        if  $\bar{a}_j$  is unchanged and  $\text{random} < p_M \cdot \text{fit}_B(a)/m$  then
          set  $\bar{a}_j = 1 - \bar{a}_j$  and mark  $\bar{a}_j$  as changed
  return child  $\bar{a}$ 

```

Fig. 1. Pseudocode for mutation operator M_B

Due to preliminary experiments with traditional crossover operators for bit strings (uniform, one-point, two-point and some variants of n -point crossover) we find no crossover being able to produce acceptable results. Hence, we have designed other crossover operators and now present C_B , the most successful one which incorporates knowledge of the structure of SAT.

The procedure C_B duplicates the parents (yielding children $\bar{a}$ and $\bar{b}$) and checks the clauses in random order. For each clause c_i all contained literals are sequentially tested whether the corresponding variable assignments of $\bar{b}$ make these literals satisfy c_i . If they do so, these variable assignments $\bar{b}_j$ are copied to $\bar{a}_j$ with probability $r \in (0, 1]$. Otherwise the corresponding values of $\bar{a}$ are copied to $\bar{b}$ with probability r .² This crossover operator transfers good parts between

² This is motivated as follows: If $\bar{a}_j \neq \bar{b}_j$ and $\bar{b}_j$ does not satisfy c_i , then $\bar{a}_j$ must satisfy c_i . Furthermore, note that (i) the assignments $\bar{a}_j$ and $\bar{b}_j$ remain unchanged if $\bar{a}_j = \bar{b}_j$, (ii) each bit can be changed at most once during a single crossover operation (one flip yields $\bar{a}_j = \bar{b}_j$ and makes more flips impossible), and (iii) r is a specific parameter of C_B that must not be confused with the crossover probability p_C in the GA.

the parents, which causes the children to have a smaller Hamming distance than the parents with high probability. This may yield a loss of diversity in the current population and thus, illustrates the high importance of a mutation operator that helps escape from local optima, a condition that is satisfied by M_B .

```

procedure CROSSOVER  $C_B$  (parents  $a, b \in \mathbb{B}^n$ )
  set  $\bar{a} = a$  and  $\bar{b} = b$ 
  for  $c_i \in \{c_1, \dots, c_m\}$  do    {random order}
    for variables  $x_j$  contained in clause  $c_i$  do
      if  $random < r$  then
        if  $\bar{b}_j$  satisfies  $c_i$ 
          then set  $\bar{a}_j = \bar{b}_j$ 
          else set  $\bar{b}_j = \bar{a}_j$ 
  return children  $\bar{a}$  and  $\bar{b}$ 

```

Fig. 2. Pseudocode for crossover operator C_B

3.4 Computational Results

Mitchell et al. [MSL92] report the hardest satisfiable instances of 3-SAT having the ratio $m/n = 4.3$. Thus, we restrict our investigations to these instances. Each instance is produced by randomly generating m clauses. The literals of each clause are generated by randomly choosing variables and negating them with probability 0.5. Furthermore, we ensure satisfiability of the instance by restricting the alternatives for the generation of one literal in each clause.³

We use a generational GA with elitism and population size 12 (this yields good results with respect to the total number of evaluations). The initial population P_0 is generated randomly. Population P_{i+1} is computed by tournament selection (tournament size 2) from P_i , mating some pairs via crossover with probability $p_C = 0.7$ and mutating all individuals using the parameter $p_M = 0.3$. The additional crossover parameter is $r = 0.65$. To ensure elitism the best individual of P_i is inserted into P_{i+1} .

Our initial experiments show that classical GA operators are more time-consuming and that they do not find any solution for SAT in many cases. Figure 3 shows the typical behaviour of standard mutation and M_B (in combination with some crossover operators) for one specific representative large instance.

A GA using bit mutation⁴ is not able to locate the optimum, while M_B makes the GA solve the instance very quickly for all crossovers. In general, bit mutation

³ We do not employ a standard generator for SAT instances. However, we observe that the hardest instances produced by our generator are found at $m/n \approx 4.3$, too.

⁴ As a high mutation rate causes disruptive behaviour, we use a lower mutation probability for bit mutation. The probability is chosen to be 0.01 which seems to exhibit the best performance.

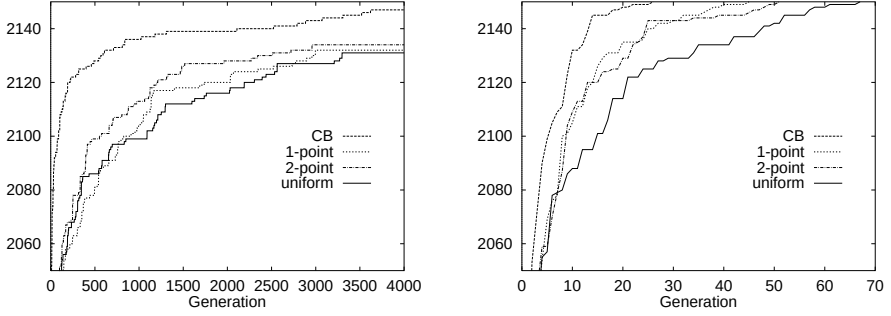


Fig. 3. Fitness progress of standard bit mutation (left) and M_B (right) for different crossover operators and one instance with $m = 2150$, $n = 500$ and $k = 3$

sometimes succeeds for smaller instances. However, it is clearly dominated by M_B which is faster and more robust in finding a solution. The crossover operator C_B speeds up the search and yields higher fitness values than the considered classical crossovers. One-point and two-point crossover seem to perform better than uniform crossover for this instance, but in general these crossovers show a very similar behaviour. To sum it up, M_B and C_B form the best observed combination of mutation and crossover operators.

Let us now compare the effect of the refining functions. For each number of variables we generate 10 instances and give each fitness function 10 runs. The numbers of generations needed to find a satisfying assignment are averaged over instances and runs, and are given in Table 1.

Table 1. Results for the bit string representation (population size 12)

m	n	fit_B	$fit_B + ref_{B1}$	$fit_B + ref_{B2}$	$fit_B + ref_{B3}$	$fit_B + ref_{B4}$
430	100	13.24	12.42	12.27	12.62	15.28
1075	250	23.88	21.10	20.31	19.72	20.01
2150	500	33.82	33.24	33.09	32.47	32.02
4300	1000	43.17	42.00	41.56	45.46	40.71
8600	2000	60.31	66.30	64.15	61.28	72.62

The five functions yield very similar performance. In many cases adding the refining function to fit_B yields small improvements, but there are also some cases where fit_B outperforms the refined fitness functions. For $n = 2000$, fit_B is superior to all refined fitness functions. One might expect an increasing influence of the refining functions for higher population sizes. This cannot be confirmed by experiments with population size 80: The differences between the five fitness

functions are even smaller than in Table 1.

We conclude that the use of refining functions not necessarily leads to improved performance when using the bit string representation. A reason for this might be the fact that the basic fitness function gives enough information to the GA. As it seems that the approach of static refining functions is not appropriate, one might guess that more elaborated and flexible refining functions could yield better results. More flexibility could be gained by dynamic or adaptive refining functions which opens a new direction for further research.

However, we observe good performance for all fitness functions. This mainly depends on the genetic operators which are very robust (as a solution has been found in all cases). The operators M_B and C_B make the GA solve even instances with $n = 2000$ within less than 75 generations, i.e. less than 900 evaluations are needed. Furthermore, the number of needed evaluations grows sublinear for increasing n . This should make the algorithm suitable for even larger instances.

4 The Path Representation

4.1 Representation and Basic Fitness Function

Any variable assignment solving a SAT instance must satisfy at least one literal in each clause. Thus, we may select one literal per clause that should be satisfied by a variable assignment. We call a sequence $p = (p_1, \dots, p_m)$ of indices $p_i \in \{1, \dots, k\}$, which select one literal within each clause, a *path*. In a path, two indices corresponding to the literals $\neg x_l$ and x_l cause an *inconsistency*, because there does not exist a variable assignment that satisfies both literals. A *feasible path* is a path without inconsistencies. Feasible paths can be used to construct a variable assignment for the SAT instance.

To clarify these concepts by an example, we consider the SAT instance with $m = 4$, $n = 4$, $k = 3$ and the boolean formula

$$(x_1 \vee \neg x_3 \vee \neg x_4) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_4)$$

which is illustrated in Fig. 4. The path $p = (1, 4, 2, 4)$ (which selects the literals $x_1, x_4, \neg x_2, x_4$ in the clauses c_1, c_2, c_3, c_4 , respectively) is shown by the lines between the clauses (depicted by ovals). This path p is feasible and hence, we can construct the feasible variable assignment $x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 1$ from p (as no literal with variable x_3 is selected by p , we may choose an arbitrary boolean value for x_3). On the other hand, the path $p' = (4, 4, 1, 1)$ contains two inconsistencies which prevent the construction of a feasible variable assignment from p' . Thus, p' is not feasible.

In general, we should search for a path with a minimal number of inconsistencies. If even the path with the least number of inconsistencies contains an inconsistency, then the SAT instance is not satisfiable. Otherwise, if there exists at least one feasible path, then there exists at least one feasible variable assignment. Paths containing lots of inconsistencies are judged as “bad” because many changes are necessary to obtain a path with minimal number of

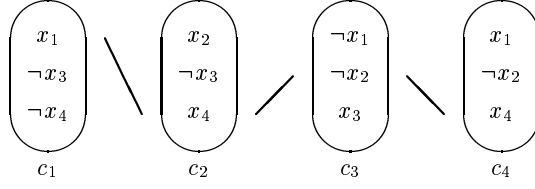


Fig. 4. An example for a path in a boolean formula in conjunctive normal form

inconsistencies. On the other hand “good” paths contain only few inconsistencies. Hence, it is convenient to define our basic fitness function to be maximized as $fit_P(p) = “w - \text{number of inconsistencies in } p”$ for a sufficient high constant $w \in \mathbb{N}$, which ensures positive fitness values.

For each path p , we can determine the set of variable assignments that can be constructed from p . This set contains 2^t ($t \geq 0$) solutions if p is feasible, and is empty otherwise. Here, t denotes the number of variables that are not selected by the path. A comparison with the bit string representation leads to another observation. The size of the path search space is k^m and therefore independent of the number n of variables – in contrast to the size 2^n of the bit string space which is independent of k and m . Hence, it could be justifiable to prefer the path representation for growing n and decreasing k and m , and bit strings for increasing k and m and decreasing n .

4.2 The Refining Functions

Like for the bit string representation it is possible to define refining functions that make the fitness landscape more distinguishable. We use basically the same idea as for fit_{B1} , fit_{B2} and fit_{B3} and thus, we only briefly sketch the refining functions for the path representation. Given a path p and a literal in p , we count the number of occurrences of this literal in f . The ratio of this number and the number of total occurrences of the corresponding variable (positive and negative) is computed. $ref_{P1}(p)$ is the normalized sum of the ratios for all literals in p . The normalization ensures that the basic fitness function remains the dominating quality criterion in $fit_P + ref_{P1}$. Note that ref_{P1} can be seen as a path representation equivalent to ref_{B1} . The other refining functions ref_{P2} and ref_{P3} are calculated in a similar fashion and correspond directly to ref_{B2} and ref_{B3} , respectively.

4.3 Mutation and Crossover

The simplest possible mutation operator just makes random changes of p_i for some $i \in \{1, \dots, m\}$ such that $p = (p_1, \dots, p_m)$ remains a path.⁵ Experimental

⁵ Note that not every random change yields a path: Suppose a change of p_i to a variable that is not contained in the corresponding clause c_i .

results indicate the poor performance of this operator which we call random change mutation. Hence, we propose a mutation that works more goal-oriented. Our operator M_P checks all clauses and selects for each clause c_i a subset L of the contained literals, where each literal is selected with probability p_M . The procedure determines one literal $j \in L$ which causes the least number of inconsistencies in p . If this literal j causes less inconsistencies than p_i in p , the child gets the value $\bar{p}_i = j$. It should be remarked that the traversing order of the clauses has no effect on the resulting child, therefore M_P checks the clauses in sequential order.

```

procedure MUTATION  $M_P$  (parent path  $p$ )
  set  $\bar{p} = p$ 
  for  $i = 1$  to  $m$  do
    set  $L = \emptyset$ 
    for variables  $x_j$  contained in clause  $c_i$  do
      if  $\text{random} < p_M$  then set  $L = L \cup \{j\}$ 
    let  $j$  be a literal from  $L$  that causes the least number of inconsistencies in  $p$ 
    if literal  $j$  causes less inconsistencies than  $p_i$  in  $p$ 
      then set  $\bar{p}_i = j$ 
return child  $\bar{p}$ 

```

Fig. 5. Pseudocode for mutation operator M_P

We have adapted the ideas of the bit string crossover C_B to the path representation. The resulting operator C_P checks each clause c_i with probability $r \in (0, 1]$ and transfers information between the parents p and q as follows. If p_i causes less inconsistencies in parent p than q_i in parent q , then the value p_i is copied to child $\bar{q}$, i.e. $\bar{q}_i = p_i$. In the other case the value q_i is transferred to $\bar{p}$, i.e. $\bar{p}_i = q_i$. Above remark about the traversing order of the clauses in M_P is valid for C_P , too. It is obvious that the produced children have more similarities than the parents. Hence, the same observation as for the bit string representation can be made – M_P is able to alter these children significantly which helps overcome premature convergence.

4.4 Computational Results

We use the same GA with some changed parameters which we found suitable for the path representation (population size 20, $p_C = 0.7$, $r = 0.65$).⁶ The GA is stopped when a generation limit of 4000 is reached. First, we compare our operators M_P and C_P with random change mutation and classical crossovers, respectively. The results for one representative instance are shown in Fig. 7. Crossover

⁶ The probabilities for random change mutation are selected as 0.02 and 0.01 for C_B and the classical crossovers, respectively. The corresponding probabilities for M_P are chosen as 0.5 and 0.3.

```

procedure CROSSOVER  $C_P$  (parents  $p, q$ )
  set  $\bar{p} = p$  and  $\bar{q} = q$ 
  for  $i = 1$  to  $m$  do
    if  $\text{random} < r$  then
      if  $p_i$  causes less inconsistencies in  $p$  than  $q_i$  in  $q$ 
        then set  $\bar{q}_i = p_i$ 
        else set  $\bar{p}_i = q_i$ 
  return children  $\bar{p}$  and  $\bar{q}$ 

```

Fig. 6. Pseudocode for crossover operator C_P

C_P yields a faster increase of fitness for both mutation operators. Nevertheless all crossovers fail within 4000 generations when random change mutation is used. Tests with a higher generation limit show that random change mutation finds a solution in only very few cases. Independent of the used crossover, the time needed to obtain a solution is unpredictable and can be arbitrary large. M_P shows a better behaviour in contrast to random change mutation, especially in combination with C_P (a solution is found very quickly, see Fig. 7). While the classical crossovers sometimes fail (1-point and uniform crossover do not find a solution in 4000 generations for the considered instance), C_P makes the use of M_P more effective.

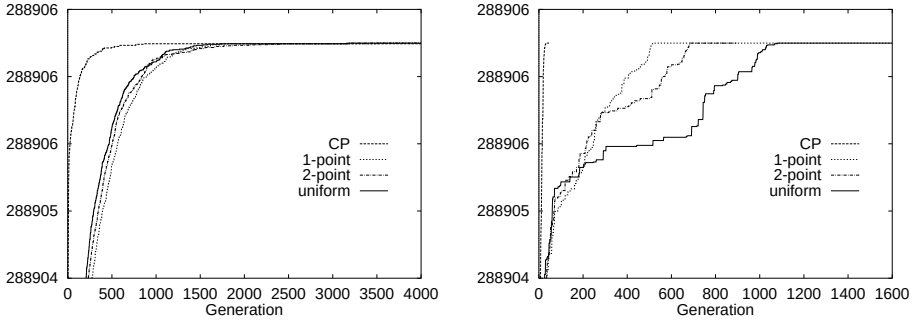


Fig. 7. Fitness progress of random change mutation (left) and M_P (right) for different crossover operators and one instance with $m = 1075$, $n = 250$ and $k = 3$

We test the fitness functions for the path representation on the same test set as the bit string representation. The results are presented in Table 2. It is important to remark that in ca. 2 % of all runs the GA fails to find a solution (the failure probability increases for larger instances). These runs are contained (with generation limit 4000) in the averages given in Table 2.

There is no clearly dominant refining function. In many cases the refined

Table 2. Results for the path representation (population size 20)

m	n	fit_P	$fit_P + ref_{P1}$	$fit_P + ref_{P2}$	$fit_P + ref_{P3}$
430	100	48.45	43.78	43.34	46.14
1075	250	114.24	71.57	118.30	209.75
2150	500	264.25	266.72	211.75	186.21
4300	1000	296.52	242.27	203.25	207.17
8600	2000	547.05	361.77	461.86	497.11

functions show better behaviour than fit_P – but not in all cases. However, for large instances the basis fitness function fit_P is inferior.

For the results with population size 80 see Table 3 (in 0.5 % of all runs the GA fails to find a solution). We observe that an increase of the population size makes the refined fitness functions dominate fit_P . The effect is getting higher for larger instances. Thus, for the path representation it makes sense to use refining functions. One reason for this might be that the basic fitness function induces a not sufficient continuous fitness landscape. Another reason could be a general difficulty to leave local optima in this search space. Hence, the GA needs additional information to avoid getting trapped in local optima. The danger of being mislead by fit_P can be diminished by the refining functions.

Table 3. Results for the path representation (population size 80)

m	n	fit_P	$fit_P + ref_{P1}$	$fit_P + ref_{P2}$	$fit_P + ref_{P3}$
430	100	21.31	20.37	22.32	21.29
1075	250	62.81	28.76	29.43	29.61
2150	500	152.45	118.23	74.86	45.24
4300	1000	76.92	51.76	45.62	45.36
8600	2000	244.15	65.91	80.14	96.25

A comparison with the bit string based GA shows the clear inferiority of the path representation. Even instances with $n = 100$ need about 50 generations of size 20 (i.e. 1000 evaluations) – that is more than the bit string GA needs for instances with $n = 2000$. As some instances are not solved, we may conclude that the path representation is much less robust than the bit string representation.

5 Conclusions

We have presented two representation schemes for SAT, together with refined fitness functions and new problem specific genetic operators. One part of our

investigation aims at the incorporation of additional heuristic information into the fitness function. The bit string GA seems to be insensitive to these refined fitness functions. On the other hand, the performance of the GA for the path representation is improved by refining functions. This effect increases with higher population sizes and larger instances. It might be interesting to design other refining functions: As the presented functions are static, it could be worthwhile to investigate dynamic or adaptive refining functions. Perhaps these more flexible types of refining functions could even improve the performance for the bit string representation.

Besides the concept of refined fitness functions, new genetic operators form the second aspect of our study. The obtained results are strongly influenced by our problem specific operators. This can easily be verified by the comparison with standard operators (e.g. bit mutation, 1-point, 2-point and uniform crossover) which exhibit a clearly inferior behaviour. Thus, it could be promising to further improve our operators. One way to achieve this could be the use of even more problem specific knowledge, or the incorporation of local optimization capabilities. It should be noticed that our crossover and mutation operators can easily be adapted to other constraint satisfaction problems. It could be worthwhile to investigate this in a later study, as their success for these problems might give more insight into possible improvements.

The third aspect of this study is the comparison between the bit string and the path representation for SAT problems. There is a clear winner in this competition of the two representations: the bit string representation. Probably, the reason is that it is the most natural representation for SAT which enables us to use an accurate fitness function. From the view of GA constraint handling techniques (see [Mic96] for a survey) the path representation has similarities with the decoder approach. Such approaches often suffer from a lack of continuity in the search space, which may explain the need for additional information (e.g. given by a refining function).

To sum it up, the results indicate that despite the use of refining functions the path representation is dominated by the most natural representation, the bit string representation. However, it must not be neglected that a representation can only be successful if there are good genetic operators available.

References

- [DP60] M. Davis and H. Putnam. A Computing Procedure for Quantification Theory. *Journal of the ACM*, Volume 7, 201 – 215, 1960
- [DJS89] K. A. De Jong and W. M. Spears. Using Genetic Algorithms to Solve NP-Complete Problems. In J. D. Schaffer (ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, 124 – 132, Morgan Kaufmann Publishers, San Mateo, CA, 1989
- [EH96] A. E. Eiben and J. K. van der Hauw. Graph Coloring with Adaptive Genetic Algorithms. Technical Report 96-11, Department of Computer Science, Leiden University, 1996

- [EH97] A. E. Eiben and J. K. van der Hauw. Solving 3-SAT with Adaptive Genetic Algorithms. In *Proceedings of the 4th IEEE Conference on Evolutionary Computation*, 81 – 86, IEEE Service Center, Piscataway, NJ, 1997
- [FF96] C. Fleurent and J. A. Ferland. Object-oriented Implementation of Heuristic Search Methods for Graph Coloring, Maximum Clique and Satisfiability. In D. S. Johnson and M. A. Trick (eds.), *Cliques, Coloring and Satisfiability: 2nd DIMACS Implementation Challenge*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Volume 26, 619 – 652, 1996
- [Fra94] J. Frank. A Study of Genetic Algorithms to Find Approximate Solutions to Hard 3CNF Problems. Golden West International Conference on Artificial Intelligence, 1994
- [Fra96] J. Frank. Weighting for Godot: Learning Heuristics for GSAT. In *Proceedings of the 13th National Conference on Artificial Intelligence and the 8th Innovative Applications of Artificial Intelligence Conference*, 338 – 343, 1996
- [Fra97] J. Frank. Learning Short-Term Weights for GSAT. Submitted to 15th International Joint Conference on Artificial Intelligence, 1997
- [GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco, CA, 1979
- [Gu94] J. Gu. Global Optimization for Satisfiability (SAT) Problem. *IEEE Transactions on Knowledge and Data Engineering*, Volume 6, Number 3, 361 – 381, 1994
- [Hao95] J.-K. Hao. A Clausal Genetic Representation and its Evolutionary Procedures for Satisfiability Problems. In D. W. Pearson, N. C. Steele, and R. F. Albrecht (eds.), *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms*, 289 – 292, Springer, Wien, 1995
- [Mic96] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Third Edition, Springer, 1996
- [MSL92] D. Mitchell, B. Selman, and H. Levesque. Hard and Easy Distributions of SAT Problems. In *Proceedings of the 10th National Conference on Artificial Intelligence*, 459 – 465, 1992
- [Par95] K. Park. A Comparative Study of Genetic Search. In L. J. Eshelman (ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms*, 512 – 519, Morgan Kaufmann, San Mateo, CA, 1995
- [SKC94] B. Selman, H. A. Kautz, and B. Cohen. Noise Strategies for Improving Local Search. In *Proceedings of the 12th National Conference on Artificial Intelligence*, 337 – 343, 1994
- [SLM92] B. Selman, H. Levesque, and D. Mitchell. A New Method for Solving Hard Satisfiability Problems. In *Proceedings of the 10th National Conference on Artificial Intelligence*, 440 – 446, 1992

Genetic algorithms at the edge of a dream

Cathy Escazut¹ and Philippe Collard²

¹ Evolutionary Computation Research Group,
Napier University
219 Colinton Road,
Edinburgh, EH14 1DJ — SCOTLAND
e-mail: cathy@dcsc.napier.ac.uk
or escazut@unice.fr

² Laboratory I3S — CNRS-UNSA
250 Av. A. Einstein,
Sophia Antipolis, 06560 Valbonne — FRANCE
e-mail: pc@unice.fr

Abstract. This paper describes a *dreamy genetic algorithm* scheme, emulating one basic mechanism of chronobiology: the alternation of awake and sleeping phases. We use the metaphor of the *REM sleep* during which the system is widely disconnected from its environment. The dream phase allows the population to reorganize and maintain a needed diversity. Experiments show that dreamy genetic algorithms improve on standard genetic algorithm, for both stationary (deceptive) and non-stationary optimization problems.

A theoretical and experimental analysis suggests that dreamy genetic algorithms are better suited to complex tasks than standard genetic algorithms, due to the preservation of the population diversity.

1 Introduction

We propose to implement within Genetic Algorithms (GAs), one of the basic mechanisms of the chronobiology: the *REM sleep*. Neurobiologists established that states of sleep in human beings are the achievement of a long evolution [13]. Taking our inspiration from this work, we dictate *circadian* cycles to the genetic population with an external control. Such an approach has not been yet explored in an artificial context, on the one hand because only a few evolutionary theories are interested in the problem of the function of the dream, and on the other hand because the metaphorical basis of GAs is extremely simplified in comparison with the natural model.

Dreamy-GAs are implemented within the framework of *dual-GAs* presented in [3, 5, 4, 6]; more precisely, they alternate awake and dream phases. The awake phase corresponds to the standard mode of dual-GA; the dream phase allows the population to reorganize in order to maintain its diversity.

We first briefly present dual-GAs and the effects of the duality on crossover and mutation operators. Then, we introduce *dreamy-GAs*, their implementation

and an analyze of their effects. Finally, we present some experimental results on deceptive functions and non-stationary environments.

2 Dual genetic algorithms

The problem of premature convergence is usually addressed by explicitly enforcing the population diversity via *ad hoc* operators or selection procedures [10, 14, 15]. This contrasts with dual-GAs keeping standard operators and selection procedures: dual-GAs rather enforce diversity at the representation level.

2.1 Basic principles

The form of dual Genetic Algorithms is the same as the one of standard GAs. They only differ in the representation of the individuals. Within dual-GAs, individuals are enhanced with an additional bit, termed *head bit*, that controls the interpretation of the individual and the computation of its fitness. More precisely, individual $\underline{0}\omega$ is intended as ω , while individual $\underline{1}\omega$ is intended as $\bar{\omega}$, the complementary of ω . In the new genotypic space Ω^* given as $\{0, 1\} \times \Omega$, where Ω is the standard genotypic space, we distinguish *chromosomes* (elements of 0Ω) and *anti-chromosomes* (elements of 1Ω). For instance, *dual individuals* $\underline{0}0100$ and $\underline{1}1011$ (complementary to each other) both represent the single individual 0100: they form a *dual pair*.

This increased search space allows an improved use of the schemas (see [5] for more details). Let us now focus on another characteristic induced by dual-GAs: *implicit mutations*.

2.2 Implicit mutations

Within conventional GAs the mutation rate is mostly handled as a global external parameter which is constant over time [19]. We showed in [4], that using a dual-GA with only the crossover operator, mutations are *implicit*. Indeed, within dual-GAs, crossing over two dual individuals 0ω and $1\bar{\omega}$ actually amounts to mutating ω . It is well known that standard crossover can be viewed as a biased mutation [7, 12, 18], with the advantage that the corresponding mutation rate is automatically controlled from the diversity of the population; but inversely, crossover does not allow for restoring a population diversity.

This limitation of standard crossover disappears in dual-GAs, provided that there exists dual pairs (0ω and $1\bar{\omega}$) in the population. For instance, let us consider the individual 101 represented by the dual pair ($\underline{0}101, \underline{1}010$). If a crossover applied on locus 2 to the pair is followed by a crossover applied on locus 3 of obtained offspring, i.e. ($\underline{0}110$ and $\underline{1}001$), we have the two individuals $\underline{0}111$ and $\underline{1}000$ representing the individual 111. These two consecutive dual crossovers have the same effect than an *explicit* mutation applied to the individual 101 on the second locus.

The implicit mutation rate due to the crossover effects therefore depends on the number of dual pairs in the population. The reason why the number of dual pairs decreases as the dual-GA converges has been examined in [4]. It follows that the rate of implicit mutations decreases from an initial value to zero. This ensures that the dual-GA achieves some tradeoff between the exploration and exploitation task: it works on the borderline of efficient optimization and almost random walk.

The *Implicit Mutation Rate* is defined as follows:

$$Imr = p_c \cdot \sum_{\omega \in \Omega} \min(P(\underline{0} \omega), P(\underline{1} \bar{\omega}))$$

where p_c is the crossover probability, $P(\xi)$ the proportion of the individual ξ in the population.

Populations having for each chromosome, the corresponding counterpart in the same proportion ($P(\underline{0} \omega) = P(\underline{1} \bar{\omega})$) are called *mirror* populations. We note that such populations have an optimal implicit mutation rate ($Imr = 0.5 \times p_c$).

2.3 The mirroring operator

The mirroring operator transforms any individual in its complementary: the mirror image of $\underline{1} 0100$ is $\underline{0} 1011$. It applies on each individual in the population with a small probability (usually around .01). Thereby, it introduces genotypic diversity while preserving the phenotypic distribution of the population. Used together with crossover operator, it expectedly allows for a new dynamic equilibrium, balancing genotypic diversity and phenotypic convergence.

The mirroring induces some interesting properties. Here are the most important ones:

1. A mirror population is invariant under the mirroring operator.
2. The space of the mirroring operators is *closed* under composition¹.
3. Repeated uses of the mirroring operator alone gives a mirror population.

Let us now present why and how we defined *Dreamy-GAs* from the basis of dual-GAs.

3 Dreamy genetic algorithms

Our goal is to design a GA with persistent diversity. Taking a clue from the basic mechanism of chronobiology, we now introduce periodic dynamics in dual-GAs: we dictate circadian cycles (*autogenous rhythm*) to the genetic population. Awake and dream phases are then integrated to the course of dual-GAs.

¹ The composition of two mirroring operators applied with a rate respectively sets to τ_1 and τ_2 , is the mirroring operator whose rate is $\tau_1 + \tau_2 - 2\tau_1\tau_2$.

3.1 The dream phase

The alternation of awake and dream phases should allow the dreamy-GA to converge, while keeping its ability to explore the search space. Although natural awake/dream cycles have to do with individuals, the dream metaphor makes sense for the population as well, considered as a single individual facing a circadian environment. Dreamy-GAs are (loosely) inspired from the *REM sleep*, during which the system is widely disconnected from its environment and there are only a few interactions between both cerebral hemispheres. Here, the spaces of chromosomes and anti-chromosomes 0Ω and 1Ω are taken as analogs of hemispheres: no interaction between those hemispheres is allowed during dream phases.

More precisely, *dual crossovers* (i.e. crossovers combining individuals of both half spaces) are inhibited during the dream phase : a crossover is only allowed between two chromosomes or two anti-chromosomes. Let us note that the space of chromosomes and the space of anti-chromosomes as well are closed under crossover. This approach resembles restricted mating [16] in the sense that it forbids the mating of individuals in different sub-populations. Still, the sub-populations of chromosomes and anti-chromosomes exchange information via the selection procedure, and overall, via the mirroring operator.

3.2 A theoretical analysis of dreamy-GAs

New genetic algorithms schemes usually undergo an experimental validation on a set of well-studied benchmark functions [8]. However, experimental studies suffer from a number of biases, e.g. the different control parameters of the GA [11, 17].

Therefore, we first analyze the dynamics of dreamy-GAs using the Whitley's executable model [23]. It uses an infinite population, no mutation and requires the enumeration of all points in the search space. Assuming proportionate reproduction, one point crossover, and random mating, an equational system can be used for modelling the proportion of the individuals (remind that this analysis is exhaustive) over time. Let us note that it is possible to develop exact equations for problems up to 15 bits in the coding. These equations generalize those given by Goldberg as far as the *Minimal Deceptive Problem* [9] is concerned. This system may be compared to the Vose and Liepins' formalism and their interpretation using matrices [21].

We simulate the behavior of a dual-GA under effects of selection, crossover and mirroring, alternating awake and dream phases. The fitness function we use involves the fully deceptive *F2* function, defined over 4 bits [22]; the function values for the 16 individuals are:

$F2(0000) = 28$	$F2(1000) = 20$
$F2(0001) = 26$	$F2(1001) = 12$
$F2(0010) = 24$	$F2(1010) = 10$
$F2(0011) = 18$	$F2(1011) = 2$

$F2(0100) = 22$	$F2(1100) = 8$
$F2(0101) = 16$	$F2(1101) = 4$
$F2(0110) = 14$	$F2(1110) = 6$
$F2(0111) = 0$	$F2(1111) = 30$

The fitness function studied here is composed of 2 copies of $F2$. Thus, individuals are composed of 8 bits. The fitness is the sum of the $F2$ fitness of the first 4 bits, and the $F2$ fitness of the last four bits, divided by 2. The optimal individual is 11111111, with fitness 30. In the dual-GA framework, individuals are composed of 9 bits, the two optimal individuals being 0 11111111 and 1 00000000. Two different simulations are performed.

Simple transitions:

Here, the crossover probability is set to 1 and the mirroring rate to .01. Each phase lasts for 250 generations. During an awake phase the population converges toward a polymorphic population containing only one dominant individual: the individual 0 11111111 (see figure 1). The Imr goes to 0. During the dream phase, the population goes to a mirror one in which only one chromosome and its anti-chromosome coexist in the same proportions. This dual pair represents the optimum. This limit population is phenotypically homogeneous while presenting a maximum genotypical diversity.

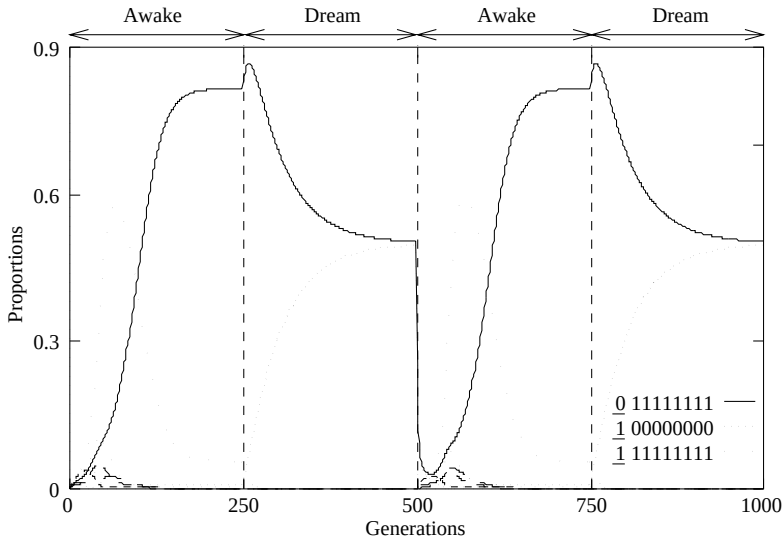


Fig. 1. Simple transitions: Evolution of proportions of best individuals.

If an individual comes to dominate the population during an awake phase, the corresponding dual pair appears during the following dream phase, and comes

to dominate the mirror population. We note that during a dream phase the *Imr* goes to its upper bound. In consequence the system maximizes its exploration abilities. This maximization is not performed to the detriment of the exploitation abilities since the dominant individual goes on reproducing. Moreover, during this dream phase if changes in the environment occur, the system becomes less and less sensitive to these changes. Indeed, the population goes to contain only two dual individuals without any possibility of crossover between them; this is in harmony with the metaphor of the *REM sleep*.

Note the smooth transition from awake to dream phases. The inverse transition is indeed abrupt; it reveals a phenomenon of *hysteresis*, and depending on the duration of the dream the transition is not performed on the same population. Such a transition can be interpreted as a *break of symmetry* [2].

Note also that the population in the first generations of an awake phase is fairly close to a uniform distribution (with all individuals being equally represented).

The results of this simulation show that the number of dual pairs dynamically adapts itself during the genetic search: it decreases during the awake phase and increases during the dream phase. So does the ratio exploitation/exploration. A second simulation is concerned with continuous transitions.

Continuous transitions:

In this simulation, dual crossovers apply with a gradually increasing probability p_{dc} , given as:

$$\begin{array}{ll} \text{IF} & (g < \frac{P}{4}) \\ \text{THEN} & p_{dc} = 1 \\ \text{ELSE} & p_{dc} = \cos\left(\pi \cdot \left(\frac{g}{P} - \frac{1}{4}\right)\right)^2 \end{array}$$

where P is the alternation period between phases (1000 in our test), and g is the current generation. Figure 2 shows that as p_{dc} decreases to 0, the *Imr* increases toward its upper bound (about generation 800): the population diversifies. Then it converges toward a polymorphic population dominated by the optimum as p_{dc} goes to 1 (about generation 1000). Finally, with the decrease of p_{dc} , the population converges toward a mirror one containing the optimum individuals.

4 Experimental results

The dreamy-GA is validated on a stationary deceptive problem, and a non-stationary problem. Both are known to be hard to be optimized with a GA.

4.1 A deceptive function

The function used is obtained from 16 copies of the Whitley's *F2* function. Individuals are represented by 64 bits for the standard GA, and 65 bits for the

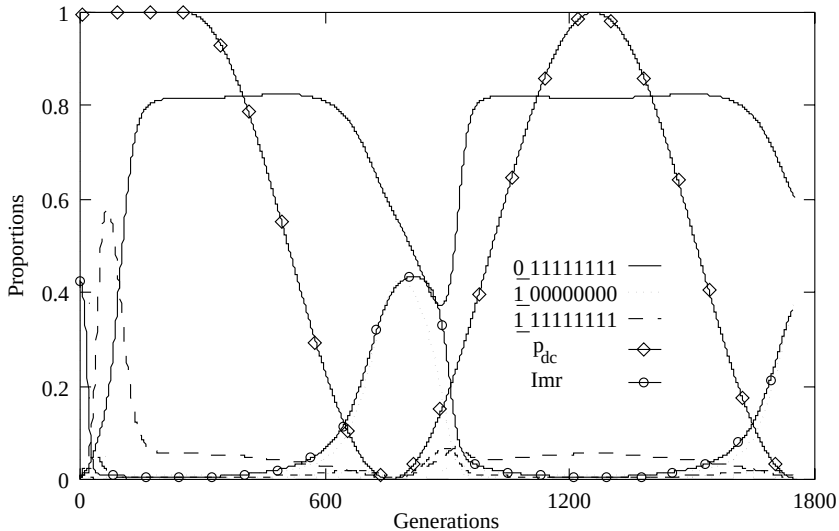


Fig.2. Continuous transitions: Evolution of proportions of best individuals.

dual and dreamy-GAs. Populations are composed of 100 individuals. The reproduction is controlled by a version of the roulette wheel proportional selection. We use a single-point crossover which probability is set to 1. The mutation rate is set to .9 per chromosome. In the case of the dual and dreamy-GAs, the mirroring rate is fixed to .02. The awake phase lasts for 20 generations; the dream phase, for 5 generations. Results are averaged on 50 independent runs. Figure 3 shows the dynamics of evolution (average best fitness reached for a number of generations). For convenience, only fitnesses between 27 and 30 are plotted.

Note that all three GAs behave the same during the first 200 generations, with a slight advantage for standard GA over dual-GA. About the generation 230, the dreamy-GA breaks away from the two other GAs whose performances are very close. Finally, the dual-GA has the upper hand and tends to come up with the dreamy-GA.

4.2 Non-stationary environments

The tradeoff between exploration and exploitation is more than ever crucial when dealing with non-stationary environments. Dream phases are purposely devised to maintain population diversity (i.e. preserve the exploration ability), while minimizing the phenotypic disruption (loss of fittest individuals). The dynamic environment we used for this test is the *pattern tracking* problem proposed by Vavak and Fogarty [20].

Using this dynamic function, the fitness of an individual is its Hamming distance to a target individual. This target individual is the optimum, and the environment changes as the target individual is modified. The difficulty of change

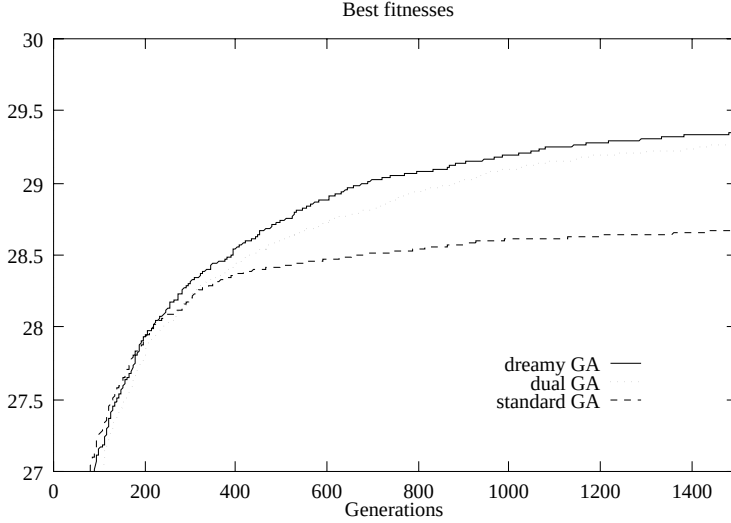


Fig. 3. Fitness evolution for 16 copies of the Whitley’s $F2$ function.

relates to the Hamming distance between two consecutive target individuals, which is gradually set to 10, 20, 30 and 40.

The size of the problem is $N = 40$, with population size 50. We use single point crossover with a probability of 1. The mutation rate is .9 for the standard GA, and .2 for the dual and dreamy-GAs; in the latter cases, the mirror rate is .02. The awake time is set to 40 generations and the dream time to 10. Every 50 generations, a change of optimum occurs, so the change is performed while the population has quasi converged. The results are averaged over 50 independent runs. Figure 4 shows the average best fitness reached for a given number of generations.

During the first period (generation 1 to 50), all GAs present similar performances, with a slight advantage for the standard GA. The 10 last generations of this period (generations 40 to 50) correspond to a dream phase. The change that occurs at generation 50 is a small one (the target individuals only differ by ten bits), and all GAs appear equally able to follow the optimum, with a slight advantage for the dreamy-GA. The standard GA appears unable to cope with greater changes (generations 100, 150 and 200): it needs at least 40 generations to catch up. Note that the performance fall is proportional to the magnitude of the change for the standard GA, whereas it is limited to the half size of the problem (20) for the dual-GA (due to the fact that if x and y are very different, then x is close to the dual of y) [6].

These experiments show that dreamy-GAs outperform standard GAs in a non-stationary environment. Indeed a dreamy-GA offers a *natural* way for tracking down changing optima. Its better performance can be explained by the fact that during the dream phase the system tends to maximize its abilities to ex-

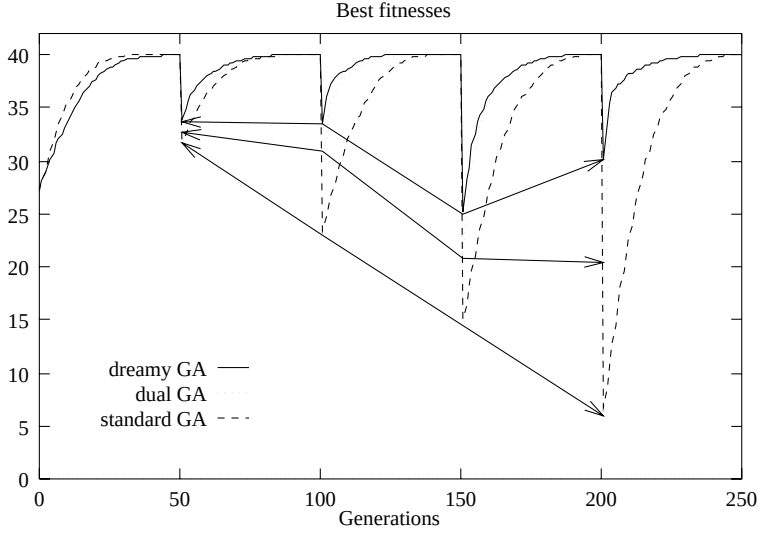


Fig. 4. Fitness evolution for the non-stationary environment.

plore, while keeping its exploitation capabilities. This confirms that changing environment can benefit from periodic reorganization of the population, in order to break symmetries [1].

Let us now, focus on how diversity evolves. Figures 5 represents respectively the genotypical and phenotypical diversities. The genotypical diversity is the mean of the normalized Hamming distance across all pairs of chromosomes in the population. In a random population this value will approximately be .5. For the dreamy-GA the genotypic diversity increases during dream phases, while the phenotypic diversity decreases to the standard one. This definitely proves its abilities to explore without endanger the quality of acquired information. A change in the environment leads all GAs to increase diversity (phenotypic in the case of standard GA, and both phenotypic and genotypic in the case of dual and dreamy-GAs). This period of increased diversity lasts longer for standard GAs than for the two other GAs; further, the amplitude of diversity is higher for dreamy-GA, than for dual or standard GAs.

5 Conclusion and further work

Our approach aims at overtaking the limits of standard GAs. First, *duality* is introduced in the population thanks to individuals with the same *phenotype* but complementary *genotypes*. Then, one of the mechanisms of chronobiology is implemented: the alternation of awake and dream phases. During the latter the population reorganizes in order to perpetuate a needed diversity. We used the metaphor of the *REM sleep* during which the system is widely disconnected from its environment. A theoretical analysis showed that during the dream phase the

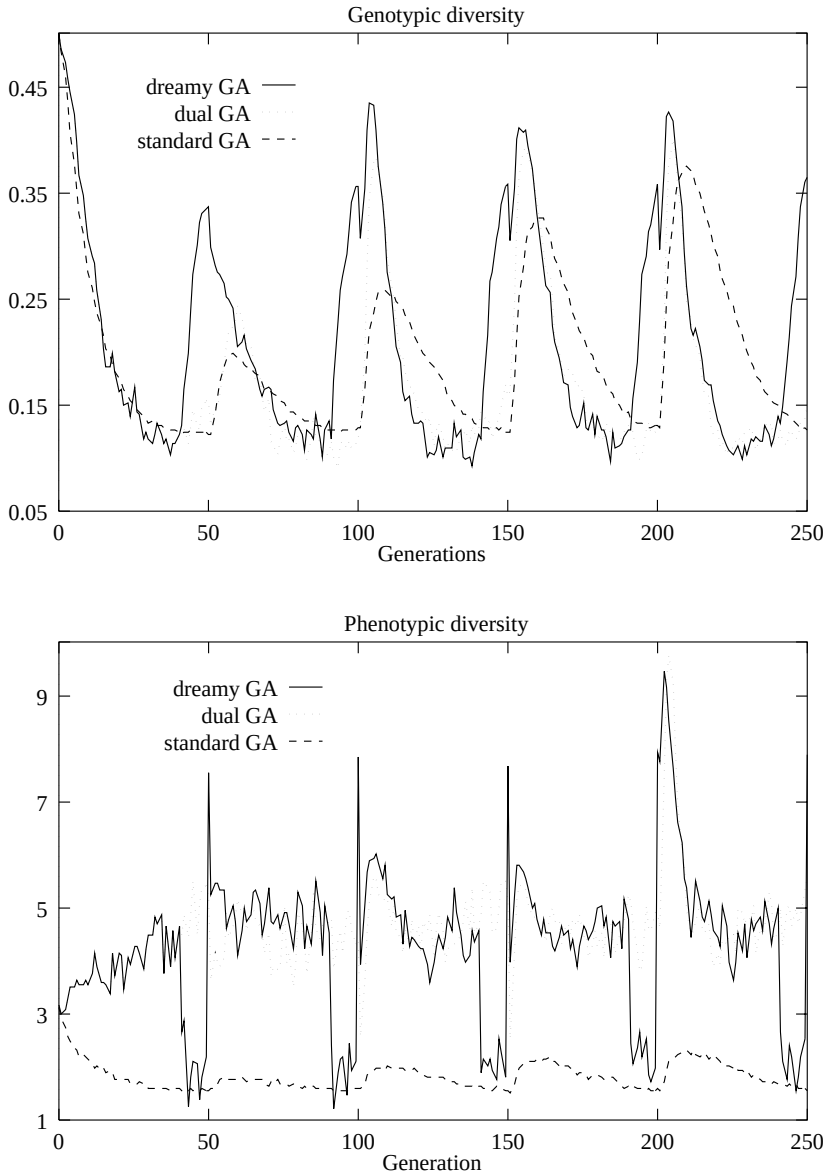


Fig. 5. Diversity evolution for the non-stationary environment.

system optimally maximizes its exploration abilities while going on to exploit available information. The exploration/exploitation ratio is related to the number of dual pairs which is dynamically adapted: it decreases during the *awake* phase and increases during the *dream* one. As a consequence, a dreamy-GA allows one to increase the capabilities of evolution on rugged fitness landscapes.

In the case of a dynamic environment the preservation of diversity causes the dreamy-GA to perform better.

A natural pursuit of this work is to exploit more extensively the basic mechanisms of chronobiology. We could distinguish some rhythmic activities depending on an *endogenous rhythm*. Thus, we could have a biological clock controlled by a gene. We can imagine the profit taken from a synchronization between endogenous and autogenous rhythms.

Acknowledgments

The authors would like to thank Michèle Sebag for her detailed comments and very useful suggestions on earlier versions of this paper. Thanks are also due to the referees for their helpful remarks.

References

1. Eric Bonabeau and Guy Theraulaz. *L'intelligence collective*, chapter 8, pages 225–261. Hermes, 1994.
2. P. Chossat. *Les symétries brisées*. Ed. Belin, 1996.
3. P. Collard and J.P. Aurand. DGA: An efficient genetic algorithm. In A.G. Cohn, editor, *ECAI'94: European Conference on Artificial Intelligence*, pages 487–491. John Wiley & Sons, 1994.
4. P. Collard and C. Escazut. Genetic operators in a dual genetic algorithm. In *ICTAI'95: Proceedings of the seventh IEEE International Conference on Tools with Artificial Intelligence*, pages 12–19. IEEE Computer Society Press, 1995.
5. P. Collard and C. Escazut. Relational schemata: A way to improve the expressiveness of classifiers. In L. Eshelman, editor, *ICGA'95: Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 397–404, San Francisco, CA, 1995. Morgan Kaufmann.
6. P. Collard and C. Escazut. Fitness Distance Correlation in a Dual Genetic Algorithm. In W. Wahlster, editor, *ECAI 96: 12th European Conference on Artificial Intelligence*, pages 218–222. Wiley & Son, 1996.
7. J. Culberson. Mutation-crossover isomorphisms and the construction of discriminating functions. *Evolutionary Computation*, 2(3):279–311, 1995.
8. K. A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, University of Michigan, 1975.
9. D. E. Goldberg. Simple genetic algorithms and the minimal deceptive problem. In L. Davis, editor, *Genetic Algorithms and Simulated Annealing*, pages 74–88. Morgan Kaufmann, Los Altos, California, 1987.
10. D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In J.J Grefenstette, editor, *ICGA'87: Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49. Lawrence Erlbaum Associates, 1987.
11. J. J Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Trans. Systems, Man, and Cybernetics*, 16(1):122–128, 1986.
12. T. Jones. Crossover, macromutation and population-based search. In L. Eshelman, editor, *ICGA'95: Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 73–80. Morgan Kaufmann, 1995.

13. M. Jouv  t. Phylogeny of the states of sleep. *Acta psychlat. belg.*, 94:256–267, 1994.
14. Samir W. Mahfoud. *Niching methods for genetic algorithms*. PhD thesis, University of Illinois at Urbana-Champaign, 1995. IlliGAL Report 95001.
15. C. Melhuish and T. C. Fogarty. Applying a restricted mating policy to determine state space niches using immediate and delayed reinforcement. In T. C. Fogarty, editor, *Evolutionary Computing: AISB Workshop*, volume 865 of *Lecture Notes in Computer Science*, pages 224–237. Springer Verlag, 1994.
16. Edmund Ronald. When selection meets seduction. In L. Eshelman, editor, *ICGA’95: Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 167–173. Morgan Kaufmann, 1995.
17. J. David Schaffer, Richard A. Caruana, Larry J. Eshelman, and Rajarshi Das. A study of control parameters affecting online performance of genetic algorithms for function optimization. In J. D. Schaffer, editor, *ICGA’89: Proceedings of the Third International Conference on Genetic Algorithms*, pages 51–60. Morgan Kaufmann, 1989.
18. M. Sebag and M. Schoenauer. Mutation by imitation in Boolean evolution strategies. In H-M Voigt, W. Ebeling, I. Rechenberg, and H-P Schwefel, editors, *PPSN IV: The Fourth International Conference on Parallel Problem Solving from Nature*, number 1141 in *Lecture Notes in Computer Science*, pages 356–365, 1996.
19. W. M. Spears. Crossover or mutation ? In L. D Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 221–233. Morgan Kaufmann, San Mateo, CA, 1993.
20. F. Vavak and T. C. Fogarty. A comparative study of steady state and generational genetic algorithms for use in nonstationary environments. In T. C. Fogarty, editor, *Proceedings of Evolutionary Computing, AISB Workshop*, number 1143 in *Lecture Note in Computer Science*, pages 297–304. Springer, 1996.
21. M. D. Vose and G. E. Liepins. Punctuated equilibria in genetic search. *Complex Systems*, 5:31–44, 1991.
22. L. D. Whitley. Fundamental principles of deception in genetic search. In G. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 221–241. Morgan Kaufmann, San Mateo, CA, 1991.
23. L. D. Whitley. An executable model of a simple genetic algorithm. In L. D. Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 45–62. Morgan Kaufmann, 1993.

Mimetic Evolution

Mathieu Peyral¹

Antoine Ducoulombier²

Caroline Ravisé^{1,2}

Marc Schoenauer¹

Michèle Sebag^{1,2}

(1): CMAP & LMS
URA CNRS 756 & 317
Ecole Polytechnique
91128 Palaiseau Cedex

(2): Equipe I & A, LRI
URA CNRS 410
Université d'Orsay
91405 Orsay Cedex

{*Prenom.Nom*}@polytechnique.fr

Abstract. Biological evolution is good at dealing with environmental changes: Nature ceaselessly repeats its experiments and is not misled by any explicit memory of the past. This contrasts with artificial evolution most often considering a fixed milieu, where re-generating an individual does not bring any further information.

This paper aims at avoiding such uninformative operations, via some explicit memory of the past evolution: the best and the worst individuals previously met by evolution are respectively memorized within two virtual individuals. Evolution may then use these virtual individuals as social models, to be imitated or rejected. In mimetic evolution, standard crossover and mutation are replaced by a single operator, *social mutation*, which moves individuals farther away or closer toward the models. This new scheme involves two main parameters: the social strategy (how to move individuals with respect to the models) and the social pressure (how far the offspring go toward or away from the models).

Experiments on large-sized binary problems are detailed and discussed.

1 Introduction

Biological evolution takes place in a changing environment. Being able to repeat previously unsuccessful experiments is therefore vital. As the result of previous experiments might change, any explicit memory of the past might provide misleading indications. This could explain why all knowledge gathered by evolution is actually contained in the current genetic material and dispatched among the individuals.

Inversely, artificial evolution most often tackles optimization problems and considers *fixed* fitness landscapes, in the sense that the fitness of an individual does not vary along time and does not depend on the other individuals in the population. In this framework, which is the only one considered in the rest of the paper, the evaluation of an individual produces reliable information, and generating this individual again does not provide any further information. Memorizing somehow the past of evolution thus make sense, as it could prevent evolution from some predictable failures. This paper focuses on gathering an explicit collective memory (EC-memory) of evolution, as opposed to both the

implicit memory of evolution contained in the genetic material in the population, and the local parameters of evolution conveyed by the individuals, such as the mutation step size in Evolution Strategies [23]), or the type of crossover applicable to an individual [26].

Many works devoted to the control of evolution ultimately rely on some explicit collective memory of evolution. The memorization process can acquire numerical information; this is the case for the reward-based mechanism proposed by Davis to adjust the operator rates [5], the adjustment of penalty factors in SAT problems [6] or the construction of discrete gradients [11], among others. The memorization process can also acquire symbolic information, represented as rules or beliefs characterizing the disruptive operators (so that they could be avoided) [20], or the promising schemas [22].

Memory-based heuristics can control most steps of evolution: e.g. selection via penalty factors [6], operator rates [5], operator effects [11, 20]... Memory can even be used to "remove genetics from the standard genetic algorithm" [4, 3] as in the *Population Based Incremental Learning* (PBIL) algorithm. PBIL deals with binary individuals (in $\{0, 1\}^N$) and it maintains the memory of the most fit individual encountered so far. This memory can be thought of as a *virtual individual*, belonging to $[0, 1]^N$. It provides an alternative to the genetic-like transmission of the information between successive populations: any population is generated from scratch by sampling the discrete neighbors of this virtual individual. And the virtual individual is then updated from the best current individual.

Another approach, termed *Evolution by Inhibitions* (EBI), is inversely based on memorizing the worst individuals encountered so far; the memory is also represented by a virtual individual, termed the *Loser* [25]. This memory is used to evolve the current population, by means of a single new operator termed *flee-mutation*. The underlying metaphor is that the offspring aim at being farther away from the loser, than their parents. Incidentally, this evolution scheme is biased against exploring again unfit regions previously explored.

A new evolutionary scheme, restricted to binary search space and combining PBIL and *Evolution by Inhibitions*, is presented in this paper. The memory of evolution is thereafter represented by two virtual individuals, the *Winner* and the *Loser*¹. These virtual individuals, or *Models*, respectively summarize the best and the worst individuals encountered so far by evolution. An individual can independently imitate, avoid, or ignore each one of the two models; a wide range of, so to speak, social strategies, can thereby be considered. For instance, the Entrepreneur imitates the Winner and ignores the Loser; the Sheep imitates the Winner and rejects the Loser; the Phobic rejects the Loser and ignores the Winner (the dynamics is that of Evolution by inhibitions [25]); the Ignorant ignores both models and serves as reference to check the relevance of the models. This new scheme of evolution, termed *mimetic evolution*, is rather inspired by social than genetic metaphors.

¹ Other metaphors, all likely politically incorrect, could have been used: the leader and the scapegoat, the yang and the yin, the knight and the villain,...

This paper is organized as follows. Section 2 briefly reviews related work dealing with virtual or imaginary individuals. Section 3 describes mimetic evolution and the social mutation operator replacing crossover and mutation. Social mutation is controlled from the user supplied social strategy, which defines the preferred direction of evolution of the individuals. Section 4 discusses the limitations of mimetic evolution, and studies the case where the dynamics of the models and the population go in a deadlock. Section 5 examines how far the offspring must go in the direction of the models; or, metaphorically, which social pressure should be exerted on the individuals. Mimetic mutation is validated on several large-sized binary problems, and the experimental results are detailed in section 6. Last, we conclude and present some perspectives of research.

2 State of the art

With no pretention to exhaustivity, this section examines how imaginary or virtual individuals have been used to support evolution. The central question still is the respective contribution of crossover and mutation to the dynamics of evolution [8, 18, 23]. Though the question concerns any kind of search space, only the binary case will be considered here.

The efficiency of crossover is traditionnally explained by the Building Block hypothesis [12, 9]. But a growing body of evidence suggests that crossover is also efficient because it operates large step mutations. In particular, T. Jones has studied the macro-mutation operator defined as crossing over a parent with a random individual². Macro-mutation obviously does not allow the offspring to combine the building blocks of their two parents; still, macro-mutation happens to outperform standard crossover on benchmark problems, everything else being equal [14].

In retrospect, crossover can be viewed as a biased mutation. The bias depends on the population and controls both the strength and the direction of the mutation. The "mutation rate" of standard crossover, e.g. the Hamming distance between parents and offspring, depends on average on the diversity of the population; and the "mutation direction" of standard crossover (which genes are modified) also depends on the population.

On the other hand, binary mutation primarily aims at preserving the genetic diversity of the population. This can be done as well through crossover with specific individuals, deliberately maintained in the population to prevent the loss of genetic diversity. For instance, the Surrogate GA [7] maintains imaginary individuals such as the complementary of the best current individual, or all-0 and all-1 individuals; crossover alone thus becomes sufficient to ensure the genetic diversity of the population, and mutation is no longer needed. Another possibility is to deliberately introduce genotypic diversity by embedding the search space

² Note that this macro-mutation fairly resembles standard crossover during the first generations of evolution, especially for large populations.

Ω into $\{0,1\} \times \Omega$ and identifying the individuals 0ω and $1\bar{\omega}$, as done in Dual Genetic Algorithms [19]. Provided that the number of dual pairs (0ω and $1\bar{\omega}$) is above a given threshold, crossover can similarly replace mutation and ensure genetic diversity.

Evolution can also be supported by virtual individuals, i.e. individuals belonging neither to the population nor to the search space. This is the case in the PBIL algorithm, mentioned in the introduction, where the best individuals in the previous populations are memorized within a vector of $[0,1]^N$. This vector noted $\mathcal{M}$ provides an alternative to crossover and mutation, in that it allows PBIL to generate the current population from scratch: for each individual X and each bit i , value X_i is randomly selected such that $P(X_i = 1) = \mathcal{M}_i$ (where A_i denotes as usual the i -th component of A). $\mathcal{M}$ is initialized to $(0.5, 0.5, \dots, 0.5)$ and it is updated from the best individual³ X_{max} at each generation, by relaxation:

$$\mathcal{M} \leftarrow (1 - \alpha)\mathcal{M} + \alpha X_{max}$$

where α in $[0,1]$ is the relaxation factor, which corresponds to the fading of the memory. The main advantage of PBIL is its simplicity: it does not involve any modification of the genetic material. The only information transmitted from one generation to another is related to the best individual; still, it is not necessarily sufficient to reconstruct this best individual. This might hinder evolution in narrow highly fit regions, such as encountered in the Long Path problem [13]. Practically, one sees that even if $\mathcal{M}$ is close to the path, the population constructed from $\mathcal{M}$ poorly samples the path [24].

Evolution by Inhibition involves the opposite memory, that is, the memory of the worst individuals in the previous populations. This memory noted $\mathcal{L}$ (for *Loser*) is also a vector of $[0,1]^N$, constructed by relaxation:

$$\mathcal{L} \leftarrow (1 - \alpha)\mathcal{L} + \alpha X_{min}$$

where X_{min} denotes the average of half the worst offspring, and α is the relaxation factor. In contrast with PBIL which uses $\mathcal{M}$ to generate a new population, $\mathcal{L}$ is actually used to evolve the current population via a specific operator termed *flee-mutation*. Flee-mutation replaces both mutation and crossover; for each individual X , it selects and flips the bits most similar to those of the loser (minimizing $|X_i - \mathcal{L}_i|$). The offspring thus is farther away from the loser, than the parent was. Metaphorically, the goal of this evolutionary scheme is: Be different from the Loser ! And incidentally, this reduces the chance for exploring again low fit regions.

The potential of evolution by inhibitions is demonstrated for appropriate settings of the flee-mutation rate (number of bits mutated): EBI then significantly outperforms PBIL [3, 25], which itself outperforms most standard discrete optimization algorithms (Hill-Climbers with multiple restarts, standard GAs, binary

³ A more robust variant is to update $\mathcal{M}$ from the *two* best individuals and the worst individual in the population [3].

evolution strategies). But the adjustment of the flee-mutation rate remains an open question.

3 Mimetic evolution

This section focuses on combining evolution by inhibitions and PBIL. Two models memorizing respectively the best and worst individuals met in past generations, are constructed. These models are used to evolve individuals through a single evolution operator.

3.1 Winner-driven evolution of the population

The winner is built by relaxation from the best individuals of the current population, in the same way as in PBIL (section 2). Table 1 illustrates how the winner $\mathcal{W}$ and the loser $\mathcal{L}$ are built from the current population.

	1	2	3	4	5	Fitness	
X	0	0	1	0	0	high	
Y	1	1	1	1	1	high	
Z	0	1	1	0	1	high	
$d\mathcal{W}$	0.33	0.66	1	0.33	0.66		$\mathcal{W} \leftarrow (1 - \alpha_w)\mathcal{W} + \alpha_w d\mathcal{W}$
S	0	0	0	1	0	low	
T	1	0	1	1	1	low	
U	1	0	0	1	1	low	
$d\mathcal{L}$	0.66	0	0.33	1	0.66		$\mathcal{L} \leftarrow (1 - \alpha_l)\mathcal{L} + \alpha_l d\mathcal{L}$

Table 1. *Individuals and virtual individuals*

Let us first examine how $\mathcal{W}$ can help evolving individual X . Given the most fit individuals of the population (X , Y and Z), some possible causes for being fit are ($bit_2 = 1$), or ($bit_3 = 1$), or ($bit_5 = 1$) (a majority of the most fit individuals has those bits set to this value). Thus, one might want for instance to flip bit_2 and let bit_3 unchanged in X ; this amounts to making X more similar to $d\mathcal{W}$, which goes to $\mathcal{W}$ in the limit. Metaphorically, X thus "imitates" the winner $\mathcal{W}$.

Practically, a model-driven mutation termed *social mutation* is implemented as follows. Given the number M of bits to flip (see section 5), one selects these M bits by tournament among the bits. For each one of these M bits, T bits $i_1, \dots, i_T$ are uniformly selected in $1..N$, and the bit i such that it maximizes $|X_i - \mathcal{W}_i|$, is flipped. This way, the offspring actually reduces its distance to the winner.

This mechanism can be compared to the majority crossover of Syswerda, setting the value of the offspring to the majoritary value of the bit in the population when the two parents differ on that bit [27]. The difference between majority crossover and social mutation is twofold: the majority crossover takes into account *all* individuals in the *current* population; social mutation takes into

account the *best* individuals in all *past* populations.

Social mutation is easily refined to also account for the loser. For instance, according to $d\mathcal{W}$, it might be a good idea to mutate bit 5; but $d\mathcal{L}$ suggests that ($bit_5 = 1$) is not a factor of high fitness. This leads to select the bits to mutate, so that the offspring "imitates" $\mathcal{W}$ and "rejects" $\mathcal{L}$. Practically, one only modifies the tournament criterion: the winner of the tournament is the bit maximizing $|X_i - \mathcal{W}_i| - |X_i - \mathcal{L}_i|$.

Note that the relaxation factors α_w and α_l respectively associated to the winner and the loser could be different, though they are set to the same value (10^{-2}) throughout this paper. In this case, $\mathcal{L}$ changes faster than $\mathcal{W}$.

3.2 Social strategies

However, there is no reason why an individual could only imitate the winner and reject the loser. A straightforward generalization is to define a pair (δ_W, δ_L) in

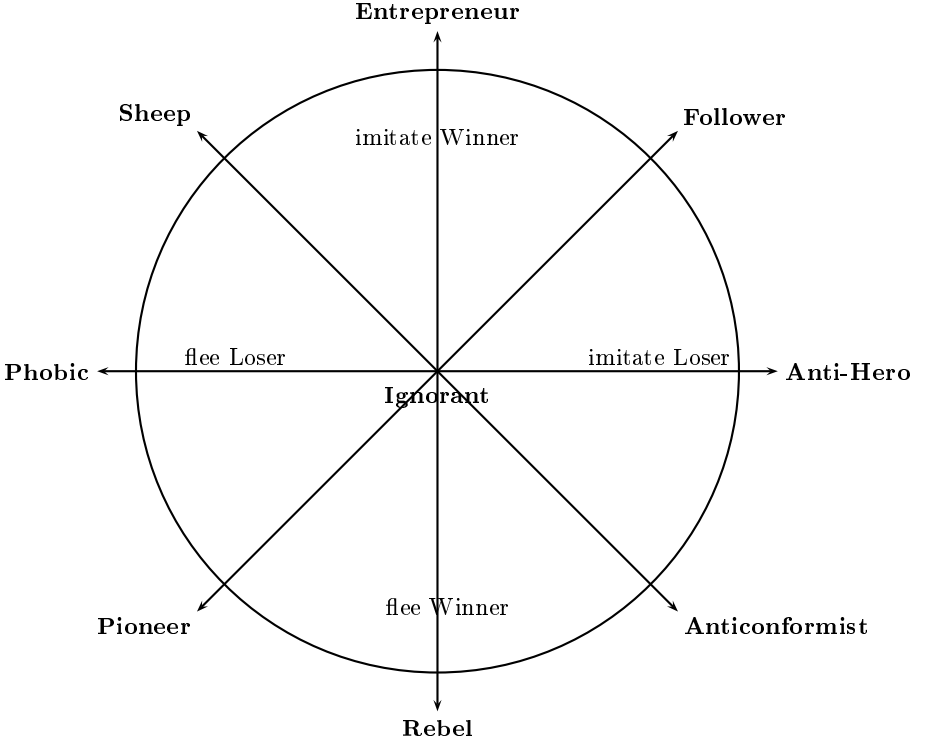


Fig. 1. Social Strategies

$\mathbb{R}^2$, and to select the bits to mutate as those maximizing

$$\delta_W |X_i - \mathcal{W}_i| + \delta_L |X_i - \mathcal{L}_i|$$

One sees that X imitates model $\mathcal{M}$ ($= \mathcal{W}$ or $\mathcal{L}$) if $\delta_M > 0$, rejects $\mathcal{M}$ if $\delta_M < 0$, and ignores $\mathcal{M}$ if $\delta_M = 0$. Social mutation finally gets parameterized by the pair (δ_W, δ_L) , termed *social strategy*. Some of these strategies have been given names for the sake of convenience; obviously, other systems of metaphors could have been imagined. We distinguish mainly:

- The *entrepreneur*, that imitates the winner and ignores the loser;
- The *sheep*, that imitates the winner and rejects the loser;
- The *phobic*, that rejects the loser and ignores the winner;
- The *pioneer*, that rejects both the winner and the loser;
- The *rebel*, that rejects the winner and ignores the loser;
- The *ignorant*, that ignores both the loser and the winner.

One notices that social mutation is unchanged if δ_W and δ_L are multiplied by a positive coefficient. With no loss of information, social strategies are thus represented by angles ($\mathbb{R}^2$ being projected onto the unit circle). This angle sets the direction of evolution of the individuals, in the changing system of coordinates given by the winner and the loser. Figure 1 shows the directions corresponding to the main social strategies, with angle 0 corresponding to imitating the loser, and angle $\pi/2$ to imitating the winner.

4 Limitations

As expected, not all social strategies are relevant and some turn out to be misleading. A clear such case is when the winner is close to a local optimum; imitating the winner just drives the population to the local optimum, which does not allow for modifying the winner; a deadlock of mimetic evolution thus occurs.

In a general way, the population and the models depend on each other: at each generation, the best (worst) individuals in the population are used to maintain the models; and these models are used to evolve the population. This dynamics can produce several kinds of deadlock.

If the population does not make any progress, so does the winner; hence any strategy based on solely imitating the winner will fail, that is, fail to re-orient the search. In the meanwhile, the loser will probably fluctuate; either imitating or rejecting the loser could help sidestep this trap.

The loser can also mislead the population. If the loser is close to the optimum, rejecting the loser will deceptively lead the population in bad regions; the loser will then change, providing more reliable beacons and allowing the population to resume its pilgrimage toward the optimum. This may produce an oscillation, satellizing the population around the optimum. Concretely, let us consider a population climbing a hill. As long as the loser is on the same side of the hill as the population, fleeing the loser will make the population duly climb the hill. But

when individuals are equally distributed on both sides of the hill, the loser gets close to the optimum, and now exerts a repelling influence on the population. This perpetuates until the symmetry is broken. Still worse, the loser may be on the other side of the hill than the majority of the population (if the relaxation factor is too large, for instance), and make the population go down the hill...

The distribution of individuals may also cause any strategy to be deceptive. Consider an even distribution of the population between the two schemas, and assume that model $\mathcal{M}$ reflects this distribution (Table 2).

Schema 1	0	1	*	*	*	...	*
Schema 2	1	0	*	*	*	...	*
$\mathcal{M}$	0.5	0.5	*	*	*	...	*

Table 2: Stable distribution of the population

Any individual X most differs from $\mathcal{M}$ by bits 1 and 2 ($|X_i - \mathcal{M}_i| = .5, i = 1, 2$). Imitating $\mathcal{M}$ means reducing the differences between X and $\mathcal{M}$, hence flipping preferably bits 1 and 2 (if the number of bits to mutate is greater than 2 — this will be discussed in the next section). But mutating bits 1 and 2 would perpetuate the distribution of the population between schemas 1 and 2.

Inversely, if the strategy is to reject $\mathcal{M}$, one wants to preserve the differences between X and $\mathcal{M}$, hence bits 1 and 2 will never be modified; and, everything being equal, this would preserve the distribution of the population too.

To sum up, observing a single model may perpetuate the traps visited by the model or the population. This is confirmed by the fact that PBIL enriches the computation of the winner with random perturbations [4], or computes the winner from the two best and the worst individuals [3]. We experimented similar heuristics based on gaussian perturbations of the models.

However, it turns out that guiding evolution according to *two* models is much more robust than with only one model: the influence of each model somehow moderates the influence of the other one, and makes it less harmful. Still, there certainly exists cases where the two combined models deliver deceptive indications. Further research will focus on the deceptivity of social strategies.

5 Social Pressure

Binary mutation is traditionnally parameterized by the mutation rate, usually very low, setting the average number of bits to mutate in the population. And the selection of the mutated bits is done at random.

In opposition, social mutation primarily concentrates on ordering the bits to mutate depending on the individual, via defining its desired direction of evolution (section 3.2). But how far the individual should go in this direction, i.e. the number M of bits to mutate, is still to determine. This parameter, termed *social pressure*, controls the balance of evolution between exploitation and exploration, respectively achieved for low and high values of M . In any case, the social

pressure must correspond to a much higher mutation rate than for standard mutation, as social mutation is meant to replace both mutation and crossover.

A previous work investigated two heuristics for the auto-adjustment of M in the framework of evolution by inhibitions [25]. The first heuristic is inspired from Davis [5], and proceeds by rewarding the values of M leading to fitness increases. The second one is inspired from self-adaptation [23, 1]: the number M is encoded within each individual, and evolution supposedly optimizes M as well as the genotypic information of the individual.

Unfortunately, none of our attempts succeeded in determining relevant global or local values of M ; rather, all heuristics rapidly lead to setting $M = 1$. Mimetic evolution thereafter behaves as a standard hill-climber, and soon gets trapped in a local optimum.

In retrospect, reward-based adjustment tends to be risk-adverse, and favor options that bring small frequent improvements, over options that bring rare, though large, improvements; this explains why $M = 1$ is preferred.

On the other hand, the self-adaptation of M fails because the strong causality principle [21] is violated: finding the optimal M amounts to finding an optimal *discrete* value in a very restricted range (say $[2..N/10]$); no wonder that the convergence results of evolution strategies [23, 1] do not apply.

We therefore used fixed schedules to determine M_t at generation t . The simplest possibility is to set M_t to a fixed, user-supplied value M_0 . A more sophisticated possibility is to decrease M_t from an initial, user-supplied, value M_0 . We used a decreasing hyperbolic schedule borrowed from [2]:

$$M_t = \frac{1}{\frac{1}{M_0} + t \cdot \frac{1 - \frac{1}{M_0}}{T - 1}} \quad (1)$$

where T is the maximum number of generations and t denotes the current generation. Social mutation then mutates exactly the integer part of M_t bits in each individual⁴.

It now remains to set the initial (or fixed) value of M_t , M_0 . We used off-line adjustments loosely inspired from Grefenstette’s Virtual GA [10]. More precisely, a (1+10) binary ES is run for a few generations, for each considered value of M_0 (2..5 in the constant schedule, and {45, 90, 450} in the decreasing schedule), and one chooses the value of M_0 leading to the best performance.

6 Experimental Validation

This section discusses the experimental results obtained by mimetic evolution on large-sized problems, and focuses on the role of social strategy.

⁴ Another possibility, not investigated here, is to mutate on average M_t bits on the population [25].

6.1 Problems

The experiments consider some standard test functions: the OneMax, the Twin-Peaks and the (deceptive) Ugly⁵ functions, taken on $\{0, 1\}^{900}$.

Other functions are taken from [3], and respectively correspond to the binary coding and the Gray coding of the continuous functions on $[-2, 56, 2.56]^{100}$ below:

$$\begin{aligned}
 y_1 &= x_1 \\
 y_i &= x_i + y_{i-1}, \quad i \geq 2 \\
 F_1 &= \frac{100}{10^{-5} + \sum_i |y_i|} \\
 F_3 &= \frac{100}{10^{-5} + \sum_i |.024 * (i + 1) - x_i|}
 \end{aligned}$$

In the latter case, each continuous interval $[-2, 56, 2.56[$ is mapped onto $\{0, 1\}^9$; individuals thus belong to $\{0, 1\}^{900}$.

The importance of the coding is witnessed, if necessary, by the fact that F_3 only reaches a maximum of 416,63 in its binary version, whereas the continuous optimum is 10^7 ; this is due to the fact that the continuous optimum ($X_i = .024 * (i + 1)$) does not belong to the discrete space considered.

6.2 Experimental setting

The evolution scheme is a (10+50)-ES: 10 parents produce 50 offspring and the 10 best individuals among parents plus offspring are retained in the next population. A run is allowed 200,000 evaluations; all results are averaged on 10 independent runs.

The winner and the loser respectively memorize the best and the worst offspring of the current generation. The relaxation factors α_w and α_l are both set to .01. The tournament size is set to 20.

Several reference algorithms have been experimented on functions F_1 and F_3 : two variants of GAs (GA1 and GA2), two variants of Hill-climbers (HC1 and HC2) [3] and two variants of evolution strategies (ES1 and ES2) [25]. These algorithms served as reference for PBIL and evolution by inhibitions (INH). Another reference algorithm is given by mimetic evolution following an ignorant social strategy (an offspring is generated by randomly mutating 3 bits).

Function	HC1	HC2	AG1	AG2	AES	TES	PBIL	INH	IGNOR
F1 Binary	1.04	1.01	1.96	1.72	2.37	1.87	2.12	2.99	2.98
F3 Gray	416.65	416.65	28.35	210.37	380.3	416.65	366.77	246.23	385.90

Table 3. Reference results on F_1 and F_3 .

More results, and the detailed description of the reference algorithms, are found in [25]. In this paper, we focus on the influence of the social strategy

⁵ Defined as 300 concatenations of the elementary deceptive function U defined on $\{0, 1\}^3$ as follows: $U(111) = 3$; $U(0XX) = 2$; otherwise $U = 0$.

parameter, and in particular, the social pressure (nb of bits mutated) is fixed to 3.

6.3 The influence of social strategies

A new visualization format is proposed on Figure 2, in order to compare the results obtained for different strategies on a given problem. As social strategies can be represented as angles, results are plotted in polar coordinates: point (ρ, θ) illustrates the results obtained for strategy θ , with ρ being the best average fitness obtained for this strategy. Two curves are plotted by joining all points (ρ, θ) : the internal curve gives the results obtained for 50,000 evaluations, and the external one gives the results obtained for 200,000 evaluations. Both curves overlap (e.g. for OneMax and Ugly) when evolution reaches the optimum in about 50,000 evaluations or less.

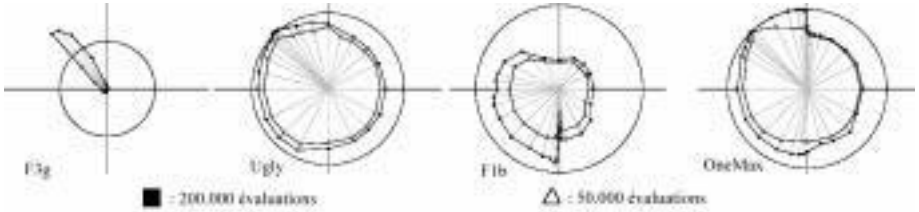


Fig. 2. *Summary results of social strategies*

The graphics obtained for functions TwinPeaks and F_{1g} , quite similar to respectively those of OneMax and F_{1g} , are omitted given the space limitations.

Note that the shape of the curve obtained for 50,000 evaluations is very close to that obtained for 200,000 evaluations; in other words, the social strategy most adapted to a problem does not seem to vary along evolution.

In all cases, strategies in the left half space are significantly better than those in the right half space: for all functions considered, the recommended behavior with respect to the loser is the flight. In opposition, the recommended behavior with respect to the winner depends on the function: flight is strongly recommended in the case of F_1 , where the Pioneer strategy is the best. Inversely, imitation is strongly recommended in the case of F_3 , OneMax and Ugly, where the Sheep strategy is the best. This is particularly true for F_3 , where the Sheep is significantly better than other strategies: the respective amounts of imitation of the winner and avoidance of the loser must be carefully adjusted.

7 Conclusion

Our working hypothesis is that the memory of evolution can provide worth information to guide the further steps of evolution.

Instead of memorizing only the best [4] or the worst [25] individuals, mimetic evolution memorizes all striking past individuals, the best and the worst ones. A population can thereby access two models, featured as the Winner and the Loser.

These models define a changing system of coordinates in the search space; an individual can thus be given a direction of evolution, or social strategy, expressed with respect to both models. And mimetic evolution proceeds by moving each individual in this direction. This paper has only considered the case of a single social strategy, fixed for all individuals, and focus on the influence of the strategy on the dynamics of evolution.

Evolution by inhibitions is a special case, driven by the loser model only, of mimetic evolution. As mimetic evolution involves both loser and winner models, the possible strategies get richer. As could be expected, 2-model-driven strategies appear more robust than 1-model-driven ones. Indeed, the more models are observed, the less predictable and deterministic the behavior of the population gets; and the less likely evolution goes in a deadlock.

Experimental results show that for each test function there exists a range of optimal social strategies (close to the Sheep in most cases; and to the Pioneer/Rebel in the others). And on most functions, these optimal strategies significantly outperform the "ignorant" strategy, which serves as reference and ignores both models.

In any case, it appears that memory contains relevant information, as the way we use it fortunately often allows for speeding up evolution. Obviously, other and more clever exploitations of this information remain to be invented. On the other hand, the use of the memory is tightly connected to its content: which knowledge exactly should be acquired during evolution? Currently the models reflect, or "capitalize", the individuals; another level of memory would capitalize the dynamics of the population as a whole, i.e. the sequence over all generations of the social strategy leading from the best parents to the best offspring.

A further perspective of research is to self-adapt the social strategy of an individual, by enhancing the individual with its proper strategy parameters δ_W and δ_L . This way, evolution could actually benefit from a mixture of different social strategies, dispatched within the population⁶.

Another perspective is to extend mimetic evolution to continuous search spaces. Computing the winner and the loser from continuous individuals is straightforward; but the question of how to use them is still more open, than in the binary case.

⁶ Incidentally, this would reflect more truly the evolution of societies. But indeed social modelling is far beyond the scope of this work.

A last perspective is to see to what extent the difficulty of the fitness landscape for a standard GA is correlated to the optimal social strategy for this landscape (among which the ignorant strategy). Ideally, the optimal strategy would allow one to compare diverse components of evolution, in the line of the Fitness Distance Correlation criterion [15]. The advantage of such criteria is to provide a priori estimates of the adequacy of e.g., procedures of initialization of the population [16], or evolution operators [17].

References

1. T. Bäck. *Evolutionary Algorithms in theory and practice*. New-York:Oxford University Press, 1995.
2. T. Bäck and M. Schütz. Intelligent mutation rate control in canonical GAs. In Z. W. Ras and M. Michalewicz, editors, *Foundation of Intelligent Systems 9th International Symposium, ISMIS '96*, pages 158–167. Springer Verlag, 1996.
3. S. Baluja. An empirical comparizon of seven iterative and evolutionary function optimization heuristics. Technical Report CMU-CS-95-193, Carnegie Mellon University, 1995.
4. S. Baluja and R. Caruana. Removing the genetics from the standard genetic algorithms. In A. Prieditis and S. Russel, editors, *Proceedings of ICML95*, pages 38–46. Morgan Kaufmann, 1995.
5. L. Davis. Adapting operator probabilities in genetic algorithms. In J. D. Schaffer, editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 61–69. Morgan Kaufmann, 1989.
6. A.E. Eiben and Z. Ruttkay. Self-adaptivity for constraint satisfaction: Learning penalty functions. In T. Fukuda, editor, *Proceedings of the Third IEEE International Conference on Evolutionary Computation*, pages 258–261. IEEE Service Center, 1996.
7. I.K. Evans. Enhancing recombination with the complementary surrogate genetic algorithm. In T. Bäck, Z. Michalewicz, and X. Yao, editors, *Proceedings of the Fourth IEEE International Conference on Evolutionary Computation*, pages 97–102. IEEE Press, 1997.
8. D.B. Fogel and L.C. Stayton. On the effectiveness of crossover in simulated evolutionary optimization. *BioSystems*, 32:171–182, 1994.
9. D. E. Goldberg. *Genetic algorithms in search, optimization and machine learning*. Addison Wesley, 1989.
10. J. J. Grefenstette. Virtual genetic algorithms: First results. Technical Report AIC-95-013, Navy Center for Applied Research in Artificial Intelligence, 1995.
11. N. Hansen, A. Ostermeier, and A. Gawelczyk. On the adaptation of arbitrary normal mutation distributions in evolution strategies: The generating set adaptation. In L. J. Eshelman, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 57–64. Morgan Kaufmann, 1995.
12. J. H. Holland. *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor, 1975.
13. J. Horn and D.E. Goldberg. Genetic algorithms difficulty and the modality of fitness landscapes. In L. D. Whitley and M. D. Vose, editors, *Foundations of Genetic Algorithms 3*, pages 243–269. Morgan Kaufmann, 1995.

14. T. Jones. Crossover, macromutation and population-based search. In L. J. Eshelman, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 73–80. Morgan Kaufmann, 1995.
15. T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In L. J. Eshelman, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 184–192. Morgan Kaufmann, 1995.
16. L. Kallel and M. Schoenauer. Alternative random initialization in genetic algorithms. In Th. Bäck, editor, *Proceedings of the 7th International Conference on Genetic Algorithms*. Morgan Kaufmann, 1997. To appear.
17. L. Kallel and M. Schoenauer. A priori predictions of operator efficiency. In *Artificial Evolution'97*. CMAP - Ecole Polytechnique, October 1997.
18. J. R. Koza. *Genetic Programming: On the Programming of Computers by means of Natural Evolution*. MIT Press, Massachusetts, 1992.
19. P. Collard and J.P. Aurand. Dual ga: An efficient genetic algorithm. In *Proceedings of European Conference on Artificial Intelligence*, pages 487–491. Amsterdam, Wiley and sons, August 1994.
20. C. Ravisé and M. Sebag. An advanced evolution should not repeat its past errors. In L. Saitta, editor, *Proceedings of the 13th International Conference on Machine Learning*, pages 400–408, 1996.
21. I. Rechenberg. *Evolutionstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*. Fromman-Holzboog Verlag, Stuttgart, 1973.
22. R.G. Reynolds. An introduction to cultural algorithms. In *Proceedings of the 3rd Annual Conference on Evolutionary Programming*, pages 131–139. World Scientific, 1994.
23. H.-P. Schwefel. *Numerical Optimization of Computer Models*. John Wiley & Sons, New-York, 1981. 1995 – 2nd edition.
24. M. Sebag and M. Schoenauer. Mutation by imitation in boolean evolution strategies. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Proceedings of the 4th Conference on Parallel Problems Solving from Nature*, pages 356–365. Springer-Verlag, LNCS 1141, 1996.
25. M. Sebag, M. Schoenauer, and C. Ravisé. Toward civilized evolution: Developing inhibitions. In Th. Bäck, editor, *Proceedings of the 7th International Conference on Genetic Algorithms*. Morgan Kaufmann, 1997.
26. W. M. Spears. Adapting crossover in a genetic algorithm. In R. K. Belew and L. B. Booker, editors, *Proceedings of the 4th International Conference on Genetic Algorithms*. Morgan Kaufmann, 1991.
27. G. Syswerda. A study of reproduction in generational and steady state genetic algorithm. In G. J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 94–101. Morgan Kaufmann, 1991.

Adaptive penalties for evolutionary graph coloring

A.E. Eiben, J.K. van der Hauw

Leiden University, The Netherlands, {gusz,jvdhauw}@wi.leidenuniv.nl

Abstract. In this paper we consider a problem independent constraint handling mechanism, Stepwise Adaptation of Weights (SAW) and show its working on graph coloring problems. SAW-ing technically belongs to the penalty function based approaches and amounts to modifying the penalty function during the search. We show that it has a twofold benefit. First, it proves to be rather insensitive to its technical parameters, thereby providing a general, problem independent way to handle constrained problems. Second, it leads to superior EA performance. In an extensive series of comparative experiments we show that the SAW-ing EA outperforms a powerful graph coloring heuristic algorithm, DSatur, on the hardest graph instances and has a linear scale-up behaviour.

1 Introduction

In this paper we consider an adaptive mechanism for constraint handling (called SAW-ing) on graph 3-coloring problems. In [13] SAW-ing was applied to 3SAT problems and the resulting EA turned out to be superior to WGSAT, the best heuristics for 3SAT problems known at the moment. It is interesting to note that optimizing the population size and the operators in the the SAW-ing EA for 3SAT resulted in an algorithm that was very similar to WGSAT itself. This EA was, however, obtained independently from WGSAT, starting with a full blown EA with a large population and using crossover. It was an extensive test series that showed that a $(1, \lambda)$ selection scheme using mutation only is superior. Despite the different origin of the two compared methods, their similarity in the technical sense might suggest that the superior performance of the SAW-ing EA is just a coincidence: it holds for 3SAT, but not for other constraint satisfaction problems. In this paper we show that this is not the case.

Graph coloring falls in the category of grouping problems. Several authors [14, 15, 24] have considered grouping problems and gave arguments that they cannot be successfully solved by usual genetic algorithms, e.g. using traditional representations and the corresponding standard operators, and proposed special representation and crossovers for such problems. In this paper we show the viability of another approach to solve a grouping problem based on an adaptively changing fitness function in an EA using a common representation and standard operators.

We restrict our investigation to graph 3-coloring problems that are pure constraint satisfaction problems, unlike the constrained optimization version as

studied by for instance Davis [7]. To evaluate the performance of our EA we also run a powerful traditional graph coloring algorithm on the same problems. The final comparison shows that the SAW-ing EA is superior to the heuristic method on the hardest problem instances.

The rest of the paper is organized as follows. In Section 2 we specify the problems we study and give a brief overview on traditional graph coloring algorithms. We select one of them, DSatur, as a competitor we compare the performance of EAs with. Section 3 summarizes our results on graph coloring obtained with a EA, and compares this EA with DSatur, as well as a hybridized EA+DSatur system. Thereafter, in Section 4 we present an adaptive mechanism that is changing the penalty function, thus the fitness landscape, during a EA run. We show that the adaptive EA highly outperforms other tested EA variants, including the hybrid system. Finally, in Section 5 we compare the adaptive EA with DSatur and conclude that the EA is superior with respect to performance on hard problem instances as well as concerning scale-up properties.

2 Graph Coloring

In a graph 3-coloring problem the task is to color each vertex $v \in V$ of a given undirected graph $G = (V, E)$ with one of three colors from $\{1, 2, 3\}$ so that no two vertices connected by an edge $e \in E$ are colored with the same color. This problem in general is NP-complete [16] making it theoretically interesting; in the meanwhile there are many specific applications like register allocation [3], timetabling [25], scheduling and printed circuit testing [17]. In the literature there are not many benchmark 3-colorable graphs and therefore we create graphs to be tested with the graph generator written by Culberson.¹ Creating 3-colorable test graphs happens by first pre-partitioning the vertices in three sets (3 colors) and then drawing edges randomly by a certain probability p , the edge density. We generated *equi-partite 3-colorable graphs* where the three color sets are as nearly equal in size as possible, as well as *flat 3-colorable graphs*, where also the variation in degree for the vertices is kept to a minimum. Determining the chromatic number of these two types of graphs is very difficult, because there is no information a (heuristic) coloring algorithm could rely on, [6]. Our tests showed that they are also tough for 3-coloring. Thorough this paper we will denote graph instances by, for example $G_{eq, n=500, p=0.10, s=1}$, standing for an equi-partite 3-colorable graph with 500 vertices, edge probability 10% and seed 1 for the random generator.

Cheeseman *et al.* [4] found that NP-complete problems have an ‘order parameter’ and that the hard problems occur at a critical value or *phase transition* of such a parameter. For graph coloring, this order parameter is the edge probability or edge connectivity p . Theoretical estimations of Clearwater and Hogg [5] on the location of the phase transition, supported by empirical validation, improved the estimates in [4] and indicate that the hardest graphs are those with

¹ Source code in C is available via <ftp://ftp.cs.ualberta.ca/pub/joe/>
GraphGenerator/generate.tar.gz

an edge connectivity around $7/n - 8/n$. Our experiments confirmed these values. We will use these values in the present investigation and study large graphs with up to 1500 vertices.

To compare the performance of our EAs with traditional graph coloring algorithms we have looked for a strong competitor. There are many (heuristic) graph coloring algorithms in the literature, for instance an $O(n^{0.4})$ -approximation algorithm by Blum [1], the simple Greedy algorithm [20], DSatur from Br  l  z [2], Iterated Greedy (IG) from Culberson and Luo [6], XRLF from Johnson *et al.* [19]. We have chosen DSatur as competitor for its high performance. DSatur uses a heuristic to dynamically change the ordering of the nodes and then applies the greedy method to color the nodes:

- A node with highest saturation degree (= number of differently colored neighbors) is chosen and given the smallest color that is still possible.
- In case of a tie, the node with highest degree (= number of neighbors that are still in the uncolored subgraph) is chosen.
- In case of a tie a random node is chosen.

Because of the random tie breaking, DSatur is a stochastic algorithm and, just like for the EA, results of several runs need to be averaged to obtain useful comparisons. For the present investigation we implemented the backtracking version of Turner [23], which backtracks to the lastly evaluated node that still has available colors to try.

3 The Evolutionary Algorithm

We implemented different steady-state algorithms using worst fitness deletion, with two different representations and tested different operators and population sizes for their performance. These tests were intended to check the hypotheses in [14, 15, 24] on the disadvantageous effects of crossover in standard representations, as well as to find a good setup for our algorithm. For a full overview of the test results see [12], here we only present the most interesting findings. It turned out that mixing information of different individuals by crossover is not as bad as is generally assumed. Using integer representation, each gene in the chromosome belongs to one node and can take three different values as alleles (with the obvious semantics). Applying heavy mixing by multi-point crossovers and multi-parent crossovers, [8], improves the performance. In Figure 1 we give an illustration for the graph $G_{eq, n=200, p=0.08, s=5}$, depicting the Average Number of Evaluations to a Solution (AES) as a function of the number of parents in diagonal crossover, [8], respectively as a function of the number of crossover points in m-point crossover. The results are obtained by averaging the outcomes of 100 independent runs.

Integer representation, however, turned out to be inferior to order-based representation. Using order-based representation, each chromosome is a permutation of nodes, and a decoder is needed to create a coloring from a permutation.

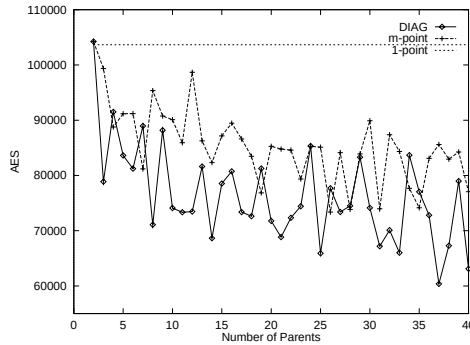


Fig. 1. Effect of more parents and more crossover points on the Average Number of Evaluations to a Solution (AES) for $G_{eq,n=200,p=0.08,s=5}$

We used a simple coloring decoder, encountering the nodes in the order they occur in a certain chromosome and giving each node the smallest² possible color. If each of the three colors leads to constraint violation, the node is left uncolored, and the fitness of a chromosome (to be minimized) is the number of uncolored nodes. After performing numerous tests the best option turned out to be an order-based GA without crossover, using mutation only with population size 1 in a (1+1) preservative selection scheme. Because of the lack of crossover we call this algorithm an EA (evolutionary algorithm), rather than a GA (genetic algorithm). This EA forms the basis of our further investigation: we will try to improve it by hybridization and by adding the SAW-ing mechanism.

Let us note that DSatur also uses an ordering of nodes as the basis to construct a coloring (in particular, it uses an ordering based on the degrees to break ties). It is thus a natural idea to use an EA to find better orderings than those used by DSatur. Technically, DSatur would still use the (dynamically found) saturation degree to select nodes and to color them with the first available color, but now it would break ties when two nodes have equal saturation degree by using a permutation (an individual in the EA) for ordering the nodes. From an EA point of view we can see DSatur as a new decoder for the EA which creates a coloring when fed with a permutation. We also tested this hybridized EA+DSatur system where the fitness value of a given permutation is the same as for the greedy decoder. The comparison between the order-based EA, DSatur with backtracking and the hybrid system is shown in the first three rows of Table 1 in Section 4.1 for the graph $G_{eq,n=1000,p=0.010}$. These results are based on four random seeds for generating graph instances, for each instance 25 independent runs were executed with $T_{max} = 300.000$ as the maximum number of evaluations for every algorithm. In the table column SR stands for Success Rate, i.e. the % of cases where the graph could be colored, the column AES is again

² Colors are denoted by integers 1,2,3

the average number of evaluations to a solution.

Based on Table 1 we can make a number of interesting observations. First, the results show even the best EA cannot compete with the DSatur, but hybridizing the two systems leads to a coloring algorithm that outperforms both of its components. Second, the performance of the algorithms is highly dependent on the random seeds used for generating the graphs. This means that they are not really powerful, they cannot color graphs with a high certainty.

4 The SAW-ing Evolutionary Algorithm

In this section we extend the EA used in the previous tests. Let us first have a look on the applied penalty function that concentrates on the nodes (variables to be instantiated), rather than on the edges (constraints to be satisfied). Formally, the function f to be minimized is defined as:

$$f(x) = \sum_{i=1}^n w_i \cdot \chi(x, i) \quad (1)$$

where w_i is the local penalty (or weight) assigned to node x_i and

$$\chi(x, i) = \begin{cases} 1 & \text{if node } x_i \text{ is left uncolored} \\ 0 & \text{otherwise} \end{cases}$$

In the previous section we were simply counting the uncolored nodes, thus used $w_i \equiv 1$. This weight distribution does not distinguish between nodes, although it is not reasonable to assume that all nodes are just as hard to color. Giving hard nodes a high weight is a very natural idea, since this gives the EA a high reward when satisfying them, thus the EA will ‘concentrate’ on these nodes. A major obstacle in doing this is, of course, that the user does not know which nodes are hard and which ones are easy. Heuristic estimates of hardness can circumvent this problem, but still, the hardness of a node is most certainly also depending on the problem solver, i.e. coloring algorithm, being applied, that is a node that is hard for one problem solver may be easy for another one. Furthermore, being hard may be also context dependent, i.e. may depend on the information the problem solver collected in a certain phase of the search. This means that even for one specific problem solver, a particular setting of weights may become inappropriate as the search proceeds. A simple answer to these drawbacks is embodied by the following mechanism.

4.1 Stepwise Adaptation of Weights

Motivated by the above reasons we decided to leave the decisions of the hardness of different nodes to the EA itself, moreover we allow the EA to revise its decisions during the search. Technically this means that we apply a varying fitness function that is repeatedly modified, based on feedback concerning the progress of the search process. Similar mechanisms have been proposed earlier in

another context by, for instance, Moris [21] and Selman and Kautz [22]. In evolutionary computation varying parameters can be divided into three classes [18], dynamic, adaptive and self-adaptive parameter control. Our approach falls in the adaptive category. The general idea is now implemented by repeatedly checking which nodes in the best individual³ violate constraints and raising the penalty w_i belonging to these nodes. Depending on when the weights are updated we can distinguish an off-line (after the run, used in the next run) and an on-line (during the run) version of this technique. In [9] and [10] the off-line version was applied, here we will use the on-line version. In particular, the EA starts with a standard setting of $w_i \equiv 1$ for each node. After each T_p fitness evaluations the best individual in the population is checked and the weights belonging to its uncolored nodes are increased by Δw , i.e. setting $w_i = w_i + \Delta w$. This mechanism introduces two new parameters, T_p and Δw and it is important to test whether the EA performance is sensitive for the values of these parameters. Limited by space requirements here we can only give an illustration. Figure 2 shows the success rates of an asexual (only SAWP mutation) and a sexual EA (OX2 crossover and SWAP mutation) for different values of Δw and T_p . These

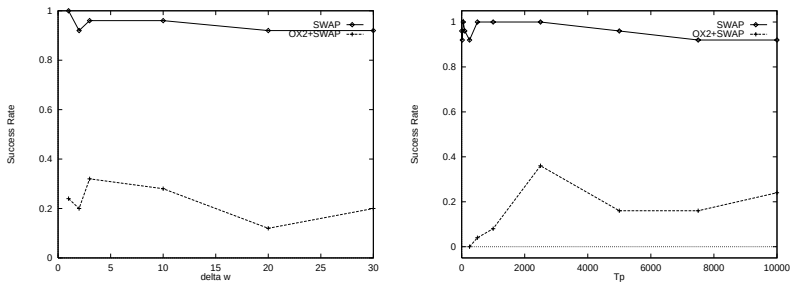


Fig. 2. Influence of Δw (left) and T_p (right) on SR. Tested with $T_{max} = 300000$ on $G_{eq,n=1000,p=0.01,s=5}$, $T_p = 250$ is used for Δw , $\Delta = 1$ is used for T_p .

tests reconfirm that using mutation only is better than using crossover and mutation, furthermore they indicate that the parameters of the SAW mechanism have only neglectable influence on the EA performance. For no specific reason we use $\Delta w = 1$ and $T_p = 250$ in the sequel.

Obviously, an EA using the SAW mechanism searches on an adaptively changing fitness landscape. It is interesting to see how the fitness (actually the penalty expressing the error belonging to a certain chromosome) changes under the SAW regime. Figure 3 shows a typical run.

As we see on the plot, the error is growing up to a certain point and then

³ More than one individuals could be also monitored, but preliminary tests did not indicate advantage of this option.

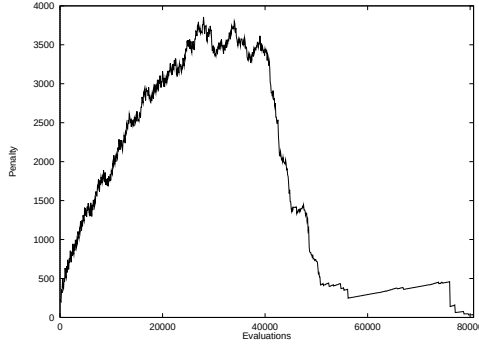


Fig. 3. Penalty curve of the best chromosome in an asexual SAW-ing EA, tested on $G_{eq,n=1000,p=0.01,s=5}$ with $T_{max} = 300000$.

it quickly decreases, finally hitting the zero-line, indicating that a solution has been found. In high resolution (not presented here) the curve shows decreasing error rates in periods of fixed T_p , followed by a sudden increase when T_p is reset, that is, we see the shape of a **saw**!

4.2 Performance of the SAW-ing EA

In Table 1 we present the performance results of the SAW-ing EA and the earlier tested EA versions. Comparing the results we see that the performance of the EA highly increases by the usage of SAW-ing, the success rates become very high and the algorithm is twice as fast as the other ones. Besides, the SAW-ing EA performs well independently from the random seeds, i.e., it is very robust.

	$s = 0$		$s = 1$		$s = 2$		$s = 3$		all 4 seeds	
	SR	AES	SR	AES	SR	AES	SR	AES	SR	AES
DSatur	0.08	125081	0.00	300000	0.00	300000	0.80	155052	0.22	220033
EA	0.24	239242	0.00	300000	0.00	300000	0.12	205643	0.09	261221
EA+DSatur	0.44	192434	0.24	198748	0.00	300000	0.64	114232	0.33	201354
EA+SAW	0.96	76479	0.88	118580	0.92	168060	0.92	89277	0.92	113099

Table 1. Comparing DSatur with backtracking, the EA, the hybrid EA and the SAW-ing EA for $n = 1000$ and $p = 0.010$ with different random seeds.

5 The SAW-ing EA vs. DSatur

The experiments reported in the previous sections clearly indicate the superiority of the SAW-ing EA with respect to other EAs. The real challenge to our weight adaptation mechanism is, however, a comparison with a powerful heuristic graph coloring technique. We have performed an extensive comparison between the SAW-ing EA and DSatur on three different types of graphs (arbitrary 3-colorable, equi-partite 3-colorable and flat 3-colorable) for different sizes ($n = 200, 500, 1000$) and a range of edge connectivity values comparing SR as well as AES. Space limitations prevent us from presenting all figures; the inter-

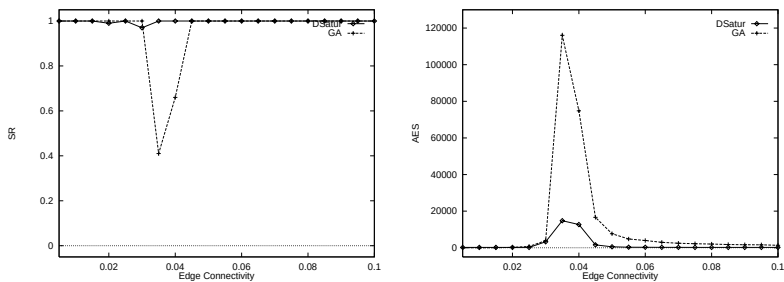


Fig. 4. Comparison of SR (left) and AES (right) for $n = 200$.

ested reader is again referred to [12]. Here we give an illustration on the hardest case: flat 3-colorable graphs. Comparative curves of success rates and the number of evaluations to a solution are given in Figure 4 and Figure 5 for $n = 200$ and $n = 1000$ respectively. The phase transition is clearly visible on each figure: the

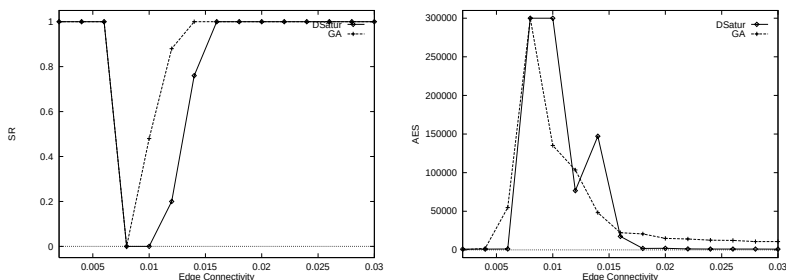


Fig. 5. Comparison of SR (left) and AES (right) for $n = 1000$.

performance of both algorithms drops at certain values of the edge connectivity p . On small graphs ($n = 200$), the deterioration of performance is smaller for DSatur, while on large graphs ($n = 1000$) the valley in the SR curve and the peak in the AES curve are narrower for the EA, showing its superiority to DSatur on these problems. The reason for these differences could be that on the small instances DSatur with backtracking is able to get out of local optima and find solutions, while this is not possible anymore for large graphs where the search space becomes too big. On the other two graph topologies (arbitrary 3-colorable and equi-partite 3-colorable) the results are similar.

An additional good property of the SAW-ing EA is that it can take more advantage of extra time given for search. For instance, on $G_{eq, n=1000, p=0.008, s=5}$ both algorithms fail (SR=0.00) when 300.000 evaluations are allowed. If we increase the total number of evaluations from 300.000 to 1.000.000 DSatur still has SR=0.00, while the performance of the EA raises from SR=0.00 to SR=0.44 (AES=407283). This shows that the EA is able to benefit from more time given, where the extra time for backtracking is not enough to get out of the local optima.

Finally, let us consider the issue of scalability, that is the question of how the performance of an algorithm changes if the problem size grows. Experiments on the hardest instances at the phase transition for $p = 8/n$ show again that DSatur is not able to find solutions on large problems. Since this leads to SR = 0 and undefined AES (or $AES = T_{max}$), we rather perform the comparison on easier problem instances belonging to $p = 10/n$. The results are given in Figure 6. These figures clearly show that the SAW-ing EA outperforms DSatur. Moreover, the comparison of the AES curves suggests a linear time complexity of the EA. Taking the scale-up curves into consideration also eliminates a possible drawback of using AES for comparing two different algorithms. Recall, that AES is the average number of search steps to a solution. For an EA a search step is the creation and evaluation of a new individual (a new coloring), i.e. AES is the average number of fitness evaluations. A search step of DSatur, however, is a backtracking step, i.e. giving a node a new color. Thus, the computational

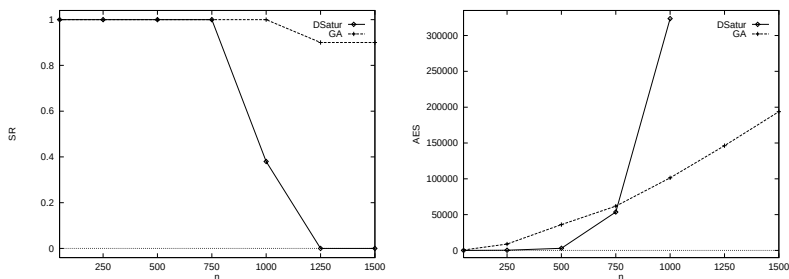


Fig. 6. Scale-up curves for SR (left) and AES (right) for $p = 10/n$.

complexity of DSatur is measured differently than that of an EA - a problem that cannot be circumvented since it is rooted in the different nature of these search algorithms. However, if we compare how the AES changes with growing problem sizes then regardless to the different meaning of ‘AES’ this comparison is fair.

6 Conclusions

In this paper we considered the Stepwise Adaptation of Weights mechanism that changes the penalty function based on measuring the error of solutions of a constrained problem during an EA run. Comparison of the SAW-ing EA with a simple EA, DSatur and a hybrid EA+DSatur system as showed in Table 1 discloses that SAW-ing is not only powerful (highly increases the success rate and the speed), but also robust (the performance becomes independent from the random seeds).

Besides comparing different EA versions we also conducted experiments to compare EAs with a traditional graph coloring heuristic. These experiments show that our SAW-ing EA outperforms the best heuristic we could find in the literature, DSatur.

The exact working of the SAW mechanism is still an open research issue. The plot in Figure 3 suggests that a SAW-ing EA solves the problem in two phases. In the first phase the EA is *learning* a good setting for the weights. In this phase the penalty increases a lot because of the increased weights. In the second phase the EA is *solving* the problem, exploiting the knowledge (appropriate weights) learned in the first phase. In this phase the penalty drops sharply indicating that using the right weights (the right penalty function) in the second phase the problem becomes ‘easy’. This interpretation of the fitness curves is plausible. We, however, do not claim that the EA could learn universally good weights for a given graph instance. First of all, another problem solver might need other weights to solve the problem. Besides, we have applied a SAW-ing EA to a graph and recorded the weights at termination. In a following experiment we have applied an EA to the same graph using the learned weights non-adaptively, i.e. keeping them constant along the evolution. The results became *worse* than in the first run when adaptive weights were used. This suggests that reason for the success of the SAW mechanism is not that SAW-ing enables the problem solver to discover some hidden, universally good weights. This seems to contradict our interpretation that distinguishes two phases of search. Another plausible explanation of the results is based on seeing the SAW mechanism as a technique that allows the EA to shift the focus of attention, by changing the priorities given to different nodes. It is thus not the weights that is being learned, but rather the proportion between the weights. This can be perceived as an implicate problem decomposition: ‘solve-these-nodes-first’. The advantage of such a (quasi) continuous shift of attention is that it finally guides the population through the search space, escaping local optima.

At the moment there are a number of evolutionary constraint handling techniques known and practiced on constraint satisfaction as well as on constrained optimization problems, [11, 26, 27]. Penalty functions embody a natural and simple way of treating constraints, but have some drawbacks. One of them is that the composition of the penalty function has a great impact on the EA performance, in the meanwhile penalty functions are mostly designed in an ad hoc manner. This implies a source of failure, as wrongly set weights may cause the EA fail to solve the given problem. The SAW mechanism eliminates this source of failure in a simple and problem independent manner.

Future research issues concern variations of the basic SAW mechanism applied here. These variations include using different Δw 's for different variables or constraints, as well as subtraction of Δw from w_i 's that belong to well instantiated variables, respectively satisfied constraints. An especially interesting application of SAW concerns constrained optimization problems, where not only a good penalty function needs to be found, but also a suitable combination of the original optimization criterion and the penalty function.

References

1. A. Blum. An $O(n^{0.4})$ -approximation algorithm for 3-coloring (and improved approximation algorithms for k -coloring). In *Proceedings of the 21st ACM Symposium on Theory of Computing*, pages 535–542, New York, 1989. ACM.
2. D. Brélaz. New methods to color vertices of a graph. *Communications of the ACM*, 22:251–256, 1979.
3. G.J. Chaitin. Register allocation and spilling via graph coloring. In *Proceedings of the ACM SIGPLAN 82 Symposium on Compiler Construction*, pages 98–105. ACM Press, 1982.
4. P. Cheeseman, B. Kenefsky, and W. M. Taylor. Where the really hard problems are. In *Proceedings of the IJCAI-91*, pages 331–337, 1991.
5. S.H. Clearwater and T. Hogg. Problem structure heuristics and scaling behavior for genetic algorithms. *Artificial Intelligence*, 81:327–347, 1996.
6. J.C. Culberson and F. Luo. Exploring the k -colorable landscape with iterated greedy. In *Second DIMACS Challenge, Discrete Mathematics and Theoretical Computer Science*. AMS, 1995. Available by <http://web.cs.ualberta.ca/~joe/>.
7. L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, 1991.
8. A.E. Eiben. Multi-parent recombination. In T. Bäck, D. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*. Institute of Physics Publishing Ltd, Bristol and Oxford University Press, New York, 1997. Section C3.3.7, to appear in the 1st supplement.
9. A.E. Eiben, P.-E. Raué, and Zs. Ruttkay. Constrained problems. In L. Chambers, editor, *Practical Handbook of Genetic Algorithms*, pages 307–365. CRC Press, 1995.
10. A.E. Eiben and Zs. Ruttkay. Self-adaptivity for constraint satisfaction: Learning penalty functions. In *Proceedings of the 3rd IEEE Conference on Evolutionary Computation*, pages 258–261. IEEE, IEEE Press, 1996.
11. A.E. Eiben and Zs. Ruttkay. Constraint satisfaction problems. In Th. Bäck, D. Fogel, and M. Michalewicz, editors, *Handbook of Evolutionary Algorithms*, pages C5.7:1–C5.7:8. IOP Publishing Ltd. and Oxford University Press, 1997.

12. A.E. Eiben and J.K. van der Hauw. Graph coloring with adaptive evolutionary algorithms. Technical Report TR-96-11, Leiden University, August 1996. also available as <http://www.wi.leidenuniv.nl/~gusz/graphcol.ps.gz>.
13. A.E. Eiben and J.K. van der Hauw. Solving 3-SAT with adaptive Genetic Algorithms. In *Proceedings of the 4th IEEE Conference on Evolutionary Computation*, pages 81–86. IEEE, IEEE Press, 1997.
14. E. Falkenauer. A new representation and operators for genetic algorithms applied to grouping problems. *Evolutionary Computation*, 2(2):123–144, 1994.
15. E. Falkenauer. Solving equal piles with the grouping genetic algorithm. In S. Forrest, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 492–497. Morgan Kaufmann, 1995.
16. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freedman and Co., 1979.
17. M.R. Garey, D.S. Johnson, and H.C. So. An application of graph coloring to printed circuit testing. *IEEE Trans. on Circuits and Systems*, CAS-23:591–599, 1976.
18. R. Hinterding, Z. Michalewicz, and A.E. Eiben. Adaptation in Evolutionary Computation: a survey. In *Proceedings of the 4th IEEE Conference on Evolutionary Computation*, pages 65–69. IEEE Service Center, 1997.
19. D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation; part II, graph coloring and number partitioning. *Operations Research*, 39(3):378–406, 1991.
20. L. Kučera. The greedy coloring is a bad probabilistic algorithm. *Journal of Algorithms*, 12:674–684, 1991.
21. P. Moris. The breakout method for escaping from local minima. In *Proceedings of the 11th National Conference on Artificial Intelligence, AAAI-93*. AAAI Press/The MIT Press, 1993.
22. B. Selman and H. Kautz. Domain-independent extensions to GSAT: Solving large structured satisfiability problems. In R. Bajcsy, editor, *Proceedings of IJCAI'93*, pages 290–295. Morgan Kaufmann, 1993.
23. J.S. Turner. Almost all k -colorable graphs are easy to color. *Journal of Algorithms*, 9:63–82, 1988.
24. G. von Laszewski. Intelligent structural operators for the k -way graph partitioning problem. In R.K. Belew and L.B. Booker, editors, *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 45–52. Morgan Kaufmann, 1991.
25. D. De Werra. An introduction to timetabling. *European Journal of Operations Research*, 19:151–162, 1985.
26. Michalewicz Z. and Michalewicz M. Pro-life versus pro-choice strategies in evolutionary computation techniques. In Palaniswami M., Attikiouzel Y., Marks R.J., Fogel D., and Fukuda T., editors, *Computational Intelligence: A Dynamic System Perspective*, pages 137–151. IEEE Press, 1995.
27. Michalewicz Z. and Schoenauer M. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1):1–32, 1996.

An Agent System for Learning Profiles in Broadcasting Applications on the Internet

Cristina Cuenca and Jean-Claude Heudin

International Institute of Multimedia
Pôle Universitaire Léonard de Vinci
92916 Paris La Défense - FRANCE
Cristina.Cuenca@devinci.fr
Jean-Claude.Heudin@devinci.fr

Abstract. In this paper we present the research results about the adequation of evolutionary algorithms and multi-agent systems to learn user's preferences during his interactions with a digital assistant. This study is done in the framework of "broadcasting" on the Internet. In our experiment, a multi-agent system with a Genetic Algorithm is used to globally optimize a user's selection of "channels" among a very large number of choices. We show that this approach could solve the problem of providing multiple optimal solutions without losing diversity.

1. Introduction

In the Internet age, users can be often overwhelmed by the very large amount of information they can find related to their search. With the basic search engines available today, based on text-retrieval, they are forced to begin querying using a few words. In most cases, this approach results in a large amount of irrelevant text that seems to contain words or concept derivations related to their search.

Facing this problem, another paradigm has been recently proposed, based on a "broadcasting" approach [Lemay97]. When you turn on a television set, you get a large number of channels to pick from: sports, news, home shopping, movies, music, etc. If you are like most of the people, you have probably got a handful of favorite channels you tune in all the time, some that you watch occasionally and others you always click past. The idea is to apply this scheme to Internet, providing "transmitters", "channels" and "tuners".

Following Marimba's terminology¹, a channel is an application or content to be distributed across the Internet. A transmitter runs in a server to deliver and maintain these channels. A tuner installed in your computer allows channel subscription and maintains, receives and manages the subscribed channels. With a tuner software

¹ URL: <http://www.marimba.com>

installed on your computer, you must be able to "tune in" to channels in the Internet in a similar way you tune in your favorite television channels.

However, even if simpler by nature, the problem remains in how to find the channels you are looking for.

In most existing approaches, you must "teach" a software about the information that is most useful to you –the music and movies you like best, for example, by storing your tastes and preferences in a form. This information is restricted to a selection of terms or a few keywords and stored in a database. Then, the system must be able to connect you directly to "channels" that fit your requirements or personal interests. This approach assumes that you know precisely what you are looking for. In addition, you must explicitly update the database if the results are irrelevant or if you simply change your points of interest.² The most recent approaches infer your preferences statistically, by making analogies between you and other peoples with similar tastes. That is the case of Firefly³, a system derived from the HOMR (Helpful On-Line Music Recommendation) project at the MIT Media Lab [Sheth93].

The aim of our project is to design a software prototype based on a multi-agent architecture which automatically learns and updates profiles in order to select channels and subscribe to programs more effectively. Transmitters and tuners are therefore considered as agents which will work and make decisions based on communication and agreements with other agents [Rosenschein94]. In particular, each tuner will include a personal digital assistant which will help the user to find the channels he is looking for. The kind of digital assistants we are interested in are self-motivated and based on an Artificial Life approach [Langton88][Heudin94].

In this paper we explore the adequation of a multi-agent system and a Genetic Algorithm on the problem of learning a user's preferences during interactions between the digital assistant and the user. We present our experiment and describe the general outlines of our solution. Then, we give our first results and discuss them. Finally, some future research directions are sketched.

2. The Experiment

The goal of the experiment is to create a digital assistant which proposes an Internet user the channels he enjoys. This assistant will learn the user's preferences during the utilization process. That means that it should be capable of dealing with this in an evolutive way, considering that a person's tastes can vary over time, temporarily or definitively. Another aspect is that a person's taste is not forced to be unique (that could be boring for him to "see" the same and only the same "channels" all the time), but he is likely to accept (and probably want) propositions of topics he doesn't know or he doesn't ask for explicitly.

² You can experience this approach at PointCast site, URL: <http://www.pointcast.com>

³ URL: <http://www.firefly.com>

2.1 Project Architecture Overview

The entire project includes two parts: the server part and the client part. The server has to provide contents to the subscribing clients and for this reason, it has to maintain client accounts and information related to its content offer. The client, by means of a multi-agent system, interacts with the user, proposes him programs, keeps track of his preferences, and negotiates the information it needs with the different servers. The general scheme is shown in Fig. 1. This paper describes only the client part of the system.

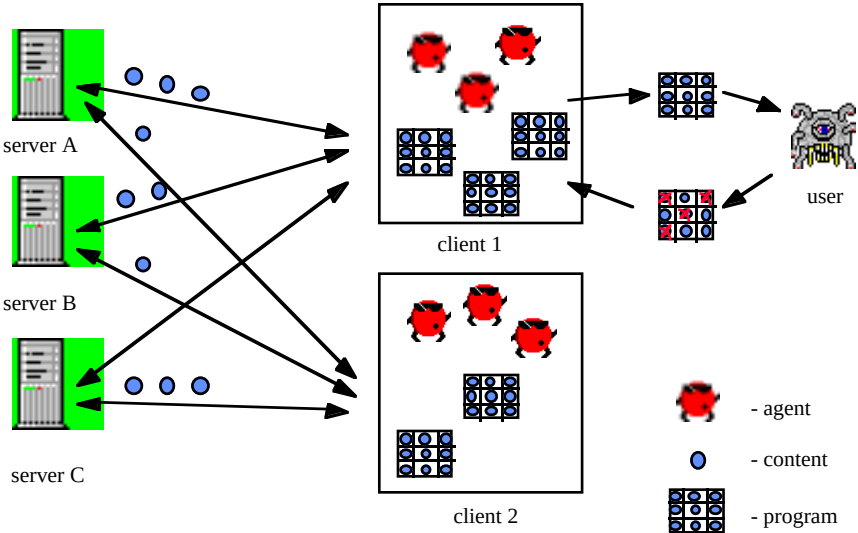


Fig. 1. The system

2.2 The Approach

Our main interest in this paper was testing the applicability of a Genetic Algorithm working in a multi-agent system as a solution to the problem of learning a user's preferences, making accurate propositions reflecting his tastes while preserving diversity in the offer. In a first stage of our work, the fact of modeling and categorizing the information to be chosen by the user was not a main objective. That's why we chose to begin facing the problem by the means of a metaphor to represent this information variety. This metaphor is explained in more details in the next section.

In a second stage, we are going to transpose this representation to its real context. It means that we are going to change the metaphor in order to work with real information, that could be video files, music files, image files, html files, etc. This process requires also a modeling of the information that is going to be processed, because the used techniques need a data quantification, as you will see later in this

paper. A start point for doing this could be the work of Sheth and Maes in Personalized Information Filtering [Sheth93].

2.3 The Metaphor

We use a graphical color panel as an abstraction of a recommendation of "channels". Color squares in the panel simulate different information, movies, news, videos, etc. Thus, when the system suggests a color panel (a "program"), the user can select its preferences by simply clicking in the color squares with the mouse. In our experiment, we use panels composed by 16 color squares.

Each color is encoded using the RGB format where the red, blue and green components are each represented by an integer in the range 0-255. Thus, we are working with 16 777 216 colors and $(16 \text{ squares} \times 16\,777\,216 \text{ colors}) = 268\,435\,456$ possible panels. This approach has the advantage of providing a large search space and also an enormous diversity of choice, which can be regarded also as categories (color families).

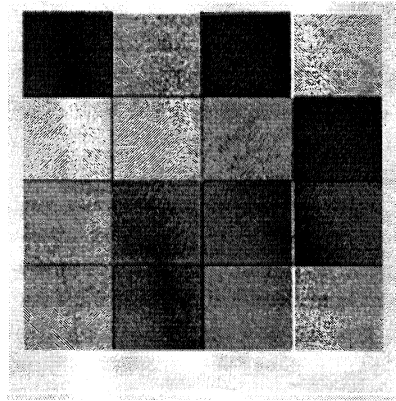


Fig. 2. A gray representation of the color panel

2.4 The Multi-Agent Architecture

We have designed a multi-agent architecture [Ferber95] based on a pyramidal model of data fusion which have been already used in the Electronic Copilot project [Gilles91]. The agents communicate with each others in a pyramidal way, according to the fact of reducing the information volume and increasing its semantics in each pyramid level. In its current implementation, the system includes five agents (cf. Fig 3).

Each agent can be purely reactive, cognitive and adaptive, depending on its objectives. Each agent's specific task will be explained in the next sections.

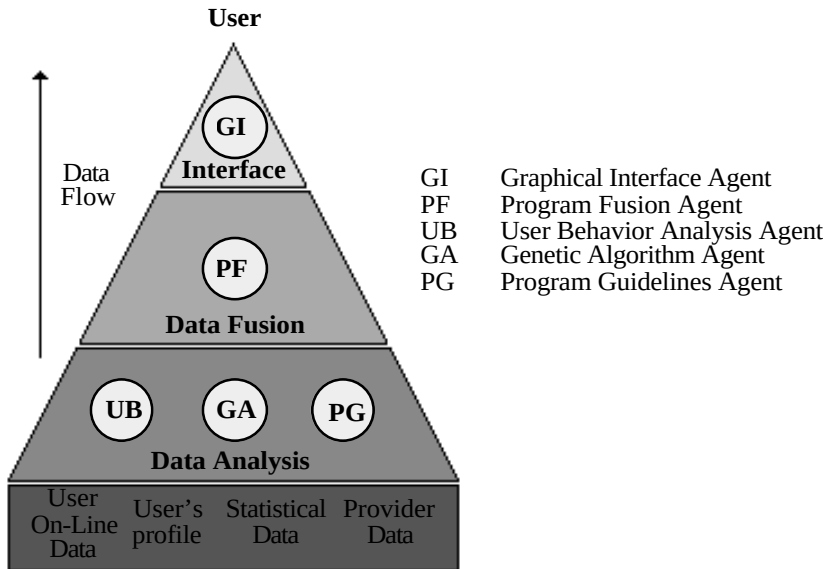


Fig. 3. The scheme

Graphical Interface Agent

It takes the "program" from the Fusion Agent and shows it to the user. It also allows the user's interaction with the panel. When a program is proposed to the user, a color panel is shown, and then, the user can select some of the colors in the panel with the mouse –to indicate the channels he wants to "tune in"(cf. figure 4).

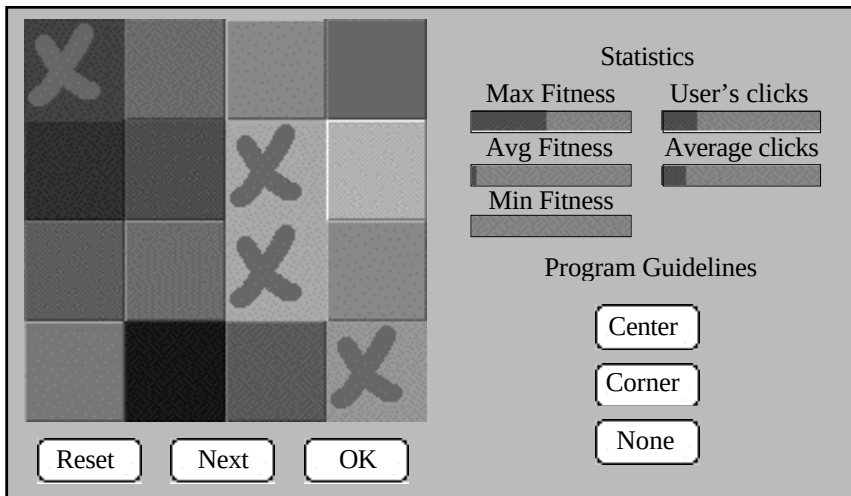


Fig. 4. A selection of colors in the graphical interface

Program Fusion Agent

It is in charge of the fusion of the Genetic Algorithm's results, and the Programs Proposition agent choice, in order to generate the "program".

User's Behavior Analysis Agent

This agent is in charge of the detection of the user's behavior, creating and maintaining a user's profile that will be used by other agents. In this stage of our work, this agent takes into account the user's number of clicks on the color panel, but in the next step it will use this information in order to identify its different states.

These states are:

- **Learning:** the user begins to use the system and start to teach it his preferences.
- **Accepting the suggestions:** The system has learnt the users preferences and proposes suitable programs.
- **Changing his mind:** The user has modified its preferences and begin to ask for new propositions. At this point, it is necessary to distinguish between a temporary change of mind and a definitive one.

It will also detect specific user's preferences, considering the clicks on specific panel positions.

Programs Guidelines Agent

A program guidelines server, charged of the generation of "programs" guidelines. A "program" is what the user is finally going to see, the selected channels presented to him in a coherent fashion. Using the metaphor, it includes the color panel the user is going to interact with but it considers also the way of reordering the colors in the panel in the most appropriate way. In this first stage, the reordering proposition agent proposes three different ways to "view the program":

- Without a specific organization, just as the proposition is done by the Genetic Algorithm.
- Putting the user's preferred colors in the center of the square.
- Putting the user's preferred colors in the corners of the square.

The behavior of this agent will be studied in a next step of our work.

Genetic Algorithm Agent

Responsible for modeling and learning the user's interests and preferences. It takes the user's profile information to make the color panel proposition evolve in order to please the user. A detailed explanation comes later.

2.5. Genetic Algorithm Agent

As we introduced previously, this is the central part of our work. This agent is responsible for the learning process and for the evolution of the program proposition.

The Algorithm

The general outline of the XY algorithm is a usual generational scheme [Heudin94]. A population Π of P individuals undergoes a succession of generations :

- **Select** two “good” and two “bad” individuals according to their relative fitness using a selection table;
- **Recombine** the two “good” selected individuals with probability C_{prob} using a single-point crossing over operator;
- **Mutate** offspring of the two parents with probability M_{prob} using an adaptive mutation operator;
- **Replace** in the population the two “bad” individuals by the offspring.

Instead of a classical iteration with a stopping criterion, and because we are not looking for an optimal and final solution (the user's interaction never ends) the algorithm develops each new generation continuously. When the user demands a new program, the algorithm selects its best individual and send it to the fusion agent in order to be displayed using the graphical interface.

The user's interactions are kept in his profile, and the algorithm can consider them at each step. The main drawback is that it must propose relatively “good” solutions in few cycles since the user cannot wait hundreds of cycles. However, it is common to have a few “not-so-bad” individuals in a population of mediocre colleagues in the first iterations. The real problem is to avoid these “good” individuals taking over a significant proportion of the finite population leading to premature convergence.

Coding Panels

We select a natural expression of the problem for coding. Each individual (a panel of 16 colors), is represented by a string of 16 32-bit integers. Each integer is composed of 8 unused bits and a red, blue, and green 8 bits components.

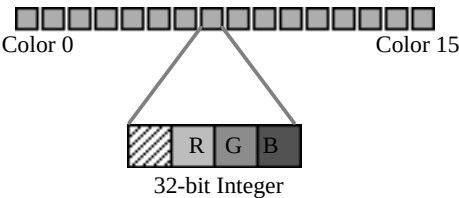


Fig. 5. The coding of an individual

Selection

Proportional selection with and without linear fitness scaling have been used. In order to reduce the stochastic errors generally associated with roulette wheel selection, we use a remainder stochastic sampling without replacement as suggested by Booker [Booker82]. Our implementation is based on a selection table and is closed to the one proposed by Goldberg [Goldberg89].

Single-point Crossover

The two selected parents are recombined using a single-point crossover operator. It is achieved by a linear combination of both parents and produces two offspring. This is simply done by choosing a random site in the interval [0, 15] and swapping the right-hand parts of the parents. The probability of recombination is $C_{\text{prob}} = 0.7$.

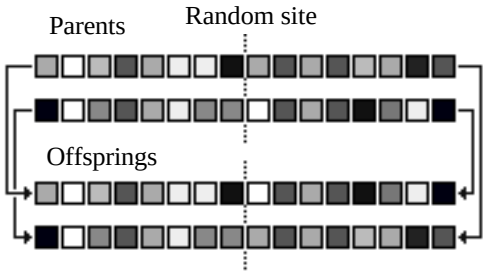


Fig. 6. Single-point crossover

Adaptive Mutation

Mutation is applied to the offspring created by crossover. Our algorithm use an adaptive mutation operator inspired from Evolution Strategies [Rechenberg73] and real-encoded Genetic Algorithms [Eschelman93]. Each color in an individual includes its own mutation probability M_{prob} . This probability is also used for computing a deviation interval in order to add some Gaussian noise to the selected color. All colors in an individual are initialized with $M_{\text{prob}} = 0.4$. Then, this value is increased or decreased according to the number of times it has been selected by the user.

Fitness Function

The fitness function depends on the history of the user selections. As we explained in the previous section, each color in an individual includes its own mutation probability. When the user clicks on a color square, that means that he likes this color, so the corresponding mutation probability will be decreased. On the contrary, if the user doesn't click on a color, this value will be increased.

That allows us to use the mutation probabilities of an individual as a measure of the user preferences, in order to compute its fitness value.

$$f(x) = (1 / N) \cdot n_x$$

where x is the current panel (an individual), N is the number of color squares in the panel and n_x is the number of color squares in the panel with a mutation probability that is inferior to the default mutation probability value (M_{prob}). Note that the fitness function is only based on the number of selected colors. In other terms, the algorithm doesn't distinguish colors, as it works at the panel level, not at the color level.

3. First Results

3.1. Quantitative Overview

Three basic user's behavior cases have been studied. In the first case, the user has a very restricted taste, and begins selecting less than 20% of the colors squares in the panel. In the second one, he begins selecting more than 20% of the colors squares. In both cases, it is assumed that the user has made a choice of the colors he wants to see and does not change his mind during the experiment. In the third case the user makes a first choice and after a long interaction period, he suddenly changes to another color.

The three cases are illustrated by graphics which represent the population statistics (maximum, minimum and average fitness values of the population) considering two runs for each case: applying the fitness function as it is, and applying a fitness scaling, during some generations of the genetic algorithm. As the algorithm works independently of the user interaction, we computed the statistics in relation to the number of generations, but we estimated also the number of user interactions with the system during this time.

First Case: The user has a wide interest

In this case, the user is interested in two colors and all the color range between both. For example, if he likes blue and green, he clicks on all their tonalities, their lights and darks, and all the mixtures between both. In practice, this reduce the search space from 16 777 216 to about 10. Therefore, the probability for one of this colors to appear is 1/10 and we have a good chance to select 3-4 squares in each panel.

After 100 generations (during this time, the user has interacted only 7 times with the system), the maximum fitness is 0.8, and 1 after 14 user interactions. The use of a fitness scaling reduces these numbers to approximately 0.2.

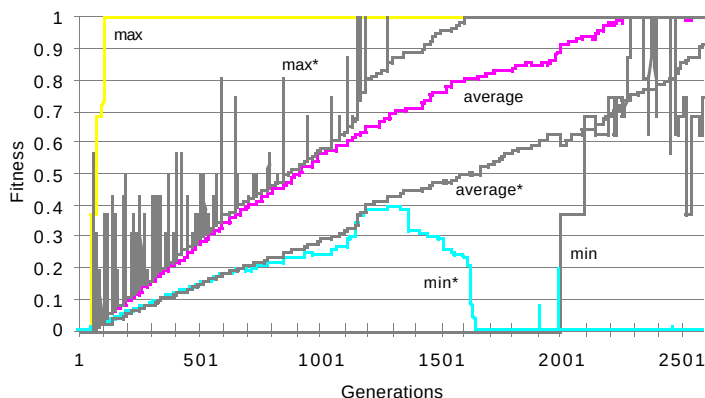


Fig. 7. First case with (*) and without fitness scaling

Second Case: The user has a precise interest

This time, the user chooses only one color, and no derivations are selected. This behavior leads to click in no more than 3 colors in a panel. This produces a slower convergence rate. In the example, after 400 generations (25 user interactions), the maximum fitness value is 0.5. After 45 user interactions, its value is 0.8. With the scaling mode, in the same conditions, this values are approximately 0.2 and 0.4.

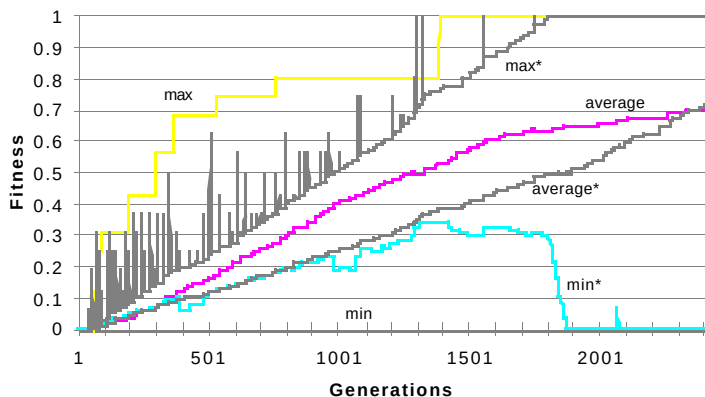


Fig. 8. Second case with (*) and without fitness scaling

Third Case: The user changes his mind

In this case, the user first chooses two colors during a long period of time, allowing the algorithm to quickly have good fitness values. Then, he decides to change his mind and to choose another color. The algorithm will not answer to this

change until a long period of time, since it will continue to propose the well adapted individuals according to the initial criteria.

In this example, after 2000 generations (200 interactions), the user tries to select a different color. The average fitness begins slowly to go down, but after 400 interactions the population is always full with the old good individuals.

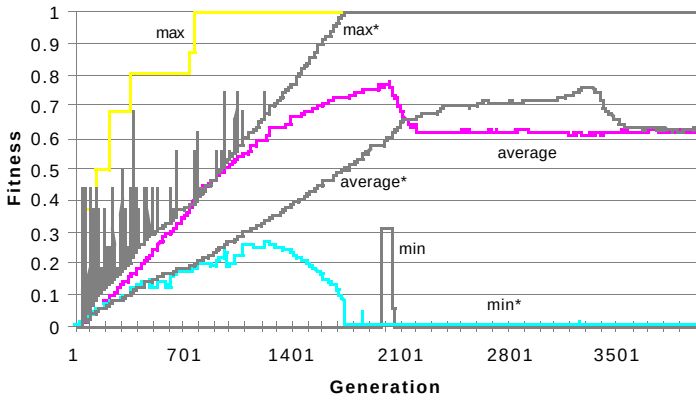


Fig. 9. Third case with (*) and without fitness scaling

3.2. Qualitative Overview

We have observed that the system learns quickly the user's preferences. That is a result of the continuous action of the genetic algorithm, which runs independently of the user's interaction, contributing to a faster convergence. Better than that, as a consequence of always displaying the best population member, the user soon receives programs that fits his taste.

However, the biggest inconvenient in those cases was the fact that, once the population has been filled with "goods individuals", it is very difficult for the algorithm to react to a user's change of mind.

On the other hand we can see that, as the algorithm computes the fitness function value for an individual if it is presented to the user, there are some members of the population which are not exposed to evaluation (in the three graphics, the min fitness value of the population remains low almost all the time). This is not really a problem since in this prototype we are working with a static population. In the real application, this will not be the case, because new individuals will arrive constantly from the servers and the system should be able to evaluate them in relation to the existing population fitness values.

3. Discussion

Our first results lend support to the hypothesis that an evolutionary approach is well-suited to the problem of generating selections of channels based on the learning of a user's preferences. After few cycles of interactions with a user, quite "good"

selections are proposed. In the long run, the population converges towards an optimal point without losing too much diversity. This diversity of channels selections is important since our goal is improvement. Attainment of the optimum is much less important in our application. We never judge a television channel by an attainment-of-the-best criterion. Perfection, that is convergence to the best, is not an issue in most walks of life.

However, we feel that the major problem with the approach we have presented here, in the actual stage of our work, is that the system does not respond correctly when the user suddenly changes his points of interest. That's why our next step is to extend the tasks of the User Behavior Analysis Agent to model the user's different states and to improve the user's profile representation.

A complementary approach to solve this problem is the use of diploidy and dominance [Goldberg89]. The redundant memory of diploidy permits multiple solutions to be carried along with only one particular solution expressed. Then, when a dramatic change in environmental condition occurs, a shift in dominance allows to express alternative solutions held in the background, allowing a rapid adaptation to the selective pressures of the changing environment.

On the other hand, the use of the metaphor is not longer suitable, because it begins to be a constraint for the experiment. A modeling of the problem considering real "documents" is one of our next steps.

4. Implementation

The experiment has been implemented as a JAVA applet [Gosling94]. This choice is motivated by the nature of the application framework and the language features (object-oriented, architecture-neutral, easy prototyping, etc.) The agents were implemented using the JAVA multi-thread mechanism.

5. Conclusions and Future Works

We have presented an application of evolutionary computing to the learning of a user's preferences. Our experiment based on a multi-agent architecture and the use of a Genetic Algorithm confirms the idea that such approach could be applied successfully in the real framework of "broadcasting" in the Internet. This preliminary results are encouraging.

Future works include the study of an algorithm that combines evolution and other learning techniques (like the Evolutionary Reinforcement Learning [Ackley91]). Our next steps are also the improvement of the user's profile representation and the User's Behavior Analysis Agent, combined with changing the metaphor in a more realistic model.

6. References

Ackley D., Littman M. (1991). Interactions Between Learning and Evolution.

Artificial Life II. SFI Studies in the Sciences of Complexity, vol. X, edited by C.G. Langton, C. Taylor, J.D. Farmer, S. Rasmussen, Addison-Wesley.

Booker L.B. (1982). *Intelligent Behavior as an Adaptation to the Task Environment*. Doctoral Dissertation, Technical Report n. 243, University of Michigan, Ann Arbor.

Eshelman L., Schaffer J.D. (1993). Real-coded Genetic Algorithms and Interval-Schemata. In *Foundations of Genetic Algorithms 2*, p. 187-202, edited by L.D. Whitley, Morgan Kaufmann, Los Altos.

Ferber J. (1995). *Les Systèmes Multi-Agents*, p. 15, InterEditions.

Gilles A., Lebiannic Y., Montet P., Heudin J.C. (1991). *A Parallel Multi-Expert Architecture for the Copilote Electronique*. 11th International Conference on Expert Systems and their applications, EC2, Avignon, France.

Goldberg D.A. (1989). *Genetic Algorithms in Search, Optimization & Machine Learning*. p 121-124, Addison-Wesley.

Gosling J. (1994). *The Java Language : A White paper*. Sun Microsystems.

Heudin J.C. (1994). *La Vie Artificielle*. Hermès, Paris.

Langton C.G. (1988). Artificial Life. *Artificial Life*. SFI Studies in the Sciences of Complexity, vol. VI, edited by C.G. Langton. Addison-Wesley.

Lemay L. (1997). *The Official Marimba Guide to Castanet*. Sams.Net.

Rechenberg I. (1973). *Evolutionstrategie : Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*. Fromman-Holzboog Verlag, Stuttgart.

Rosenschein J.S., Zlotkin G. (1994). Designing Conventions for Automated Negotiation, *AI Magazine*, Vol. 15, n. 3, p. 29, AAAI, Menlo Park.

Sheth B., Maes P. (1993). Evolving Agents for Personalized Information Filtering, *Proceedings of the Ninth Conference on Artificial Intelligence for Applications '93*, Orlando, Florida, IEEE Computer Society Press

Application of evolutionary algorithms to protein folding prediction

A. Piccolboni and G. Mauri

Dipartimento di Scienze dell'Informazione
Università di Milano
Via Comelico 39/41, 20135 Milano, Italy

Abstract. The aim of this paper is to show how evolutionary algorithms can be applied to protein folding prediction. We start reviewing previous similar approaches, that we criticize emphasizing the key issue of representation. A new evolutionary algorithm is described, based on the notion of distance matrix representation, together with a software package that implements it. Finally, experimental results are discussed.

1 Protein folding prediction

Proteins are molecules of extraordinary relevance for live beings. They are chains of aminoacids (called also residues) that assume very rich and involved shapes *in vivo*. The prediction of protein tertiary structure (3D shape) from primary structure (sequence of aminoacids) is a daunting as well as a fundamental task in molecular biology. A large amount of experimental data is available as far as it concerns sequence, and large projects are creating huge amounts of sequence data. But to infer the biological function (the ultimate goal for molecular biology) from sequence we have to pass through tertiary structure, and how to accomplish this is still an open problem (*ab initio* prediction). Indeed, experimental resolution of structures is a difficult, costly and error-prone process. The prediction problem can be recast as the optimization of the energy function of a protein, under the assumption that an accurate enough approximation of this function is available. Indeed, according to the so-called *Anfinsen hypothesis* [AHSJ61], native conformations correspond, at a first approximation, to global minima of this function.

2 Evolutionary algorithms

Evolutionary algorithms (EAs) [Hol75, De 75, FOW66, BS93] are optimization methods based on an evolutionary metaphor that showed effective in solving difficult problems. Distinctive features of EAs are:

- a set of candidate solutions is considered at each time step instead of a single one (population);
- candidate solutions are combined to form new ones (mating operator);

- solutions can be randomly slightly modified (mutation operator);
- better solutions according to the optimization criterion (fitness) are given more reproductive trials.

These basic principles result in an overall population dynamics that can be roughly described as the spreading of good features throughout the population. This naive idea is made more precise in the so called “schema theorem” [Hol75].

According to [Gol90], the best performances can be obtained when hyperplanes in the solution space with under average energy exist (building blocks). According to the more general definition found in [Vos91], a building block is a property of solutions which is inherited with high probability by offsprings, i.e. which is (almost) preserved after crossover and mutation and is rewarding from the point of view of fitness. In the following we will refer also to genetic algorithms (GAs) which are a special case of EAs.

3 Previous work

The influence of representation on the dynamics and effectiveness of EAs has been already recognized [BBM94]. Three main representation techniques have been proposed for protein structures:

Cartesian coordinates are unsuitable for a population based algorithm, since basically identical structures (up to a roto-translation) can have completely different coordinates;

internal coordinates define aminoacid positions w.r.t. neighboring aminoacids, specifying distances and angles; this is the choice of all genetic approaches to protein folding so far;

distance geometry describes a structure by means of the matrix of all the distances between every couple of points and has been proposed as a tool for energy minimization since [NS77]; our main contribution is its joint use together with EAs.

To the best of our knowledge, all evolutionary approaches to folding prediction so far have been based on an internal coordinate representation. It is straightforward to show that relevant structural features can not be described as hyperplanes under this approach.

Schulze-Kremer [SK93, SK95] defines a real-coded GA in internal coordinate space and a simplified energy function, but fails *ab initio* prediction for a test protein. His algorithm proves useful for side chain placement.

Unger and Moult [UM93] compare a GA against Monte Carlo methods using an idealized 2D lattice model and simplified internal coordinates. Large performance gains over Monte-Carlo are achieved but no comparison is possible with real proteins (Patton *et al.* [PPG95] report an improvement to this approach and we will compare our results to theirs).

Dandekar and Argos [DA94] use a standard GA with an heuristic, heavily tailored fitness and an internal coordinate discretized representation. Results are encouraging, but the generality of the method is questionable.

Herrmann and Suhai [HS95] use a standard genetic algorithm in internal coordinate space together with local search and a detailed model. It proved interesting only for very small structures.

A simple observation against internal coordinate representation is the following: typical force fields are a sum of terms like relaxation distances or relaxation angles that are convex functions of pairwise distances. It is likely that structures minimizing these relaxation terms will have a lower energy and optimal structures have a lot of relaxation terms close to zero. So distances and angles can act as building blocks for genetic search. But internal coordinates include only distances between neighboring residues in the sequence. The distance between an arbitrary pair of residues can be calculated only using a complex formula involving coordinates of all the residues that appear between the two in the sequence, i.e., in the worst case, the whole representation. The same is true for angles. It is very difficult, thus, to describe useful schemas in internal coordinate space and guarantee some minimal properties such as *stability* [Vos91], *low epistasis* [BBM93] and others that can not guarantee the success of a GA but are believed to be key ingredients of it [Gol90]. On the contrary, we show that some of these properties hold for suitable genetic operators in distance matrix space.

4 Energy function

We tried to keep our model as simple as possible. We model each residue as a unique point and according to [Dil85] we consider the hydrophobic/hydrophilic interaction as the dominant force that drives folding. Under this assumption, protein instances can be seen as sequences of either hydrophobic or hydrophilic aminoacids. Thus our energy function E_{tot} is made up of three terms only

$$E_{tot} = E_{rep} + E_{chn} + E_{hyd}$$

that we will describe briefly. E_{rep} is a “solid body” constraint penalty term, that prevents two residues from occupying the same place. Namely we have

$$E_{rep} = k_{rep} \sum_{i=1}^{N-2} \sum_{j=i+2}^N g(d_{ij})$$

where N is the length of the protein, k_{rep} a suitable constant, d_{ij} is the distance between residues i and j and g is defined as

$$g(x) = \begin{cases} x - 2 + \frac{1}{x} & 0 < x \leq 1 \\ 0 & x > 1 \end{cases}$$

E_{chn} is a chain constraint penalty term, that forces neighboring residues in the sequence to lay spatially close, whose form is

$$E_{chn} = k_{chn} \sum_{i=1}^{N-1} (d_{i,i+1} - 1)^2$$

where k_{chn} is a constant. Finally, E_{hyd} is an hydrophobic interaction term, rewarding closeness between hydrophobic aminoacids and is defined as

$$E_{hyd} = k_{hyd} \sum_{i=1}^{N-2} \sum_{j=i+2}^N h(d_{ij})$$

where k_{hyd} is a constant and

$$h(x) = \begin{cases} \log((x-1)^2) + \frac{1}{e} & \text{if residue } i \text{ and } j \text{ are both hydrophobic} \\ 0 & \text{otherwise} \end{cases}$$

Although the exact form of the energy is subject to wide variations [Neu93], this energy function models at least qualitatively the most important structural properties of proteins.

From an evolutionary algorithm point of view this function satisfies the ideal condition of *no epistasis* [BBM93], since a change in a variable always produces the same change in energy, despite the values of all other variables. This is the main advantage of a distance based representation.

5 Distance matrix representation

We describe a different approach to folding prediction with EAs. The solution space is the set of aminoacid distance matrices, that is the set of $N \times N$ symmetric, zero diagonal matrices with positive entries representing pairwise distances between aminoacids. Unfortunately, it turns out that such a representation describes a superset of possible configurations. To cope with this fact and, in general, to deal with such a kind of representation we need some concepts belonging to *distance geometry* [HKC83].

Definition 1 A distance matrix $\mathbf{D} = \{d_{ij}\}$ is embeddable in $\mathbf{R}^n$ if there exist a set of N points in $\mathbf{R}^n$, $\mathbf{C} = \{\mathbf{c}_i\}$ s.t.

$$||\mathbf{c}_i - \mathbf{c}_j|| = d_{ij}$$

and such a $\mathbf{C}$ is called an embedding of $\mathbf{D}$.

We use the same notation for the set of points $\mathbf{C} = \{\mathbf{c}_i\}$ and for the $n \times N$ matrix with $\{\mathbf{c}_i\}$ as columns.

Definition 2 Given a matrix $\mathbf{D}$, its Gram matrix is $\mathbf{M} = \{m_{ij}\}$, with

$$m_{ij} = d_{i1}^2 + d_{j1}^2 - d_{ij}^2$$

Definition 3 Given a set of points $\mathbf{C}$, its metric matrix is $\mathbf{C}^T \mathbf{C}$.

Theorem 1. In an inner product space, if $\mathbf{C}$ is an embedding of $\mathbf{D}$ and $\mathbf{M}$ is the Gram matrix of $\mathbf{D}$, then $\mathbf{M}$ is equal to the metric matrix of $\mathbf{C}$.

In our context this equality will always hold. From an algorithmic point of view, $\mathbf{C}$ is computable from $\mathbf{M}$ by Cholesky factorization [Van92].

Theorem 2. *A matrix $\mathbf{D}$ is embeddable in $\mathbf{R}^n$ iff its Gram matrix $\mathbf{M}$ is positive semidefinite of rank at most n .*

We observe that, since $\mathbf{M}$ is $N \times N$ its rank can be at most N . We have obtained thus a very elegant way to define our solution space: it is the set of symmetric, zero diagonal matrices with positive entries whose metric matrix is positive semidefinite of rank at most 3.

6 EA specification

According to [MS96], among different constraint handling strategies for EAs we explored these two possibilities:

- whenever an unfeasible solution is produced “repair” it, i.e. find a feasible solution “close” to the unfeasible one (*repair strategy*);
- admit unfeasible individuals in the population, but penalize them adding a suitable term to the energy (*penalize strategy*).

6.1 The repair algorithm

We are given a distance matrix $\mathbf{D}$ which is, for some reason, unfeasible and we want to find a feasible solution “close” to it, where “close” is to be specified. First of all, we may safely assume that $\mathbf{D}$ is symmetric, zero diagonal with positive entries, since, as we will see it, is easy to enforce such properties in the population, whatever kind of GA we are defining.

Next, we turn our attention to positive semidefiniteness. We can test this property in polynomial time by evaluating the smallest eigenvalue (it is positive iff the matrix is positive semidefinite), but in case it isn’t verified there’s nothing we can do (apart from rejecting the solution). This shortcoming is common to other similar algorithms in the literature [HKC83] and we are currently investigating this issue. Our guess is that without positive semidefiniteness the problem of embedding a distance matrix could be much harder.

Let us suppose we are given a symmetric, zero diagonal matrix $\mathbf{D}$ with positive entries whose metric matrix M is positive semidefinite of rank $n > 3$, so that the condition on rank only is not satisfied. The repair algorithm proceeds as follows (it is a modification and generalization of the one in [LLR95]):

1. find a coordinate set $\mathbf{C}$ in $\mathbf{R}^n$, $n \leq N$ by Cholesky factorization;
2. compute the projection of $\mathbf{C}$ (with distance matrix $\mathbf{D}' = \{d'_{ij}\}$) onto a random hyperplane P through the origin; call it $\mathbf{C}'$;
3. multiply $\mathbf{C}'$ element-wise by $\sqrt{n/3}$ (in the following we will consider, for the sake of generality, an m -dimensional hyperplane and a multiplicative factor of $\sqrt{n/m}$).

We have the following

Theorem 3. *With probability greater than $1/e$, for each i, j*

$$\left| \frac{d_{ij}^2 - d'_{ij}{}^2}{d_{ij}^2} \right| \leq O(\sqrt{n \log N})$$

Proof. (It is a generalization of the one in [Tre96]) Let us introduce the random variable (call it *distortion*)

$$X = \left| \frac{d_{ij}^2 - d'_{ij}{}^2}{d_{ij}^2} \right|$$

and the quantities

$$y_k = ||c'_{ik} - c'_{jk}||^2$$

(we can temporarily forget about the dependence of y_k from i and j since what we are going to prove with y_k 's is true for every i, j). Without loss of generality we may assume that P is a coordinate hyperplane (whose dimension is m to be more general, but we are interested in $m = 3$). This is because we are interested in properties of distances, which are invariant under rotation. Since

$$c'_{ik} = \sqrt{\frac{n}{m}} c_{ih}$$

for some h , depending only on P , we have that

$$d'_{ij}{}^2 = \frac{n}{m} \sum_{k=1}^m y_{I_k}$$

whereas

$$d_{ij}^2 = \sum_{k=1}^n y_k$$

where I_k are the indices of dimensions parallel to P , so that $d'_{ij}{}^2$ is a random variable which is the sum of a subset of size m of $\{y_k\}$ multiplied by n/m . Let

$$Y_k = \frac{\frac{n}{m} y_{I_k} - \frac{1}{m} \sum_{i=1}^n y_i}{\sum_{y=1}^n y_i}$$

so that

$$X = \left| \sum_K = 1^m Y_k \right|$$

Straightforward calculations show that $E[Y_k] = 0$ and $-1 \leq Y_k \leq 1$. We are ready to apply Hoeffding inequality [Hoe63] to obtain

$$P \left(X \leq \frac{1}{2} \sqrt{n(\log 2 + \log(N^2 + 1))} \right) \geq 1 - \frac{1}{N^2 + 1}$$

The probability of the same bound to hold for all N^2 distances at once is greater or equal than $(1 - \frac{1}{N^2 + 1})^{N^2}$ which is greater than $1/e$. Q.E.D.

6.2 The penalization term

We would like to be able to measure the “level of unfeasibility” of a solution $\mathbf{D}$. As with the repair algorithm, we take for granted that $\mathbf{D}$ has positive entries and zero diagonal. Next, in case $\mathbf{D}$ metric matrix is not positive semidefinite, we assign $\mathbf{D}$ a conventional, very high penalization. Thus we have to deal only with positive semidefinite matrices. If we measure the difference between two configurations $\mathbf{C}$ and $\mathbf{C}'$ (the first one of higher dimension than the second) by the Frobenius norm of their difference

$$\|\mathbf{C} - \mathbf{C}'\|_F = \sqrt{\sum_{ij}^N c_{ij} - c'_{ij}}$$

we have the following theorem (adapted from [HKC83])

Theorem 4. *If $\mathbf{C}$ has rank n the matrix $\mathbf{C}'$ of rank $m \leq n$ that minimizes*

$$\|\mathbf{C} - \mathbf{C}'\|_F$$

is obtained as follows

1. *compute $\mathbf{C}$ metric matrix $\mathbf{M}$;*
2. *decompose $\mathbf{M}$ as $\mathbf{M} = \mathbf{Y}^T \mathbf{\Lambda} \mathbf{Y}$, where $\mathbf{Y}$ is unitary and $\mathbf{\Lambda}$ is $\mathbf{Diag}(\lambda_1, \dots, \lambda_N)$ with $\lambda_1, \dots, \lambda_N$ the eigenvalues of $\mathbf{M}$ in decreasing order;*
3. *finally, $\mathbf{C}' = \mathbf{Diag}(\lambda_1^{\frac{1}{2}}, \dots, \lambda_m^{\frac{1}{2}}, 0, \dots, 0) \mathbf{Y}$*

Moreover, we observe that the minimum of $\|\mathbf{C} - \mathbf{C}'\|_F$ so attained is

$$\sqrt{\sum_{i=m+1}^N \lambda_i}$$

Therefore, this is (with $m = 3$) a good candidate as a penalization term. Since we would like it to be invariant to scale changes, we normalized it to obtain

$$k \frac{\sqrt{\sum_{i=m+1}^N \lambda_i}}{\sqrt{\sum_{i=1}^N \lambda_i}}$$

where k is a weighting factor to be described later on.

6.3 Genetic operators

The EA can be further specified with the definition of genetic operators. We defined and experimented with a number of different recombination operators and we studied the stability of schemata, the feasibility of solutions produced by each of them and their behavior w.r.t. the repair algorithm. The following definitions are a generalization of the ones in [Vos91].

Definition 4 A schema is a subset of the solution space.

Definition 5 A schema H is said to be stable under some genetic operator G if whenever G is applied to individuals all belonging to H then every generated offspring belongs to H .

Every operator is supposed to have as input one or two zero diagonal symmetric matrices (parents) with positive entries and outputs, as it is easy to check, a matrix with the same properties.

The first genetic operator is the customary *uniform crossover*, i.e. let $\mathbf{D}'$ and $\mathbf{D}''$ be parents of $\mathbf{D}$, then for each i, j independently

$$P(d_{ij} = d'_{ij}) = \frac{1}{2} \quad P(d_{ij} = d''_{ij}) = \frac{1}{2}$$

It is easy to show that every coordinate hyperplane in matrix space is stable under this operator. The resulting matrix is not guaranteed to have a positive semidefinite metric matrix $\mathbf{M}$, so that the new individual has to be rejected or penalized, depending on constraint handling strategy. Moreover, even when $\mathbf{M}$ is positive semidefinite, it can have full rank also when parents have rank 3, so that the upper bound on distortion in theorem 3 can be rewritten as

$$\left| \frac{d_{ij}^2 - d_{ij}''^2}{d_{ij}^2} \right| \leq O(\sqrt{N \log N})$$

This can be better evaluated considering that, in general, in a feasible structure $\mathbf{D}$ $1 \leq d_{ij} \leq N$ and in a maximally compact structure (and low energy structures are usually very compact) $1 \leq d_{ij} \leq O(N^{\frac{1}{3}})$. We found experimentally that using a similar operator but with probability biased toward one of the parents (19/20) positive semidefiniteness is often preserved and distortion is lower. We call this biased operator *graft crossover*. See table 1 for an experimental analysis of this operator.

The third operator (called *arithmetic crossover*) is a convex combination of element-wise squared matrices i.e.

$$d_{ij}^2 = \alpha d_{ij}'^2 + (1 - \alpha) d_{ij}''^2$$

with α uniformly distributed between 0 and 1. It is straightforward to show that every convex set in matrix space is stable under this operator, so that it has a very rich set of stable schemata, that includes, for example, coordinate hyperplanes and general hyperplanes. As to what concerns feasibility, we have the following

Theorem 5. [HKC83] *The set of all matrices of squared distances is convex in zero-diagonal matrix space. Furthermore, the convex combination of two matrices of squared distances embeddable in k and l dimensions respectively is embeddable in at most $k + l$ dimensions.*

This means that when adopting the repair strategy we are guaranteed that the generated individual has a positive semidefinite metric matrix of rank at most 6, since parents are guaranteed to be feasible. This turns out in a much lower distortion, as we can see specializing theorem 3 to this case as follows

$$\left| \frac{d_{ij}^2 - d_{ij}^{\prime 2}}{d_{ij}^2} \right| \leq O(\sqrt{\log N})$$

A fourth operator (*block crossover*) tries to emulate the effects of multi-point crossover applied to internal coordinates, as in previous approaches cited before. It is akin to uniform crossover with submatrices instead of elements as basic entities. It works as follows

1. The set of aminoacids is partitioned in intervals $I_1, \dots, I_k$, according to their order in the aminoacidic sequence;
2. Both parents' distance matrices ($\mathbf{D}'$ and $\mathbf{D}''$) are partitioned in submatrices

$$\mathbf{D}'_{ij} = \{d'_{hk}\}_{h \in I_i, k \in I_j} \quad \mathbf{D}''_{ij} = \{d''_{hk}\}_{h \in I_i, k \in I_j}$$

3. For each i, j choose a parent $p(i, j)$ with probability $1/2$;
4. Define a new solution as

$$d_{hk} = \{d_{hk}^{p(ij)}\}_{h \in I_i, k \in I_j}$$

The analysis of this operator is basically the same as the one for uniform crossover.

In table 1 the four crossover operators are analyzed experimentally on a sample population of 27 residue long individuals and size 1000, generated according to the strategy explained in 6.3. For 1000 individuals generated by crossover we report the ratio of individuals with positive semidefinite metric matrix, the ratio of individuals with energy lower, intermediate or higher than the two parents.

crossover	pos. sem.	lower	interm.	higher
arithmetic	100.0	23.3	49.5	27.2
block	45.4	7.3	19.2	73.5
graft	65.6	6.3	17.6	76.1
uniform	33.1	1.2	5.3	93.5

Table 1. Statistical analysis of crossover operators

The mutation operator is a classic creep operator that randomly chooses a variable and modifies it with the addition of a small gaussianly distributed

amount ε , preserving distance symmetry. The positivity of entries is preserved by suitable truncation of the probability distribution. Since the corresponding creep in Gram matrix eigenvalues is $O(\varepsilon)$ [Van92], mutation can be tuned so that feasibility is almost preserved. Small negative eigenvalues are rounded to zero.

Further details of the EA can be summarized as follows:

- we use a steady-state algorithm (offsprings are inserted in the population as soon as they are created);
- reproduction and replacement candidates are selected out of a small pool (*tournament selection*), according to fitness and diversity criteria (individual less fit and more similar to a new offspring are more likely to be replaced).
- each individual in the initial population is generated in the following way: the first aminoacid is at the origin; the position of the i th aminoacid is chosen uniformly at random on the unit sphere centered around the $(i - 1)$ th aminoacid.

7 Implementation issues

The simulation program has been coded in C++, using the mathematical library Lapack++ and the genetic algorithm library GAlib. A graphical front end for simulation control and basic data analysis has been written in Perl and Tcl/Tk and using the visualization program Rasmol. Simulation have been run on a IBM-compatible with Pentium processor under Linux operating system, but a porting to a four processor Silicon Graphics Origin 2000 server is under way.

8 Simulation results

We compare our results to those found in [PPG95], but the reader should be warned that the comparison is somewhat unfair, since in that paper a lattice model is used, so that a residue neighborhood can accommodate only 6 residues, instead of 12 [CS92] as in our continuous model. Moreover, in the same paper the energy is the opposite of the number of hydrophobic residue pairs at unit distance, so we tried to evaluate our structures using the same criterion, but introducing a little tolerance.

In table 2 we summarize our results on three test cases taken from [PPG95] using the *repair strategy*, comparing different crossover operators and tournament sizes. Column C1 reports the number of hydrophobic pairs at distance 1 ± 0.2 , C2 at distance 1 ± 0.1 . The corresponding results in [PPG95] are 15 for P27-4, 11 for P27-6 and 13 for P27-7.

In figure 1 the best structure obtained for P27-4 is shown. The compact hydrophobic core is very protein-like. The simulations ran with a population of 1000 individuals and the algorithm stopped after the generation of 200,000

		Tournament size 10			Tournament size 4		
sequence	crossover	C1	C2	energy	C1	C2	energy
P27-4	arithmetic	20	20	24.211	18	15	19.655
	block	35	28	-23.438	31	27	-19.484
	graft	35	31	-23.051	36	30	-28.068
	uniform	22	19	22.071	25	22	5.267
P27-6	arithmetic	10	7	16.592	10	8	15.730
	block	17	15	-4.172	17	15	-4.511
	graft	13	10	0.961	16	11	0.459
	uniform	12	8	6.580	13	9	8.270
P27-7	arithmetic	20	15	-2.190	16	12	12.719
	block	25	21	-5.368	21	19	-4.231
	graft	23	19	-1.343	21	17	2.959
	uniform	22	19	-4.640	15	12	13.631

Table 2. Simulation results for “repair” strategy

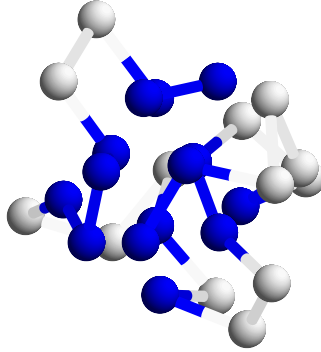


Fig. 1. Best structure for P27-4

offsprings, taking about 21 minutes. The probability of applying the crossover and mutation operators were set to 0.9 and 0.5 respectively.

Simulation with the *penalize strategy* gave unsatisfactory results. We tried different penalization strategies, with fixed or time varying weight factor. No run stopped producing a feasible solution. This can be explained noting that all genetic operators described tend to increase the rank of the metric matrix of new individuals.

9 Discussion and further work

The ultimate test for any protein folding prediction algorithm is the prediction of experimental structures of biological sequences. Of course the EA described in the present work is not ready for such a task, at least because the very simple model prevents accurate predictions. Anyway it compares favorably with previous approaches and can be straightforwardly extended to more complex and realistic models, something that lattice model oriented algorithms can't. The experimental analysis of this algorithm is clearly in its preliminary stage and we plan to test longer sequences as soon as possible. Moreover we would like to clarify the relative strengths and weaknesses of different crossover operators. A first distinction can be made between *arithmetic crossover* and the others, since this operator has some nice property w.r.t rank constraints but doesn't explore thoroughly the search space. This observation should be made more precise and supported with analytic or experimental evidence.

Acknowledgments

This work has been partly supported by MURST project "Efficienza di Algoritmi e Progetto di Sistemi Informativi" and by CNR grant 97.02399.CT12.

References

- [AHSJ61] C. B. Anfinsen, E. Haber, M. Sela, and F. H. White Jr. The kinetics of formation of native ribonuclease during oxidation of the reduced polypeptide chain. In *Proceedings of the National Academy of Sciences of the U. S. A.*, volume 47, pages 1309–1314, 1961.
- [BBM93] David Beasley, David R. Bull, and Ralph R. Martin. An overview of genetic algorithms: Part 2, research topics. *University Computing*, 15(4):170–181, 1993.
- [BBM94] D. Beasley, D. R. Bull, and R. R. Martin. Complexity reduction using expansive coding. In T. C. Fogarty, editor, *Evolutionary computing: aish workshop: selected papers*, page 304, Leeds, UK, april 1994.
- [BS93] Th. Bäck and Hans-Paul Schwefel. An overview of evolutionary algorithms for paramater optimization. *Evolutionary Computation*, 1(1):1–23, 1993.
- [CS92] J. H. Conway and N. J. A. Sloane. *Sphere packings, lattices and groups*. Number 290 in Grundlehren der mathematischen Wissenschaften. A series of comprehensive studies in mathematics. Springer-Verlag, New York, second edition, 1992.
- [DA94] T. Dandekar and P. Argos. Folding the main chain of small proteins with the genetic algorithm. *Journal of Molecular Biology*, 236:844–861, 1994.
- [De 75] Kenneth De Jong. *An analysis of the behaviour of a class of genetic adaptive systems*. PhD thesis, University of Michigan, 1975.
- [Dil85] Ken A. Dill. Dominant forces in protein folding. *Biochemistry*, 24:1501, 1985.

- [FOW66] Lawrence J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial intelligence through simulated evolution*. Wiley, New York, 1966.
- [Gol90] D. E. Goldberg, editor. *Genetic Algorithm in search, optimization and machine learning*. Addison-Wesley, 1990.
- [HKC83] T. F. Havel, I. D. Kuntz, and G. M. Crippen. The theory and practice of distance geometry. *Bulletin of Mathematical Biology*, 45(5):665–720, 1983.
- [Hoe63] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, (58):13–30, 1963.
- [Hol75] John H. Holland, editor. *Adaptation in Natural and Artificial Systems*. MIT Press, 1975.
- [HS95] Frank Herrmann and Sándor Suhai. Energy minimization of peptide analogues using genetic algorithm. *Journal of Computational Chemistry*, 16(11):1434–1444, 1995.
- [LLR95] Nathan Linial, Eran London, and Yuri Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995.
- [MS96] Zbigniew Michalewicz and Marc Schoenauer. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary computation*, 4(1):1–32, 1996.
- [Neu93] Arnold Neumaier. Molecular modeling of proteins and mathematical prediction of protein structure. *SIAM Rev.*, pages 407–460., 1993.
- [NS77] G. Nemethy and H. Scheraga. Protein folding. *Quarterly reviews in Biophysics*, 10:239–352, 1977.
- [PPG95] A. Patton, W. Punch, and E. Goodman. A standard ga approach to native protein conformation prediction. In L. Eshelman, editor, *Proc. Sixth Int. Conf. Gen. Algo.*, volume 574. Morgan Kaufmann, 1995.
- [SK93] Steffen Schulze-Kremer. Genetic algorithms for protein tertiary structure prediction. In P. B. Brazdil, editor, *Machine Learning: ECML-93*. Springer-Verlag, 1993.
- [SK95] Steffen Schulze-Kremer. Genetic algorithms and protein folding. <http://www.techfak.uni-bielefeld.de/bcd/Curric/ProtEn/contents.html>, June 1995.
- [Tre96] Luca Trevisan. When hamming meets euclid: The approximability of geometric TSP and MST. In *Proceedings of the 28th Symposium on the Theory of Computing*, 1996.
- [UM93] R. Unger and J. Moult. Genetic algorithms for protein folding simulations. *Journal of Molecular Biology*, 231:75–81, 1993.
- [Van92] Charles Van Loan. A survey of matrix computations. In A. H. Rinnooy Kan, J. K. Lenstra, and E. G. Coffman, editors, *Handbooks in operations research and management science*, volume 3: computing, chapter 6, pages 247–321. Elsevier Science Publisher, 1992.
- [Vos91] Michael D. Vose. Generalizing the notion of schema in genetic algorithms. *Artificial Intelligence*, 50:385–396, 1991.

Telephone Network Traffic Overloading Diagnosis and Evolutionary Computation Techniques

Isabelle Servet¹, Louise Travé-Massuyès¹ and Daniel Stern²

¹ LAAS/CNRS 7, avenue Colonel Roche F-31077 Toulouse Cedex France

² CNET 38-40 ,rue du Général Leclerc F-92131 Issy-les-Moulineaux France

Abstract. Traffic supervision in telephone networks is a task which needs to determine streams responsible for call losses in a given network by comparing their traffic values to nominal values. However, stream traffic values are not measured by the on-line data acquisition system and, hence, have to be computed. We perform this computation by inverting an approximate knowledge-based model of stream propagation in circuit-switched networks. This inversion is computed thanks to three evolutionary computation techniques (multiple restart hill-climbing, population-based incremental learning and genetic algorithms) for which both a binary version and a real variant have been experimented with several fitness measures. The final results first point out how the fitness measure choice can impact on their quality. They also show that, in this case, real variants of the algorithms give significantly better results than binary ones.

1 Introduction

Independently of the network communication organs characteristics, the telephone traffic flow quality can be affected by some streams overloads.

This traffic modification can be linked to:

- a daily or seasonal change: some switches offered traffic increase whereas the others decrease. This is, for instance, the case of switches connected to tourist sites during holidays.
- a whole overload; this can arrive on special occasions like Christmas or New Year's day.
- an unexpected event such as a televised or radiotelephone game, a natural, rail or air disaster or else a political event.

Therefore, to limit the consequences connected to these unforeseen events, it is necessary to make a real control of circuit-switched networks, such as the French long distance telephone network.

The real time network control objective is to avoid network degradations and, this, allowing the network supervision and the implementation of traffic control actions; these actions will permit, in the case of an overload, that most of the

call requests can reach their destination, using the maximal number of available resources in the network.

The real time network control can be decomposed in the following way:

1. network state and working conditions supervision,
2. supervision data collection and analysis,
3. network abnormal circumstances detection,
4. disturbances causes diagnosis,
5. corrective actions over the network or the traffic.

In this paper, we will only consider the points 3 and 4 of this decomposition.

Due to the increasing size of telephone networks and to their high connectivity (several organs are shared by several streams), the task of networks managers becomes more and more complex. Determining which streams are responsible for an overloaded situation is difficult, especially as the disturbances may propagate within the network in a very short time frame.

Real-time network supervision does not require accurate values (approximate values are sufficient enough to incriminate the streams responsible for overloads) but it needs to compute them rapidly.

Thus, the stream propagation computation cannot be done using a call by call simulation program because of the extremely long computer running time it requires. This is the reason why we have established an approximate stream propagation model [STMS96b].

Then, as our goal is to know the values of the offered traffic corresponding to each stream, given the traffic losses and the offered traffic at each network element which are the only measures available, we have to *invert* the previous model thanks to optimization methods.

Our model is indeed an highly non linear and non differentiable function so that neither a direct inversion nor classical optimization methods are appropriate.

However, evolutionary computation techniques are the subject of many researches, given their high potential for solving complex optimization problems. In the third part of this paper, three of these methods are presented :

- *multiple restart hill-climbing*, an heuristic and, then, very simple method,
- *population based incremental learning* which can be seen as an abstraction of genetic algorithms,
- *a standard genetic algorithm*, a well-known method to solve complex optimization problems.

These methods have been chosen because of their simplicity and, then, rapidity. They initially have been developed with binary variables but a real variant of these methods is proposed too.

Comparative results of the model inversion are given in the third part of this paper.

2 Stream propagation model

The simplest approach for network analysis is the one-moment method where offered traffic is only characterized by its mean. The model briefly presented here uses this method. It is dedicated to compute the whole traffic losses and offered traffic for each communication organ, for each stream and the lost and offered traffic for each stream at each network element (i.e. each communication organ), all this given the network topology and the offered traffic to each stream.

This model has been developed under classical assumptions of single moment traffic modelling methods [Pas84, GBG84] :

- **A1** Call arrival is a Poisson process.
- **A2** Call holding time has a negative exponential distribution.
- **A3** Blocking probabilities are statistically independent.
- **A4** The network is in statistical equilibrium.
- **A5** Call arrival (node-originated plus overflow) on any element is a Poisson process.

2.1 Telephone network structure

In the first hand, the French long distance network can easily be seen as a graph with:

- nodes, representing **commutation centres** (or *switches*) whose capacity represents the maximum number of simultaneous carried calls,
- arcs, that represent the **circuit groups** between two switches; their capacity (i.e. their number of circuits) is equal to the maximum number of calls that can be carried simultaneously.

Then, the telephone network carries some **streams**, each stream corresponding to a set of calls going from a switch (called the origin node) to another one, their destination node; a stream is also characterized by its *offered traffic* value (the average number of call arrivals during a period corresponding to the average call holding time) and its *routing table* that provides the possible ways to reach the destination node.

2.2 Model

This model is based on the concept of **blocking** network elements and the search of such elements in the network.

Definition 1 *A network element (NE): exchange or circuit group is said to be blocking when, according to the network structure, it is likely to experiment traffic loss.*

It can be divided in 3 main stages [STMS96a]:

- *Blocking organs search* which is performed thanks to 2 qualitative rules: the overdimensioned capacities rule (independent of the offered traffic values) and the loss rate rule which is performed using **Erlang’s formula** given in the definition below.

Definition 2 *Consider an organ whose capacity is N and which routes a traffic stream of intensity A . Then, the probability that the N circuits are busy is: $E[N, A] = \frac{A^N/N!}{1+\dots+A^N/N!}$*

- *Blocking NEs ordering*. This is especially important when overflow from a circuit group to an other is allowed, the offered traffic on the latter depending on the amount of lost traffic on the former.
- *Traffic loss computation* for each blocking NE, which is computed thanks to a *resource availability backpropagation* algorithm [STMS96a]

3 Model inversion

To supervise the French long distance telephone network, we propose to determine streams responsible for call losses by comparing their traffic values to nominal values. However, stream traffic values are not measured by the on-line data acquisition system and, hence, need to be computed in real time. This is done by inverting the model of Part 2 thanks to evolutionary techniques, where genes values are given by the corresponding stream traffic values ; in binary versions of the presented evolutionary algorithms, Gray code is used to translate real traffic values into binary ones : using Gray code representations lead to better results than using classical binary representation[Hol71].

The method consists in calling iteratively the stream propagation model with stream traffic values updated by one of the considered evolutionary techniques and in using a fitness measure to compute the distance between the observed values and computed values. First, the observed (measured) quantities can be either the number of calls offered to each organ or the stream carried traffic value, which are the only on-line available measures in the French long distance telephone network.

Then, the distance between observed and computed values can be chosen among:

- the Euclidean Distance (ED), defined by

$$\sqrt{\sum_{i=1}^n (\text{observed\_value}_i - \text{computed\_value}_i)^2}$$

- the χ^2 distance (χ^2), defined by

$$\sqrt{\sum_{i=1}^n \frac{(\text{observed\_value}_i - \text{computed\_value}_i)^2}{\text{observed\_value}_i}}$$

- the Infinite Norm (IN), defined by

$$\max_{i=1\dots n} |\text{observed\_value}_i - \text{computed\_value}_i|$$

where i is the number of considered quantities.

These distances have been chosen for their simplicity.

The 8 methods described in this paper have been tried successively with 6 fitness measures (the 3 distances being applied to the number of offered calls (OC) as well as the stream carried traffic (CT)).

In a perspective of real-time traffic supervision, each of these methods has been computed with only 500 iterations and they last about 10 minutes on a Sparc Station 5.

3.1 Evolutionary algorithms

Evolutionary computation is an umbrella term used to describe computer-based problem solving systems which use models of evolutionary processes as key elements in their design and implementation [HB94].

Their common base is the simulation of the evolution of *individuals* via processes of *selection*, *mutation* or *reproduction* which depend on the performance, computed thanks to a *fitness measure*, of the individuals.

Hence, before an evolutionary algorithm can be run, a suitable representation for each structure must be determined. It is assumed [BBM93a] that a potential solution to a problem may be represented as a set of parameters, known as *genes* that are joined together to form a string of values, often called a *chromosome*.

Besides, these algorithms are often used for function optimization[Bal95], their effectiveness strongly depending on the particularities of the function.

Multiple restart hill-climbing (MRH)

Binary version

This iterative method, although very simple, sometimes leads to better results than genetic algorithms [Bal94]. Its simplest algorithm is:

```

V ← randomly generated chromosome
BEST ← V
Loop NB_ITERATIONS
  MUT_POSITION ← random bit position
  N ← mutation of V in MUT_POSITION
  If N is better than BEST
    Then BEST, V ← N

```

What follows considers a variant of this algorithm which consists in maintaining a list of mutation position bits that have been tried without improving the solution according to the fitness measure. These positions are not attempted

again until a better solution is found.

When a better solution is found, this list is emptied. If the list becomes as large as the solution vector size, no mutation of V can improve the solution and the algorithm is restarted at a new random location with an empty mutation position list.

Real-coded variant

First, the chromosome V , which is randomly generated, is a real vector. Hence, when optimizing a function, each component of this chromosome represents the value of the corresponding variable.

Then, $MUT\_POSITION$ corresponds to the range of the variable which has to undergo a mutation.

Therefore, a mutation of the i^{th} component $V[i]$ of the chromosome V consists, like in genetic algorithms real-coded variants [BBM93b], in adding to $V[i]$ a small random real (for instance a real in $[-V[i]/10, V[i]/10]$).

Population-based incremental learning (PBIL)

Binary version

Contrary to the previous method, this one does not consider a single chromosome but a set of nb chromosomes called a *generation*.

The object of this method is to create a real-valued *probability vector* P which is dedicated to reveal high quality chromosomes[Bal95]. Hence, P is updated at each generation and the chromosomes of each generation are created according to its values: each component $V[i]$ of each chromosome V is determined as follow :

Given $random_i$ a random value in $[0,1]$,

If $random_i > P[i]$ then $V[i] \leftarrow 0$ else $V[i] \leftarrow 1$

Therefore, $P[i]$ represents the probability to have '1' at this position in the best chromosome ; hence, $P[i]$ is, first,taken equal to 0.5 and P evolution, from a generation to an other, is computed thanks to competitive learning mechanisms[Bal94].

Real-coded variant

First, we assume that each gene (i.e. each variable in the case of function optimization) has a continuous and bounded definition set. Hence, we consider that each component $V[i]$ of a chromosome V belongs to an interval $[Low_i, Up_i]$.

Then, each component $P[i]$ of P gives the probability of the i^{th} variable to be greater than $\frac{Low_i + Up_i}{2}$.

The probability vector updating is similar to P updating in the binary version but a step for intervals $[Low_i, Up_i]$ updating is added: when, for a given iteration, the i^{th} component of P is such that:

- $P[i] \geq 0.9$, the i^{th} component of the best chromosome is assumed to be greater than $(Low_i + Up_i)/2$. Then, Low_i is updated to $(Low_i + Up_i)/2$ and $P[i]$ is reinitialized at 0.5.

- $P[i] \leq 0.1$, the i^{th} component of the best chromosome is assumed to be lower than $(Low_i + Up_i)/2$. Then, Up_i is updated to $(Low_i + Up_i)/2$ and $P[i]$ is reinitialized at 0.5.

P reinitialization is due to the fact that the change of a bound of the interval $[Low_i, Up_i]$ comes to start again the algorithm with a new definition set.

A genetic algorithm (GA)

Binary version

In this method, an offspring can be the result of gene mutation, as in the 2 previous methods, but also the result of 2 chromosomes crossover. We have chosen to apply 3 crossover techniques :

- *1-point crossover* that takes two individuals, cuts their chromosome strings at some random position to produce two head and two tail segments and, then, swaps over the tails to produce two full length chromosomes,
- *2-point crossover* that regards chromosome strings as loops by joining the ends together and, then, exchanges a randomly segment from one loop with that from another loop [BBM93b];
- *uniform crossover* that generates a random crossover mask: where there is a 1 in this mask, the gene is copied from the first parent, and where there is a 0, the gene is copied from the second parent [BBM93b].

The genetic algorithm description [HB94, BBM93a, Bal94] is:

```

Generate POPULATION_SIZE random chromosomes
Compute fitness of each individual of the population
Loop GENERATIONS
  Loop POPULATION_SIZE/2
    • Select probabilistically 2 chromosomes  $V_1$  and  $V_2$ 
    • Recombine (thanks to crossover)  $V_1$  and  $V_2$  to give  $C_1$  and  $C_2$ 
    • Perform mutation according to MUT_RATE of  $C_1$  and  $C_2$ 
    • Compute fitness of  $C_1$  and  $C_2$ 
    • Insert  $C_1$  and  $C_2$  in the new generation
  WORST  $\leftarrow$  worst vector of the new generation
  BEST  $\leftarrow$  best vector of the old generation
  Replace WORST by BEST in the new generation

```

The choice of the parent vectors V_1 and V_2 is done thanks to tournament selection. Indeed, it is often used in conjunction with noisy or imperfect fitness functions [MG95] which comes close to our problem which is to inverse a model that gives *approached* values.

Real-coded variant

As in the real-coded variant of MRH, the only changes in real-coded GAs concern the way the chromosomes operators are applied.

The mutation operator has already been described in the section concerning the real-coded variant of MRH. Moreover, many real-coded crossover operators can be envisaged. However, to keep the idea of some random choice, we apply a *blend-crossover* (BLX- α) [ES92] which is able to create 2 offsprings.

Let us consider 2 parents p_1 and p_2 and their 2 offsprings c_1 and c_2 . Then, for $i=1,2$, each component $c_i[j]$ of c_i is a random value that belongs, if we assume $p_2[j] > p_1[j]$, to the interval

$$[p_1[j] - \alpha(p_2[j] - p_1[j]), p_2[j] + \alpha(p_2[j] - p_1[j])]$$

where α is a user specified GA parameter. We have chosen :

POPULATION_SIZE = 50, MUT_RATE = 0.01 and $\alpha = 0.5$.

3.2 Numerical results

We have tested each of the inversion methods on a set of 34 configurations that regroup all the particularities of the stream propagation model. For each of these 34 tests, we have computed the maximum of the

$$\frac{|\text{real stream traffic value} - \text{computed stream traffic value}|}{\text{real stream traffic value}}$$

Table 1 gives the average of these results.

	(IN)		(ED)		(χ^2)	
	(CT)	(OC)	(CT)	(OC)	(CT)	(OC)
Bin. MRH	58%	61%	63%	58%	64%	55%
Real MRH	22%	26%	19 %	12%	19%	10%
Bin. PBIL	59%	61%	59%	59%	59%	59%
Real PBIL	282%	39%	255%	37%	236%	19%
1 pt GA	62%	63%	61%	64%	61%	62%
2 pt GA	61%	64%	62%	63%	62%	63%
Uni. GA	60%	63%	62%	62%	62%	60%
Real GA	13%	22%	14%	15%	13%	11%

Table 1. Mean Results

First, we can notice that, in the case of our application, the choice of the crossover operator does not influence at all the results of binary evolutionary

algorithms.

Then, Table 1 clearly shows that, in most of the cases, the fittest techniques are the real ones.

Let us make a statistical analysis of the results of each of the 8 proposed techniques.

	Mean	Root-mean square	Variation factor
Bin. MRH	59.8	2.77	4.63
Bin. PBIL	59.33	0.75	1.26
1pt GA	62.17	1.07	1.72
2pt GA	62.5	0.96	1.47
Uni. GA	61.5	1.25	1.82
Real MRH	18	5.51	30.6
Real PBIL	144.67	113.34	78.35
Real GA	14.67	3.49	23.83

where *Mean* is the arithmetical mean of x_{ij} , values computed for each of the distance pair (*norm_i*, *indicator_i*)

Root-mean square is defined by: $\sqrt{\frac{\sum |x_{ij} - \text{Mean}|^2}{n}}$
(*n* being the number of considered distances, i.e. *n* = 3*2)

Variation factor is defined by $100 * \frac{\text{Root-mean square}}{\text{Mean}}$.

Then, we can notice, according to the values of the variation factors, that, whereas the choice of the fitness measure does not really impact on the binary versions of the studied evolutionary computation techniques, it is obviously very important for their real variants.

Besides, looking at the real variants of the algorithms, the norm choice is more important when the indicator is the number of offered calls than when it is the value of the carried traffics, as shown in the following table:

	(CT)	(OC)
Real MRH	7.07	44.49
Real PBIL	7.32	28.37
Real GA	3.53	29.13

Table 2. Variation factor and indicator choice

Moreover, it is the fitness measure defined by the χ^2 distance applied to the number of offered calls that gives the better results and, this, whatever the considered evolutionary algorithm.

Finally, concerning the choice of the model inversion method, the most promising results are given by either the *real variant of MRH* or the *real variant of GA*.

4 Conclusion

First of all, although the inverted model gives only approximate values and despite its complexity, in term of non-linearities and non differentiability (due to the blocking NEs ordering and to the resource availability backpropagation phenomenon), the results computed thanks to evolutionary computation techniques are quite good.

Then, an important conclusion of our study is the obvious influence of distance choice on evolutionary computation techniques performance. Unfortunately, there is no methodology so far, excepted empirical methods of course, to say a priori whether a distance is able to give better results than an other one or not.

Besides, our application clearly shows that, real-coded variants, that are closer to human reasoning, can outperform binary evolutionary computation techniques.

However, in a perspective of real-time traffic supervision, we cannot afford a larger population because of its long running time. To solve this problem, we envisage to use an explicit parallelism to improve the speed of both population-based incremental learning and genetic algorithms.

5 ACKNOWLEDGEMENTS

This research was funded by the Centre National d'Études des Télécommunications, contract 93 1B 142, project N°513.

References

- [Bal94] S. Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical Report CMU-CS-94-163, School of computer Science, Carnegie Mellon University, Pittsburgh, 1994. Disponible par ftp anonyme sur le serveur *reports.adm.cs.edu*.
- [Bal95] S. Baluja. An empirical comparison of seven iterative and evolutionary function optimization heuristics. Technical Report CMU-CS-95-193, School of computer Science, Carnegie Mellon University, Pittsburgh, 1995. Disponible par ftp anonyme sur le serveur *reports.adm.cs.edu*.
- [BBM93a] D. Beasley, D.R. Bull, and R.R. Martin. An overview of genetic algorithms : Part 1, fundamentals. *University Computing*, 15(2):58–69, 1993.
- [BBM93b] D. Beasley, D.R. Bull, and R.R. Martin. An overview of genetic algorithms : Part 2, research topics. *University Computing*, 15(4):170–181, 1993.

- [ES92] L.J. Eshelman and J.D. Schaffer. Real-coded genetic algorithms and interval schemata. In *Foundations of Genetic Algorithms 2*, 1992.
- [GBG84] F. Le Gall, J. Bernussou, and J.M. Garcia. A one-moment model for telephone networks with dependance on link blocking probabilities. In *Performance (E. Gelenbe, Ed.)*, pages 449–458, 1984.
- [HB94] J. Heiktocker and D. Beasley. The hitch-hiker’s guide to evolutionary computation: A list of frequently asked question. available by anonymous ftp at rtfm.mit.edu, 1994.
- [Hol71] R.B. Hollstein. *Artificial genetic adaptation in computer control systems*. PhD thesis, University of Michigan, 1971.
- [MG95] B.L. Miller and D.E. Goldberg. Genetic algorithms, tournament selection and the effects of noise. Technical Report 95006, IlliGAL, Juillet 1995.
- [Pas84] A. Passeron. Notions élémentaires sur le trafic téléphonique. Technical Report DE/ATR/57.84, CNET, 1984.
- [STMS96a] I. Servet, L. Travé-Massuyès, and D. Stern. Traffic supervision based on a one-moment model of telephone networks built from qualitative knowledge. In *IMACS/IEEE CESA ’96*, 1996.
- [STMS96b] I. Servet, L. Travé-Massuyès, and D. Stern. Traffic supervision in telephone networks and qualitative modelling. *Annals of telecommunications*, 51(9-10):483–492, 1996.

Genetic Algorithms for Genetic Mapping

Christine Gaspin and Thomas Schiex
{gaspin,tschiex}@toulouse.inra.fr

Institut National de la Recherche Agronomique, Biometry and AI Dept.
Chemin de Borde Rouge BP 27, Castanet-Tolosan 31326 Cedex, France

Abstract. Constructing genetic maps is a prerequisite for most in-depth genetic studies of an organism. The problem of constructing reliable genetic maps for any organism can be considered as a complex optimization problem with both discrete and continuous parameters. This paper shows how genetic algorithms can be used to tackle this problem on simple pedigree. The approach is embodied in an hybrid algorithm that relies on the statistical optimization algorithm EM to handle the continuous variables while genetic algorithms handle the discrete side. The efficiency of the approach lies critically in the introduction of greedy local search in the fitness evaluation of the genetic algorithm, using a neighborhood structure which has been inspired by an analogy between the marker ordering problem and a variant of the famous traveling salesman problem. This shows how genetic algorithms can easily benefit from existing efficient neighborhood structures developed for local search algorithms. The resulting program, called CAR_TAGENE, has been applied both to real data, from a small parasitoid wasp, and simulated data. In both cases, it compares quite favorably to existing packages.

1 Introduction to Genetic Mapping

The aim of genetic mapping is to locate genes, and more generally genetic markers at *loci* (positions) on the chromosomes. The specific content of a locus in a given chromosome is termed the *allele* (in a computer analogy, a locus is a memory location and the allele is the content of the memory location). Given a locus on a chromosome, individuals of *diploid species* have two alleles, one on each of the corresponding chromosomes contributed by the parents. An individual is said to be *homozygous* at a locus if the two alleles are identical at this locus, else the individual is said to be *heterozygous*.

Each chromosome contributed by each parent is built, during the meiosis, by using sections of either member of each pair of chromosomes of the parent. Changes in the chromosome used are called *crossovers*. At a given locus, there is a 50% chance of having either one of the parental allele. However, the “closer” two loci are on the chromosome, the higher the probability that the two alleles on this chromosome will appear together on the contributed chromosome. Two loci (or genetic markers) are thus said to be *linked* if the parental allele combinations are preserved more often than would be expected by random choice. When a parental allele combination between two loci is *not* preserved, a recombination

event is said to have occurred (this corresponds to an odd number of crossovers between the two loci).

The degree of linkage, or genetic distance, between two loci is a function of the frequency of recombinations. The measurement of genetic distance is expressed in Morgan (or more usually cM for centiMorgan) and is defined as the expected number of crossovers between the two loci on the chromosome.

The idea used to build the genetic map of an organism is to observe the alleles of several loci of interest in a large number of meiosis and to build a map that, in a precise sense, best explains what is observed. In the simplest case, this is done using the so-called *backcross* pedigree (see Fig. 1) which consists in breeding an individual α which is homozygous at all loci of interest with another individual ω which is heterozygous at all loci of interest. After the meiosis, α will contribute a chromosome where all loci have the same allele and therefore are not informative on the recombinations. But ω will give a chromosome where each recombination event changes the type of the allele used in the chromosome. In the figure, if we suppose that the order ABCDE is the correct order, the string 01100 indicates two recombinations, one between A and B, another between C and D. This type of measure is done on a large number of descendants of α and ω .

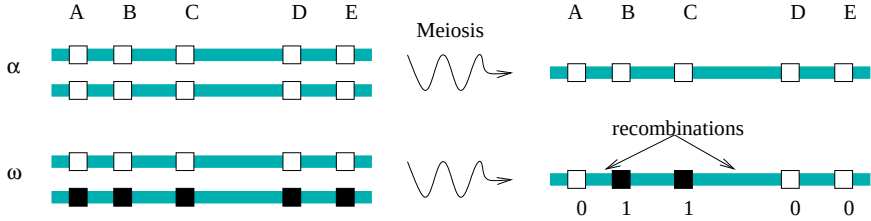


Fig. 1. An exemple of backcross for one individual and 5 markers (noted A,B,C,D,E).

We consider N markers, and K descendants. For a given loci ℓ and a given descendant i , the allele contributed by the heterozygous parent, which can be either 0 or 1 will be denoted X_{ℓ}^i . This defines the observations and we end up with a data set which is defined by an array of bits X_{ℓ}^i . A map is defined by a linear order of the loci *i.e.*, a one-to-one mapping ϕ from the set $\{1, \dots, N\}$ to $\{1, \dots, N\}$ and a distance between adjacent loci or more precisely, a probability of recombination between two adjacent loci $\phi(\ell)$ and $\phi(\ell + 1)$, denoted $\hat{\theta}_{\ell, \ell+1}$. See Fig. 2, for an exemple of a map.

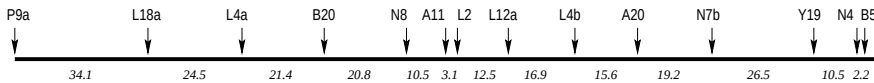


Fig. 2. A genetic map exemple (parasitoid wasp *Trichogramma brassicae*).

The genetic mapping problem is therefore to find an order of the markers and probabilities of recombination between pairs of markers that best explain the available data set. The task is difficult in several aspects: it is not always obvious to say when a map best explains the data set (criteria to optimize), the space of all maps is defined both by a very large number of orders (rapidly tremendous since $\frac{N!}{2}$ different orders exist) and by $N - 1$ continuous parameters (the distances).

The best criteria one can probably use to quantify how well a map explains the data set is the so-called *multipoint likelihood* of the map *i.e.*, the probability of observing the data set. It is a powerful criteria, which uses all the available evidence, but which may be in itself very expensive to compute. It is used in some packages such as MAPMAKER [Lander et al.87] or LINKAGE [Lathrop et al.85].

Using the previous notations, one can say that a recombination (resp. non-recombination) event has been observed for two adjacent markers when $|X_{\phi(\ell)}^i - X_{\phi(\ell+1)}^i|$ is equal to 1 (resp. 0). If we assume, as it is traditional, that recombination events occur independently in each interval, the probability of the observations given the map is simply obtained by multiplying the probabilities of all the events observed which yields the following formula:

$$\prod_{\ell=1}^{\ell=N-1} \prod_{i=1}^{i=K} \left[(1 - \hat{\theta}_{\ell, \ell+1}) \cdot (1 - |X_{\phi(\ell)}^i - X_{\phi(\ell+1)}^i|) + \hat{\theta}_{\ell, \ell+1} \cdot |X_{\phi(\ell)}^i - X_{\phi(\ell+1)}^i| \right]$$

The optimization problem is now well-defined: we have to find an order of the markers (defined by ϕ) and probabilities of recombinations (the $\hat{\theta}$ s) that maximize the likelihood. For a given order of the loci, we can easily compute the probabilities $\hat{\theta}^*$ that maximize the likelihood. Looking at first and second-order derivatives of the logarithm of previous formula, the $\hat{\theta}^*$ that maximize the likelihood can be easily obtained¹:

$$\hat{\theta}_{\ell, \ell+1}^* = \frac{\sum_{i=1}^{i=K} |X_{\phi(\ell)}^i - X_{\phi(\ell+1)}^i|}{K}$$

So, in this case, when all X_{ℓ}^i are known, and as far as recombination fractions are considered, the logarithm of the likelihood for optimal recombination fractions can be rewritten:

$$\sum_{\ell=1}^{\ell=N-1} \underbrace{K \cdot \left[\hat{\theta}_{\ell, \ell+1}^* \log(\hat{\theta}_{\ell, \ell+1}^*) + (1 - \hat{\theta}_{\ell, \ell+1}^*) \log(1 - \hat{\theta}_{\ell, \ell+1}^*) \right]}_{\text{elementary contribution to log-likelihood}}$$

Thus, the maximum log-likelihood of an order is equal to a sum of elementary contributions which depend only on two loci. One can therefore precompute all

¹ This is obtained by solving the simple equations stating that first-order derivatives are equal to 0. At this point, one can further notice that the matrix of second order derivatives is diagonal negative on the domain of optimization, which shows that this point is a maximum.

these contributions for all pairs of loci and the problem of finding an order that maximizes the maximum loglikelihood is in essence identical to the *symmetric wandering salesman problem* (a variant of the famous *symmetric traveling salesman problem*): given n cities and the distances between each pair of cities, find a path that goes once through each city and that minimizes the overall distance. The choice of the first and last cities in the path is free. One can simply associate one imaginary city to each marker, and define as the distance between two cities the opposite of the elementary contribution to the loglikelihood defined by the corresponding pair of markers².

This connection is interesting in several aspects: the WSP is known to be an NP-hard problem and this shows that the marker ordering problem may be difficult in some cases (computational complexity theory tells us that the tremendous number of existing orders is not sufficient to conclude this). More interestingly, all the techniques which have been developed for the TSP, and which can easily be adapted to the WSP, can also be applied here.

However, *real* data usually contain missing measures *i.e.*, some of the X_ℓ^i are unknown. In fact, there can be a lot of extra missing data introduced if the data set has been obtained by pooling several data sets on several different families, each family being informative on a different set of markers. In order to be able to handle data sets with missing measures, the quality of a map will not be simply obtained by summing elementary contributions as above, but using a dedicated statistical optimization algorithm : the EM algorithm [Dempster et al.77]. The algorithm given an order, computes the maximum likelihood of this order. It is a relatively expensive iterative procedure, each iteration being in $O(NK)$. Naturally, the pure theoretical connection with the WSP is lost when missing data exist, but our assumption is that the structure of the problem will still be close to the structure of the WSP and therefore, that techniques which proved to be efficient for the TSP will be efficient on the marker ordering problem.

2 Genetic mapping solving

When missing measures appear, the connection with the TSP is theoretically lost and the best available techniques, like Branch and Cut [Applegate et al.95], cannot be used. In that case, heuristic approaches offer an alternative to solve difficult optimization problems. Because heuristic approaches do not offer any garanty of optimality, we have used two types of algorithms to solve the genetic mapping problem: tabu search (TS) and genetic algorithm (GA). This allows to give more confidence in results when similar optima are found.

Genetic mapping software should not only give the most likely map, but also be able to indicate how strongly the best map is supported by the data (if there is another map whose likelihood is very close to the optimal map, then there is no reason to choose one or the other). To offer this service, our software, CAR_HAGENE, maintains a set of fixed size containing the k best different

² We would like to acknowledge the fact that the similarity between TSP and simpler two-points criteria is mentioned in [Liu95].

maps encountered during the search (k fixed, actually equal to 31). A hash table [Cormen et al.90] is used to efficiently test if the map is already in the set, a heap structure is used to efficiently manage insertions/deletions [Cormen et al.90]. At the end of the search, the user can browse this set and check for the existence of other maps whose likelihood is close to the best map’s likelihood in order to get an idea of how strongly the best map is supported. This is incorporated in all the algorithms presented in the sequel.

2.1 Tabu search algorithm (TS)

Tabu search [Glover89, Glover90] repeatedly scans the current neighborhood, selecting the best neighbor (the neighbor with the best likelihood) to be the new solution. The neighborhood we have chosen is the *2-change* neighborhood of the individual, a successful well-known neighborhood structure introduced in [Lin et al.73] to tackle the TSP. Adapted to the WSP, the *2-change* neighborhood of a map is the set of all maps obtained by an inversion of a subsection of the map. Thus, for N markers, the neighborhood has a size of $\frac{N \cdot (N-1)}{2} - 1$.

To avoid being stucked in local optima, the content of the neighborhood of the current solution is influenced by a memory mechanism which may forbid some moves (which are said to be tabu) in the neighborhood. In CAR_HTAGENE, the tabu moves are the recent moves (*i.e.*, subsections of the map which have been recently inverted). The precise definition of “being recent or not” varies stochastically during search as advocated by [Taillard91]. A tabu move may eventually be chosen if it leads to a map which improves the best likelihood known (this is called “aspiration” in tabu terminology).

We observed that tabu moves, while breaking cycles, weren’t able to avoid being stuck in large locally optimum plateaus. We therefore used the hash table that memorizes the best orders encountered to also memorize the number of times each of them is reached. When this count exceeds a fixed number, a random jump is performed. This largely enhanced performances. However, since only the best solutions are memorised in the hash table, it is still useless if the algorithm is stucked in a plateau of poor quality (defined by orders whose likelihood is far from the best likelihoods memorized in the hash table). Further improvement should be possible by adding a new hash table that memorizes *recent* positions rather than *best* positions.

2.2 Basic genetic algorithm (BGA)

Genetic algorithms are general adaptative heuristic search algorithms based on an analogy with the genetic structure and behavior of chromosomes within a population of individuals. Individuals represent potential solutions to a given problem. A fitness score is associated to each individual and represents its adaptation ability. The algorithm makes the population of individuals evolve maintaining both diversity and favoring the existence of best individuals. Starting from an initial population, a new generation is created by randomly applying

mutation and by crossing pairs of individuals (favoring crosses of good individuals). The hope is that the population will evolve towards one which contains optimal individuals with respect to the fitness.

Representation: In CAR_HAGENE, each individual represents one genetic map (an ordering of markers). The crossover operator used is the order crossover [Telfar94] which computes two offspring individuals, I_1 and I_2 from two parents P_1 and P_2 (figure 3). The parents are cut into three sections by selecting randomly two markers. The middle section of P_1 is copied into the corresponding position of I_1 , the rest of I_1 being filled in with values taken in order from the third, first and second section of P_2 , skipping values that have already been copied from the first parent. I_2 is computed in the same way by reversing the parents.

P1 : 2 3 7	4 1 6 9	8 5 10
P2 : 5 9 1	6 7 3 2	10 4 8
I1 : 7 3 2	4 1 6 9	10 8 5
I2 : 4 1 9	6 7 3 2	8 5 10

Fig. 3. Example of the order crossover operator. I_1 is computed from its parents P_1 and P_2 . The middle section of P_1 is copied into the middle section of I_1 . The rest of I_1 is filled in with respectively 10 and 8 from section 3 of P_2 , 5 from section 1 of P_2 and 7, 3 and 2 from section 2. I_2 is computed in the same way by reversing the parents.

The mutation operator selects two markers and simply exchanges them. The selection relies on a biased roulette wheel [Goldberg89]. For markers ordering, a solution is an order of the markers, and its fitness score is the maximum likelihood of the map. Thus a simple fitness function is given by the evaluation of the map using the EM algorithm.

A comparison of the basic genetic algorithm with tabu search was performed on randomly generated problems³. On such problems, the original map used to generate the data, and its likelihood are naturally available. The likelihood of the original map is usually among the highest and gives a generally good lower bound of the optimal likelihood.

	GO	BL
<i>BGA</i>	59 %	70 %
<i>TS</i>	81 %	100 %

Fig. 4. Results

In our test, we measured the number of times each algorithm was able to found the original order (noted GO, aka Good Order) and the number of times each algorithm was able to found a likelihood larger than or equal to the likelihood of the original map (noted BL, aka Better Likelihood). The results show that *TS* gives better results than *BGA*. We explain these results by the power of the greedy optimization which is performed in *TS*. This suggested us to embed such a greedy optimization in *BGA*.

³ Simulated data consist of $N = 25$ markers per individual and the number of individuals in each data set is 100. The probability p for an allele to be missing is 20%. One hundred problems were solved.

2.3 Incorporating greedy optimization in genetic algorithms

There are several different possible ways to use the idea of *2-change* neighborhood in a genetic algorithm:

- A first question is whether a full greedy optimization should be performed (until a local optima is reached) or if a limited and cheaper optimization (a fixed number of greedy steps) would suffice.
- A second question is when the actual greedy optimization should take place. It could be incorporated in the mutation operator, or applied systematically (eg. inside the fitness evaluation function).

After several trials, it appeared that performing a full greedy optimization (until a local optimum is reached) on each new individual yields the best performances. In practice, the greedy optimization is performed during fitness evaluation: when a local optima is reached, the individual is replaced by this local optimum and its likelihood is used as the fitness. This function usually modifies the individual under evaluation. It is expensive, but it yields the best results.

With this approach, the genetic algorithm does not perform an optimization in the full space of all orders, but only in the space of local optima, skipping from a local optimum to another throughout the crossover and mutation operators. In the sequel of the paper, the resulting algorithm will be denoted *GA*. On the previous data sets, it performed exactly as *TS*.

3 Experimental results

CAR_HTAGENE has been applied both on simulated and real backcross-like data. In each case, an empirical analysis is used to compare genetic algorithm (*GA*) and tabu search.

Results now compare the performances of the algorithms when running with their own stopping criteria. In each case, the stopping criteria depends on the number of markers and performances are evaluated with regard to the quality of solutions and the number of EM calls.

3.1 Simulated data

In Figure 5 we investigate the relation between the percentage of missing data and the number of individuals needed to find optimal orders that are original orders. In each individual data set, the number of markers is 10. The number of individuals in each data set is either 25, 50, 100 or 250. The probability for an allele to be missing varies from 0% to 20% by 5% step. One hundred problems are solved for each combination of these parameters. Each curve is associated with a number of individuals and gives the percentage of original orders found by CAR_HTAGENE. It appears that 250 or even 100 individuals seem to be largely sufficient to find the good order even with missing data. These curves are similar for both *TS* and *GA*.

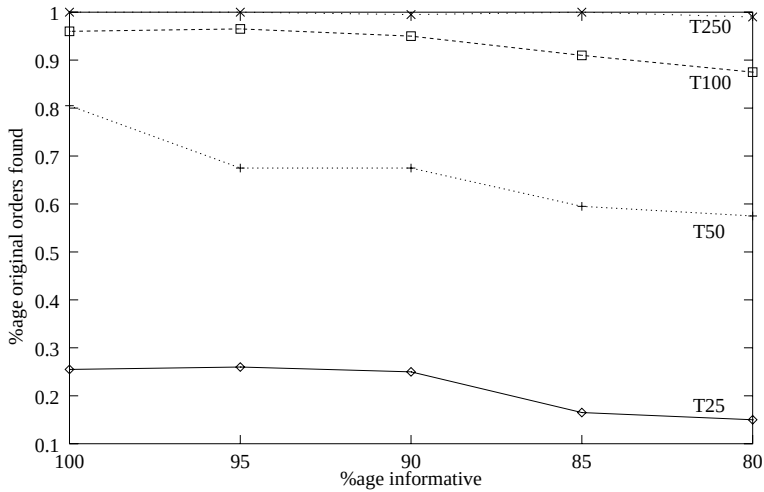


Fig. 5. Percentage of original orders found.

We do not report here the curves giving the percentage of maps found with a loglikelihood equal or better than the loglikelihood of the original map. It was always equal to 100%.

Table 1 shows results obtained by running genetic algorithm and tabu search with their own stopping criteria on backcross-like data involving two populations.

	2 markers in common						5 markers in common						10 markers in common					
	Failures			EM calls			Failures			EM calls			Failures			EM calls		
size	<i>TS</i>	<i>GA</i>	both	<i>TS</i>	<i>GA</i>		<i>TS</i>	<i>GA</i>	both	<i>TS</i>	<i>GA</i>		<i>TS</i>	<i>GA</i>	both	<i>TS</i>	<i>GA</i>	
25	4	0	0	29998	38433		4	0	0	20833	25879		7	4	1	9672	10093	
50	18	3	0	31174	39880		6	2	0	21460	28305		9	7	2	9546	9259	
100	13	3	0	32750	41120		2	0	0	21959	29688		7	6	0	9338	9218	
250	10	0	0	33903	42961		4	0	0	22554	30129		25	8	3	9430	8427	
Mean	11.2	1.5	0	31956	40598		4	0.5	0	21702	28500		12	6.25	1.5	9496	9249	

Table 1. Comparison of *GA* and *TS* with an increasing number of common markers and different sizes of sample. A failure occurs when the best likelihood found is worse than the likelihood of the original map.

Data have been generated as described in [Schiex et al.97]. In each individual data set, the number of markers is 15. The number of common markers is respectively 2, 5 and 10. The size of each data set is either 25, 50, 100 or 250, leading to a joined map built from data sets of 50, 100, 200 and 500 individuals. The probability p for an allele to be missing is 20%. Additional blocks of missing

data are introduced which come from markers available in one individual data set and not in the other. This results in a percentage of missing data always higher than p depending on the number of common markers. The number of markers varies from 20 (15 markers in each population and 10 common markers) to 28 (15 markers in each population and 2 common markers). One hundred problems are solved for each combination of these parameters.

Stopping criteria have been tuned so that “good quality” solutions are produced in a “reasonable” amount of time. Tabu search stopping criteria considers the number of iterations which have not improved the likelihood of the best solution. When this number is equal to twice the number of markers in the data set, tabu search stops. Genetic algorithm stops as soon as two generations of individuals produce the same best individual or the maximum likelihood has not been improved.

In table 1, it appears that the number of EM calls decreases with the number of common markers for both approaches. This can be explained by the fact that the total number of markers decreases with the increasing number of common markers and the stopping criteria depends on the total number of markers.

In all cases *GA* appears better than *TS* in finding best maps and this remains true for other values of p ($p = 10\%$ and $p = 0\%$, results not given there). Nevertheless, one can note that *TS* and *GA* generally do not fail on the same instances. One can exploit this complementarity to obtain a lower failing rate. Nevertheless, in all cases, except for 10 markers in common and sample sizes 50, 100 and 250, *TS* stops before *GA*. In spite of the fact that the case of 10 common markers indicates *GA* as the most powerful approach as well for the number of EM calls as the quality of solutions, these results are certainly strongly linked to the running protocols that have been used and therefore have to be considered with caution.

3.2 Real backcross-like data

The real data consist of backcross-like data for the *Trichogramma brassicae* genome. Full results are described in [Laurent et al.] where *TS* and *GA* give similar solutions. One can note there that *TS* was always less expensive in EM calls than *GA*. We also compared CAR_HAGENE with existing software on these data. For all the maps, the orders obtained using CAR_HAGENE were more likely (up to 10^{16} more likely) than JOINMAP, an existing commonly used software [Schiex et al.97].

Conclusion

CAR_HAGENE shows how existing efficient neighborhood structures can be exploited to boost the performances of genetic algorithms. It is likely that the idea of using greedy local search in the fitness function could be used for other problems as well.

Experiments show how difficult the measure and comparison of performances between different approaches is. On simulated and real backcross-like data tabu search and genetic algorithms give similar results when the number of markers is around 15. In those cases, *TS* is always less consuming in EM calls than *GA*. The case of simulated data with a higher number of markers and 20% of missing data gives *GA* as the most robust approach. This result is to be considered with caution and more experiments remain to do to confirm it. More interestingly, the analysis of the failures suggests to combine the two approaches in order to increase the reliability.

In genetic mapping, CAR_THAGENE extends the scope of application of multi-point maximum likelihood criterion to data sets with larger number of markers and larger amount of missing, a situation frequently encountered when the data set is build from several families. In this case, it compares quite favorably with other dedicated packages [Stam93, Schiex et al.97].

References

- [Applegate et al.95] Applegate (D.), Bixby (R.), Chvátal (V.) et Cook (W.). – *Finding cuts in the TSP (a preliminary report)*. – Technical Report 95-05, DIMACS, March 1995.
- [Cormen et al.90] Cormen (Thomas H.), Leiserson (Charles E.) et Rivest (Ronald L.). – *Introduction to algorithms*. – MIT Press, 1990. ISBN : 0-262-03141-8.
- [Dempster et al.77] Dempster (A.P.), Laird (N.M.) et Rubin (D.B.). – Maximum likelihood from incomplete data via the EM algorithm. *J. R. Statist. Soc. Ser.*, vol. 39, 1977, pp. 1–38.
- [Glover89] Glover (F.). – Tabu search – part I. *ORSA Journal on Computing*, vol. 1 (3), Summer 1989, pp. 190–206.
- [Glover90] Glover (F.). – Tabu search – part II. *ORSA Journal on Computing*, vol. 2 (1), Winter 1990, pp. 4–31.
- [Goldberg89] Goldberg (D. E.). – *Genetic algorithms in Search, Optimization and Machine Learning*. – Addison-Wesley Publishing Company, 1989.
- [Lander et al.87] Lander (E.S.), Green (P.), Abrahamson (J.), Barlow (A.), Daly (M. J.), Lincoln (S. E.) et Newburg (L.). – MAPMAKER: An interactive computer package for constructing primary genetic linkage maps of experimental and natural populations. *Genomics*, vol. 1, 1987, pp. 174–181.
- [Lathrop et al.85] Lathrop (G.M.), Lalouel (J.M.), Julier (C.) et Ott (J.). – Multilocus linkage analysis in humans: detection of linkage and estimation of recombination. *Am. J. Hum. Genet.*, vol. 37, 1985, pp. 482–488.
- [Laurent et al.] Laurent (V.), Vanlerberghe-Masutti (F.), Wajnberg (E.), Mangin (B.), Schiex (T.) et Gaspin (C.). – Construction of a composite genetic map of the parasitoid wasp *trichogramma brassicae* using RAPD informations from three populations. *Submitted*.
- [Lin et al.73] Lin (S.) et Kernighan (B. W.). – An effective heuristic algorithm for the traveling salesman problem. *Operation Research*, vol. 21, 1973, pp. 498–516.

- [Liu95] Liu (B. H.). – The gene ordering problem, an analog of the traveling salesman problem. *In: Plant Genome 95*.
- [Schiex et al.97] Schiex (T.) et Gaspin (C.). – Cartagene: Constructing and joining maximum likelihood genetic maps. *In: Proceedings of the fifth international conference on Intelligent Systems for Molecular Biology*. – Porto Carras, Halkidiki, Greece, 1997. Available at <http://www-bia.inra.fr/T/schiex>.
- [Stam93] Stam (P.). – Constructing of integrated genetic linkage maps by means of a new computer package:JOINMAP. *The Plant Journal*, vol. 3 (5), 1993, pp. 739–744.
- [Taillard91] Taillard (E.). – Robust taboo search for the quadratic assignment problem. *Parallel computing*, vol. 17, 1991, pp. 443–455.
- [Telfar94] Telfar (G.). – *Generally Applicable Heuristics for Global Optimization: An Investigation of Algorithm Performance for the Euclidean TSP*. – Master's thesis, Victoria University of Wellington, 1994.

Inverse problems for finite automata: a solution based on Genetic Algorithms

B. Leblanc¹, E. Lutton¹ and J.-P. Allouche²

¹ INRIA - Rocquencourt, B.P. 105, F-78153 LE CHESNAY Cedex, France
Tel: +33 (0)1 39 63 55 23 - Fax: +33 (0)1 39 63 59 95
e-mail: Benoit.Leblanc@inria.fr, Evelyne.Lutton@inria.fr

<http://www-rocq.inria.fr/fractales/>

² CNRS, LRI, Bât. 490, Université Paris-Sud, F-91405 Orsay Cedex, France
Tel: 33 (0)1 69 15 64 54
e-mail: Jean-Paul.Allouche@lri.fr

Abstract. The use of heuristics such as Genetic Algorithm optimisation methods is appealing in a large range of inverse problems. The problem presented here deals with the mathematical analysis of sequences generated by finite automata. There is no known general exact method for solving the associated inverse problem. GA optimisation techniques can provide useful results, even in the very particular area of mathematical analysis. This paper presents the results we have obtained on the inverse problem for fixed point automata. Software implementation has been developed with the help of “ALGON”, our home-made Genetic Algorithm software.

1 Introduction

A *finite automaton* is defined as a symbolic substitution σ acting on strings of symbols. More precisely σ is a map from a finite set of symbols S to S^* , the set of strings of symbols in S . The elements of S^* are called *words* and the images by σ of elements of S are called *words of the automaton*. The map σ is extended to S^* by concatenation (the image of a word is obtained by concatenating the images of its symbols), making σ a morphism of the free monoid S^* .

A sequence of words can be produced by successive applications of σ to an initial word s_0 . If we denote by $s_n = s_{n_1}s_{n_2}\dots s_{n_p}$ the word at step n , the word obtained at step $n+1$ is then:

$$s_{n+1} = \sigma(s_n) = \sigma(s_{n_1})\sigma(s_{n_2})\dots\sigma(s_{n_p}).$$

Note that the words $(s_n)_{n \in \mathbb{N}}$ are concatenations of the words of the automaton (in Example 1 this fact is highlighted by the alternation of bold and standard fonts).

Example 1. $S = \{1, 2, 3\}$

$$\sigma \begin{cases} 1 \rightarrow 211 \\ 2 \rightarrow 13 \\ 3 \rightarrow 123 \end{cases}$$

Iteration	Word
0	1
1	211
2	13211211
3	2111231321121113...

Of course, it is clear that the sequence of words $(s_n)_{n \in \mathbb{N}}$ is determined by the automaton σ and the initial word s_0 .

An interesting property of such words concerns the frequency of occurrences of symbols of S . Let σ be an automaton acting on $S = \{\alpha_1, \alpha_2, \dots, \alpha_m\}$, let s_0 be an initial word. The number of occurrences of any of the symbols observed in the word s_k (for any k) can be computed. Let us denote by

$$O_k = \begin{pmatrix} o_k^1 \\ o_k^2 \\ \dots \\ o_k^m \end{pmatrix}$$

the occurrence vector of s_k (o_k^i being the number of occurrences of the symbol α_i observed in s_k). Then $O_{k+1} = A * O_k$ with $A = (a_{ij})_{(i,j) \in \{1, \dots, m\}^2}$ the “growth” matrix, a_{ij} being the number of symbols α_i in the word $\sigma(\alpha_j)$.

We thus obtain $O_k = A^k * O_0$ with O_0 the occurrence vector of s_0 . For Example 1, we have:

$$A = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}.$$

Let us also define the square (and in the same way any power) of an automaton: $\forall i \in \{1, \dots, m\}$, $\sigma^2(\alpha_i) = \sigma(\sigma(\alpha_i))$. The associated matrix is then A^2 . For more information about substitutions, see [4].

2 The inverse problem for finite automata

2.1 Motivations and formulation

To know whether a given sequence is generated by a finite automaton and to know explicitly one such automaton, can be useful in many situations. We list but three of them.

- In combinatorics on words: the third author, J. Currie and J. Shallit proved recently that the lexicographically least overlap-free³ sequence on a two-letter alphabet, that begins with a given word (if it exists), must end with a

³ An *overlap* is a string of the form $axaxa$ where a is a letter and x a (finite) word. A (finite or infinite) word is called *overlap-free* if it does not contain any overlap.

tail of the Thue-Morse sequence $10010110\dots$, hence is the pointwise image (i.e. image under a morphism that sends each letter to a letter) of a fixed point of a morphism of constant length [8]. It is not known whether the least square-free sequence on a three-letter alphabet has the same property.

- In number theory: let $(u_n)_{n \in \mathbb{N}}$ be a sequence with values in the finite field $\mathbb{F}_q$. Then the formal power series $\sum u_n X^n$ is algebraic over the field of rational functions $\mathbb{F}_q(X)$ if and only if the sequence $(u_n)_{n \in \mathbb{N}}$ is the pointwise image of a fixed point of a morphism of length q over a finite alphabet, [10, 11]. For example the transcendence of values of Carlitz functions can be proved by showing the non-automaticity of the corresponding formal power series (see for example [9, 7, 12]). A hint that a sequence is not a fixed point of a morphism (it is more complicated for the pointwise image of a fixed point) is that, when solving the inverse problem with longer and longer prefixes of the sequence, the automaton we obtain keeps growing. That means almost certainly that there is no automaton that generates the infinite sequence.
- In physics: quasi-crystals are the 3-D analogue of the Penrose tiling. A one-dimensional description of this tiling involves the Fibonacci sequence, i.e., the fixed point of the morphism $0 \rightarrow 01, 1 \rightarrow 0$.

Another occurrence of this “inverse” question for finite automata, in music, is recalled below.

Suppose we have a string of symbols and we want to know whether it has been produced by iterating an automaton. Of course if the string is finite, there always exists a trivial solution: the substitution sending the first letter of the string to the string itself. But we are interested in non-trivial solutions if any. Apart from the mathematical aspects of this “inverse” problem, it might be interesting to note that this work began as a composer, T. Johnson, produced a sequence of notes using a finite automaton, kept only the sequence of notes, and wanted to remember the automaton he used. (For the use of finite automata in a piece of T. Johnson, see [1].)

If the word s_0 happens to be a prefix of the word $s_1 = \sigma(s_0)$, it is not hard to see that the sequence of words $(s_n)_{n \in \mathbb{N}}$ converges to an infinite word. And this infinite word is clearly a *fixed point* of the substitution σ (extended to infinite words by concatenation). Now suppose an infinite word is given, is this word the fixed point of a substitution? Or is it the pointwise image of a fixed point of a substitution? No general answer to these questions is known either: a theoretical answer to the second question is known if the substitution has constant length [3] (i.e., if all words of the automaton have the same length), and also if the substitution is *primitive* (i.e., is such that there exists an m with the property that σ^m of any symbol contains at least one occurrence of each symbol of the set S) as proved recently by Durand [5]. Looking at finite prefixes of the given infinite sequence that have well-chosen lengths, we see that we can first restrict to finite words.

Hence, ideally, we would like to solve in the general case (i.e., not only for fixed point automata) what we called the inverse problem, that is:

Given a finite word s , find an automaton σ and an initial word s_0 such that $\sigma^n(s_0) = s$ for some n .

Of course, in its generality, this problem is extremely complex and can have many solutions or no non-trivial solution at all⁴.

It can be reformulated as an optimisation problem on the search space of all possible σ , n and s_0 that minimizes a distance between $\sigma^n(s_0)$ and s . In order to reduce the complexity of this problem one also has to give some restrictions to the search space. We suppose in the following that the length of the words of σ is limited to l_{max} and that s_0 is a single symbol.

2.2 A bruteforce GA implementation

The first GA implementation that comes to mind is to perform a search over the space of all automata having word lengths smaller than or equal to l_{max} . Each individual of the GA thus represents an automaton with the following characteristics:

- m chromosomes for an individual: one per word of the automaton;
- variable length chromosomes: their lengths may vary between 1 and l_{max} ;
- an m -ary coding: the allele set is S .

These particular characteristics imply of course some modified GA operators, that are implemented in ALGON [6].

Thus, setting $l_{max} = 4$, the automaton of Example 1 would have the coding shown in Figure 1.

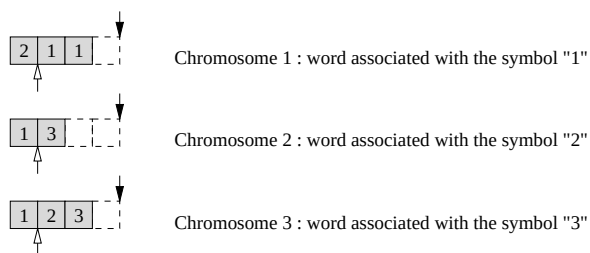


Fig. 1. Direct coding of example 1 with $l_{max} = 4$.

The fitness function that must reflect the “resemblance to the target”, is based on a comparison between words (Hamming distance, frequencies of occurrences of symbols, of couples of symbols, etc. . .).

⁴ In fact, for a given solution couple (σ, s_0) and for any divisor p of n , the couple (σ^p, s_0) is also a solution.

This approach did not lead to interesting results, due to the size of the search space with respect to m and l_{max} : $|S| = \left(\sum_{i=1}^{l_{max}} m^i\right)^m$. The size would not be a problem if the resulting fitness landscape was smooth enough⁵, but in our case one can easily check that a single change in the genetic code leads to important changes in the observed words.

Though this direct approach is not appropriate, its benefit is to highlight the difficulty of solving this problem. In fact, it is obvious that the coding should use more efficiently the information contained in the target word.

3 The fixed point hypothesis

If we restrict the inverse problem to the search of automata with fixed points, the complexity of the problem is reduced.

3.1 Definition and properties

A finite automaton has a fixed point if there exists a symbol α such that the first letter of $\sigma(\alpha)$ is α itself.

The sequence of words s_n produced by such an automaton starting with the initial symbol $s_0 = \alpha$ converge to a fixed point of σ : the beginning of the word at iteration $n + 1$ is exactly the word at iteration n . In fact each iteration adds symbols to the end of the previous word.

Example 2. $S = \{1, 2, 3\}$

$$\sigma \begin{cases} 1 \rightarrow 21 \\ 2 \rightarrow \mathbf{231} \\ 3 \rightarrow 13 \end{cases}$$

Iteration	Word
0	2
1	231
2	2311321
3	2311321211323121 ...

The inverse problem for a fixed point automaton is then much easier to solve than in the general case. Indeed, the information contained in the target word can be efficiently exploited, taking advantage of the fact that a fixed point is a succession of words of the automaton as well as the succession of symbols which generated them. Of course, it is necessary to know the lengths of the words in order to identify the connection between the two successions. A simple assumption on the lengths of words of the automaton permits then to identify it with a mechanism of simultaneous identification and reconstruction.

Checking an hypothesis is then a direct process of “reconstruction-comparison”. As previously outlined, the first symbol of the fixed point is associated to the first word, its size is given by assumption, so it can be identified. The second

⁵ With a few secondary optima.

Incorrect hypothesis (2,2,2).		Correct hypothesis (2,3,2).	
$\sigma(2) = 23$	23 11321211323121	$\sigma(2) = 231$	231 1321211323121
$\sigma(3) = 11$	23 11 321211323121	$\sigma(3) = 13$	231 13 21211323121
$\sigma(1) = 32$	2311 32 1211323121	$\sigma(1) = 21$	23113 21 211323121
$\sigma(1) = 12$	231132 12 11323121	$\sigma(1) = 21$	2311321 21 1323121
Contradiction on "1".		$\sigma(3) = 13$	231132121 13 23121
		$\sigma(2) = 231$	23113212113 231 21
		$\sigma(1) = 21$	23113212113231 21

Fig. 2. Hypothesis propagation.

word, associated to the second letter, start right after the first, and knowing its size by hypothesis it can be identified too, and so on all along the fixed point. If the hypothesis is not correct, then the case will arise when the same symbol will be associated to two different words, discarding it. If it is correct the whole fixed point will be "reconstructed" without such contradiction.

Let us consider Example 2 and take $s = \sigma^3(2)$ as the target word. Assume each word of σ has two symbols:

- The initial symbol s_0 is simply the first symbol of s , i.e., $s_0 = 2$.
- The word associated with this symbol is a prefix of the target word, and it is assumed to be composed of two symbols, then we directly identify $\sigma(2) = 23$.
- The identification process is continued until a contradiction appears or the end of the target word is reached, as shown in Figure 2: in the incorrect hypothesis case we get $\sigma(1) = 32$ at step 2 and $\sigma(1) = 12$ at step 3 then the hypothesis is infirmed. Conversely, in the correct case, for the same symbol, the same word is always recognized, so the whole fixed point word is "reconstructed".

3.2 A GA to search the space of word lengths

Coding the individuals :

An individual of the GA population may just represent an assumption on the words lengths, the corresponding automaton being reachable through the identification mechanism previously exposed.

If we set an upper limit l_{max} for the possible lengths of the words, the genetic coding is the following:

- A set of alleles of cardinality l_{max} .
- A single chromosome per individual containing as many genes as elements in the symbol set S . The gene k codes the length of the word associated with the symbol α_k . The coding of a right assumption for Example 2 is:

$$|2|3|2| \rightarrow \sigma \begin{cases} 1 \rightarrow ?? \\ 2 \rightarrow ??? \\ 3 \rightarrow ?? \end{cases}$$

Compared to the “brute-force” implementation, a substantial improvement is the reduction of the search space which size is now $|S| = (l_{max})^m$.

Fitness function :

The evaluation of an individual relies on the validation process of the assumption it encodes. If the assumption appears to be valid, it is assigned a maximal fitness value. Note that any power of an automaton that is a solution to the problem is also a solution. Hence there is a potentially infinite number of solutions, as soon as one solution is found. But practically the number of solutions is limited by the length of the target word and the l_{max} value. The minimal solutions (in terms of lengths) are obviously the most interesting ones.

Invalid assumptions are given an intermediate value, and it is also desirable to differentiate these non-valid assumptions in order to drive the search towards a solution.

If a contradiction arises, two cases are considered:

- The contradiction arises before the identification of all the words of the automaton:

$$f(i) = \epsilon * \text{Number of identified words}$$

with ϵ a very small positive value.

- If a contradiction arises after the identification of all words of the automaton:

$$f(i) = \left(\frac{\text{Length of the “checked” sequence}}{\text{Length of the target sequence}} \right)$$

The “checked” word denotes the part of the target word checked before the contradiction occurs.

The maximum of f is then 1, corresponding to the case where the target word has been entirely checked. In order to give a best fitness value to any assumption leading to a complete identification of the words of the automaton than to any other that doesn't, the number ϵ simply has to fulfill the following condition:

$$\epsilon < \left(\frac{m + 1}{\text{Length of the target word}} \right).$$

3.3 Results and discussions

We present here results obtained with ALGON [6], on two target words which are prefixes of fixed points of two different automata using 6 symbols. The maximal lengths of words being $l_{max} = 6$, the size of the search space is then: $|S| = 6^6 = 45656$

The general parameters of the GA are:

- A population of 100 individuals. Each individual being unique.
- A mutation probability $p_m = 0.125$.
- One point crossover with probability $p_c = 0.85$.

- An elitist population replacement with a ratio $r_s = 0.4$ of surviving individuals, i.e., 60 new individuals are created at each generation replacing the 60 worst individuals of the previous population.
- Selection performed with *Stochastic Universal Sampling* (see [2]).

Automaton 1:

$$\sigma \left\{ \begin{array}{l} 1 \rightarrow 61 \\ 2 \rightarrow \mathbf{2}34 \\ 3 \rightarrow 52 \\ 4 \rightarrow 234 \\ 5 \rightarrow 6551 \\ 6 \rightarrow 433 \end{array} \right. \quad (1)$$

The target word s is then obtained by iterating 5 times the automaton starting from the initial seed “2”, that is a word of length 196. We present in table 1 some statistics obtained over 20 runs. The following quantities are computed:

N_1 : Number of generations to obtain an assumption leading to a complete automaton (before a contradiction arises in the identification process).

N_2 : Number of generations to find a solution.

$N_3 = N_2 - N_1$: number of generations to find a solution when at least one individual has lead to a complete automaton.

The results are summarized in table 3.3.

Table 1. Results for automaton 1.

	Mean	Std.
N_1	4.2	2.38
N_2	18.95	24
N_3	14.75	23.3

About 1000 fitness evaluations are necessary to find a solution, which is to be compared to the search space size.

Automaton 2:

$$\sigma \left\{ \begin{array}{l} 1 \rightarrow 11116 \\ 2 \rightarrow \mathbf{2}4 \\ 3 \rightarrow 5 \\ 4 \rightarrow 35 \\ 5 \rightarrow 23341 \\ 6 \rightarrow 666 \end{array} \right. \quad (2)$$

The target word s is again obtained by iterating 5 times the automaton starting from the initial seed “2”. It has been designed to slow down the automaton

identification process: the symbol “6” first appears quite far in s (26th position), so a contradiction has a greater chance to arise before all words have been identified.

Table 2. Results for automaton 2.

	Mean	Std.
N_1	8.65	2.38
N_2	13.2	7.76
N_3	4.55	3.76

We can see that, for this apparently more tricky automaton, the performances of the GA are better. But it can certainly be explained by the fact that the frequency of the hypothesis leading to a complete automaton identification (before a contradiction arises) is lower than previously, but other points of the search space seem to lead quite easily to those interesting regions.

4 Conclusion and further works

The results we obtained on fixed points automata suggest a coding of the general problem based on a set of possible words observed in the target word to be analysed. Such an approach, by considerably reducing the search space of possible automata, allows to obtain interesting results in the general case. This will be studied in a forthcoming paper.

References

1. J.-P. Allouche, T. Johnson (1995): *Finite automata and morphisms in assisted musical composition*. Journal of New Music Research **24**, 97–108.
2. J. E. Baker (1987): *Reducing bias and inefficiency in the selection algorithm*. Genetic Algorithms and their application: Proceedings of the Second International Conference on Genetic Algorithms, p. 14–21.
3. A. Cobham (1972): *Uniform tag sequences*. Math. Systems Theory **6**, 164–192.
4. S. Eilenberg (1974): *Automata, Languages, and Machines*. Vol. A, Academic Press.
5. F. Durand (1997): *A characterization of substitutive sequences using return words*. Disc. Math., to appear.
6. B. Leblanc, E. Lutton (1997): *ALGON: A Genetic Algorithm software package*, <http://www-rocq.inria.fr/fractales/>
7. J.-P. Allouche (1996): *Transcendence of the Carlitz-Goss Gamma function at rational arguments*. J. Number Theory **60**, 318–328.
8. J.-P. Allouche, J. Currie, J. Shallit (1997): *Extremal infinite overlap-free binary words*. Preprint.

9. V. Berthé (1994): *Automates et valeurs de transcendance du logarithme de Carlitz*. Acta Arith. **66**, 369–390.
10. G. Christol (1979): *Ensembles presque périodiques k -reconnaissables*. Theoret. Comput. Sci. **9**, 141–145.
11. G. Christol, T. Kamae, M. Mendès France, G. Rauzy (1980): *Suites algébriques, automates et substitutions*. Bull. Soc. Math. France **108**, 401–419.
12. M. Mendès France, J.-y. Yao (1997): *Transcendence and the Carlitz-Goss gamma function*. J. Number Theory **63**, 396–402.

Evolving Turing Machines From Examples

Julio Tanomaru

Faculty of Engineering, The University of Tokushima,
Tokushima 770 Japan

Abstract. The aim of this paper is to investigate the application of evolutionary approaches to the automatic design of automata in general, and Turing machines, in particular. Here, each automaton is represented directly by its state transition table and the number of states is allowed to change dynamically as evolution takes place. This approach contrasts with less natural representation methods such as trees of genetic programming, and allows for easier visualization and hardware implementation of the obtained automata. Two methods are proposed, namely, a straightforward, genetic-algorithm-like one, and a more sophisticated approach involving several operators and the 1/5 rule of evolution strategy. Experiments were carried out for the automatic generation of Turing machines from examples of input and output tapes for problems of sorting, unary arithmetic, and language acceptance, and the results indicate the feasibility of the evolutionary approach. Since Turing machines can be viewed as general representations of computer programs, the proposed approach can be thought of as a step towards the generation of programs and algorithms by evolution.

1 Introduction

Recently, paradigms of *Evolutionary Computing* [3], a relatively novel area of Computer Science which employs principles of natural evolution to solve engineering problems, have been proposed as alternative methods for optimization and machine learning. Focusing on the automatic generation of automata, the literature registers approaches based on *Genetic Algorithms* (GAs) [7], *Evolutionary Programming* (EP) [5], and *Genetic Programming* (GP) [9], mainly the last two fields.

When evolving automata, application of a GA demands the design of a coding scheme to represent each automaton as a chromosome. This process often results in complex representations and in the need of customized operations to avoid the generation of chromosomes that cannot be translated into meaningful automata. Furthermore, with a few notable exceptions [6, 8], GAs operate on fixed-length chromosomes. This often poses a problem when one does not know *a priori* the optimal size of the automaton necessary to solve a given task.

On the other hand, while the core of EP methods seems to be suitable to the evolution of automata, EP has only been tested on specific problems of evolution of small-scale automata for Artificial Intelligence (AI) problems [4]. Finally, concerning GP, this method requires that the automata be mapped into

convenient tree structures, which are then evolved. Later, after evolution finishes or is terminated, it is necessary to map the obtained solutions back to the space of automata. Generally, this last operation is not straightforward.

This paper examines an alternative form of evolutionary generation of automata, in which each automaton in a population is represented by the corresponding state transition table (STT), and the number of states is allowed to change dynamically. Overall, the system resembles a customized GA in its population model, selection method and operators, but instead of chromosome representations, having each automaton denoted by its STT facilitates the interpretation of the evolution results. This is an essential property when the automata are to be analyzed or implemented in hardware. Genetic operators were devised and applied to the particular case in which each automaton is a Turing Machine (TM), chosen because of its generality as a computer model.

2 Related Research

Perhaps the first application of evolutionary techniques to generate automata was done by Fogel, Owens, and Walsh [5], who developed EP to evolve small automata to solve an environment (represented by a sequence of symbols) prediction problem. Recently, there have been new applications of the technique to evolve automata for specific problems as, for example, to solve the classic prisoner's dilemma and to model human behavior observations [1]. Studies involving the application of EP to produce more general automata have not been done so far.

The need for a chromosome representation as a string of symbols from a finite alphabet, together with the fact that virtually all GAs operate on chromosomes of fixed length, make it troublesome to evolve automata by GAs. One of the few reported applications was for the task of finding an automaton to navigate an artificial ant so as to find all the food lying along an irregular trail defined on a toroidal grid of 32×32 cells [2]. Collins and Jefferson coded each possible state and action as a binary substring and concatenated such substrings in a fixed order, in such a way that any finite state automaton could be represented by a binary string. Although it was able to find high-quality solutions for the ant problem, that approach assumes that at least a good estimate of the necessary number of states is known beforehand.

Finally, GP has been extensively applied to automatic generation of programs and automata in recent years [9]. However, GP deals with the evolution of tree-like structures, usually LISP S-expressions, combining primitive functions and operators. This creates a representation problem when trying to evolve automata, since GP is not directly applicable to automata represented by their STTs. As a consequence, it becomes difficult to translate the programs obtained by evolution into automata, which is the original problem.

We argue that an ideal evolutionary method for generation of automata should have intuitive representation, allow for easy mapping of solutions to STTs

(to facilitate analysis and hardware implementation by flip-flops and logic gates), allow for variation of the number of states, and be robust.

3 Proposed Approaches

3.1 Automata Representation and Population

Generally, an automaton M is a sequential machine specified as

$$M = (Q, \Sigma, \Delta, \delta, \lambda, q_0) \quad (1)$$

where Q is the finite set of all states, Σ and Δ are the finite sets of input and output symbols, respectively, δ is the state transition function (a function specifying the next state for each pair of current state $p \in Q$ and input symbol $a \in \Sigma$, that is, $\delta(p, a) = q \in Q$, λ is the output function defining the output symbol in the form $\lambda(p, a) = b \in \Delta$, and q_0 is the initial state.

Since automata are typically represented by their state transition tables, this seems to be the most natural representation to be adopted. In the proposed approach, each automaton is represented by an array in which each row denotes a possible current state, each column an input symbol, and each entry is a pair of the next state and the corresponding output symbol. Evolution takes place in a population Pop of *pop-size* machines μ . Formally, for each generation t we have

$$Pop(t) = [\mu^1(t), \mu^2(t), \mu^3, \dots, \mu^{\text{pop-size}}(t)]. \quad (2)$$

Each automaton μ^i is represented by a matrix (table) in which each entry is a pair of the form “next state|output”, that is

$$\mu^i = [a_{j\ k}^i \in Q | b_{j\ k}^i \in \Delta] \quad (3)$$

for $i = 1, 2, \dots, \text{pop-size}$, $j = 1, 2, \dots, n^i$, and $k = 1, 2, \dots, \dim(\Sigma)$, where n^i stands for the number of states of the i -th automaton and $\dim(\Sigma)$ gives the number of different input symbols. For consistency, it is assumed that all the states and input symbols are labeled in a fixed order, with the first row of each STT corresponding to the current state being the initial state q_0 . A population of automata of various sizes is illustrated in Fig. 1.

3.2 Population Initialization and Consistency

In the beginning, a population $Pop(0)$ of automata is generated randomly, as it is usual with evolutionary procedures. However, due to the limited size of the population, it is important that one impose limits on the size of each automaton, that is, make

$$n_{\min} \leq n^i \leq n_{\max} \quad \text{for } i = 1, 2, \dots, \text{pop-size}. \quad (4)$$

The limits should be wide enough to allow for a global search, but small enough for the sake of efficiency. To avoid operating on unmeaningful automata,

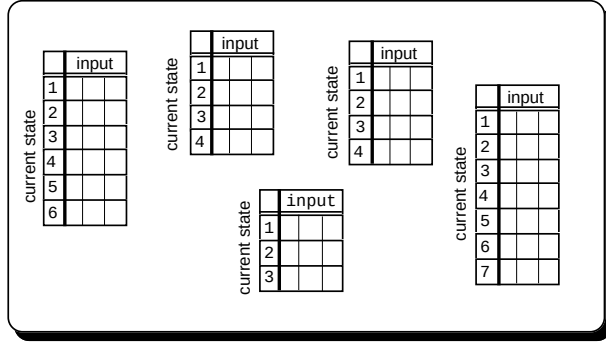


Fig. 1. Example of a population of automata with different sizes.

it is also crucial to make sure that the automata are consistent, that is, that they not include transitions to non-existing states. This is trivial when generating the initial population, since it suffices to choose the number of states n^i first, and then limiting the choice of the next states $a_{j_k}^i$ in the range $[1, n^i]$.

3.3 Evaluation Function

Careful selection of the evaluation and fitness functions are essential for attaining good performance. Such a choice is, of course, problem-dependent, but a few points should be emphasized. First, common sense and economical reasoning favors the well-known *Occam's razor* principle, by which “the simpler, the better”. Accordingly, the literature registers applications of different measures to favor small, simple individuals, including the minimum description length [8] and the minimum message length [1]. This paper uses a simpler approach based on the summation of individual costs.

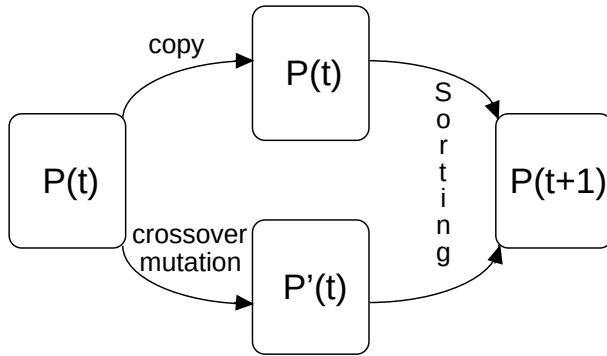


Fig. 2. Generation model.

3.4 Generation Model and Genetic Operators

A variation of the continuous generation model was employed. At the t -th generation, the population $Pop(t)$ is initially duplicated, and then an auxiliary population $Pop'(t)$ of the same size is produced by either crossover or mutation. Denoting by $0 \leq \rho \leq 1$ the ratio of crossover, in $Pop'(t)$

$$n_{\text{crossover}} = \text{int}\left(\frac{\rho \times \text{pop-size}}{2} + 1\right) \times 2 \quad (5)$$

children automata are produced by crossover, whereas the remaining individuals result from mutation. Finally, the next population $Pop(t+1)$ results from selecting the best automata from the pool of $2 \times \text{pop-size}$. This generation model is shown in Fig. 2.

Crossover Operator A 2-point crossover operator was applied as follows. First, a pair of automata in the population is chosen with probability proportional to their fitness, using the well-known *roulette-wheel method*. Crossover then takes place by exchanging groups of contiguous rows in the automata's STTs. Both crossover points in each automaton are chosen randomly. This is repeated until the number of children reaches the value specified in Eq. (5).

The procedure allows for dynamic variation of the number of states of the automata. However, as in GP, in practice most of the times the crossover operation results in bad individuals, and may even produce inconsistent automata which refer to non-existing states or have states which are never reached, for example. This suggests that the crossover ratio ρ should be kept small for the sake of efficient search.

Mutation Operator A simple mutation operator can be defined by selecting cells to undergo mutation and then changing some of the corresponding entry values within a certain range. Using fitness-proportional selection, at first $(\text{pop-size} - n_{\text{crossover}})$ automata are chosen to undergo mutation. Next, a percentage p_{mut} of the STT entries in each selected automaton are chosen randomly. Finally, in each selected entry (a table cell) a single symbol is changed randomly.

4 Evolution of Turing Machines

4.1 Problem Relevance

It is a well-known fact that Turing machines (TMs) are universal models of computer programs, in the sense that any computer program can be emulated by a convenient TM. The opposite is also obviously true, since one can easily translate any given TM into a more conventional computer program.

Assume now that one wants to generate a computer program or an algorithm to solve a certain processing task, and that all that is given is a limited collection of possible inputs to the program and desired (corresponding) outputs. The

approach considered in this paper is an indirect one: 1) First, a TM tha can solve the task, learning from the example patterns, is evolved; 2) Later, by observing the structure and behavior of successfully evolved TMs while they process the data, the final computer programs or algorithms are produced. Only the first part is considered in this paper, since the latter is usually straightforward.

Therefore, generating TMs from examples is actually an approach for the automatic generation of programs and algorithms. Furthermore, since TMs are very simple automata consisting of extremely simple elements, this approach seems more natural than the one of GP, where elemental functions are assumed to be known beforehand.

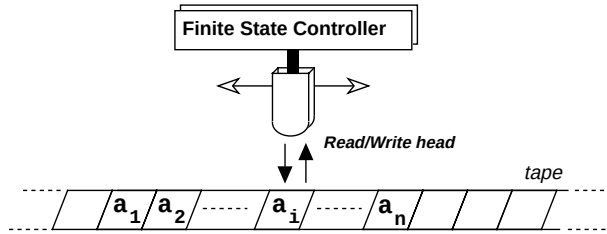


Fig. 3. Turing machine.

4.2 Turing Machines

Consider now the special case in which each automaton in the population represents a Turing Machine. A TM is a general automaton able to read and write in arbitrary positions of a tape, as shown in Fig. 3. In the basic model, there is only one unidimensional tape divided into cells of same size, each one able to contain a symbol in a specified tape symbol set Γ . The TM features a head capable of reading the symbol immediately below it, changing it, and moving to the right or left by one cell. At first, the head is positioned on the beginning (leftmost symbol) of an input string of symbols belonging to the input symbol set $\Sigma \subset \Gamma$. The set $\Gamma - \Sigma$ contains the symbol $\mathcal{B}$, the blank character. Generally the input string is surrounded by blank characters on both sides, and the tape can grow without limit in both directions. In practice, however, the tape must be limited. The head is controlled by a finite state machine of states in the set Q , initialized to the state q_0 . For each current state $p \in Q$ and input symbol $a \in \Sigma$, the “next move function” δ defines the next state q , the symbol b to replace a and the head movement, that is

$$\delta(p, a) = (q, b, D) \quad \text{where } D \in L, R. \quad (6)$$

The function δ does not necessarily has to be defined for all the combinations (p, a) , and it may also specify any or none of the terms on the right-hand side

of Eq. (6). If the direction is not specified, for example, it means that the head simply should not move. Similarly to Eq. (1), a TM can be denoted as

$$TM = (Q, \Gamma, \Sigma, \delta, q_0, \mathcal{B}, F) \quad (7)$$

where F is the set of final states. At any instant, the state of a TM can be given by its instantaneous description, as follows:

$$TM : (p, a_1 a_2 \dots \overset{\downarrow}{a_i} \dots a_n)$$

where p is the current state and the next symbol to be read is indicated by the arrow.

Table 1. Training tapes for the two-symbol sorting problem.

Pair #	Input	Target
1	bbaabaaaaa	aaaaaaabbb
2	aabbbabaaa	aaaaabbbb
3	aabaabaaaa	aaaaaaaabb
4	aaaababaaa	aaaaaaaabb
5	abaababbba	aaaaabbbbb
6	aababaaabb	aaaaabbbbb
7	bbaabaaaaa	aaaaaaabbb
8	abaabaabba	aaaaabbbbb
9	aababbaaba	aaaaabbbbb
10	abaababbba	aaaaabbbbb

4.3 Experimental Results

Two cases were investigated, namely, a two-symbol sorting problem and a unary proper subtraction problem. In both cases, starting from a population of randomly-initialized single-tape TMs, the objective was to generate general TMs for the tasks. For training, each problem used a few pairs of input tapes and respective desired output tapes. It was assumed that all the non-blank input symbols were gathered contiguously on the center of tapes of length equal to 30 characters, with blank characters on both sides.

For each problem, a population of 100 TMs was generated randomly with maximum number of states equal to 10, and was set to evolve for 1,000 generations. In the temporary population, 10 TMs were generated by crossover, whereas the remaining 90 TMs were produced by mutation (that is, crossover ratio $\rho = 0.1$). The maximum number of states exchanged by crossover per time was fixed at 3, and children TMs with more than 20 states were not allowed ($n_{\max} = 20$).

The execution of each TM terminates when one of the following conditions is reached: 1) the head advances beyond the tape’s limits; 2) the TM fails to stop within a maximum number of steps; 3) the TM stops acting; or 4) the TM refers to an non-existing state.

It is also important to have a high mutation rate early in the run to allow good exploration of the search space, and reduce the rate as evolution advances. Accordingly, the mutation rate p_{mut} was scheduled in such a way that a symbol in each entry of each STT to undergo mutation was mutated during the first 500 generations, while only 10% of the cells had a symbol mutated during the last 100 generations.

To evaluate each TM, first a cost function was defined as a weighted sum considering the number of wrong positions with respect to the learning tapes, the difference between the number of symbols between the input and desired output tapes, the complexity of each TM (number of states, number of symbols in the STT), etc., for all input training tapes and their corresponding target output tapes. Finally, the obtained cost values were linearly coded into fitness values between 1 and 10.

Table 2. Best results for 10 runs for both the two-symbol sorting. The obtained TMs were successfully tested against a set of tapes not used for learning. The best TMs were shown to be general solutions.

Run	States	Cost	No. of Gen.	Time (s)
1	4	310	916	143
2	6	321	947	132
3	4	311	860	127
4	5	315	856	126
5	3	310	871	135
6	3	310	801	119
7	6	312	937	144
8	3	313	740	116
9	3	310	802	123
10	4	310	850	123

Two-Symbol Sorting The objective of the sorting problem is to generate an optimal TM able to sort random tapes containing the symbols a and b , that is, to learn the relationship $a < b$ from only 10 pairs of training tapes, as shown in Table 1. It is important to note the enormous dimensions of the space of TMs. Even if only 4-state TMs with 3 input-output symbols (if blanks are also included) are considered, there are $36^{12} \approx 4.7 \times 10^{18}$ machines.

Results for 10 runs are shown in Table 2. For the sorting problem, despite the different number of states and/or complexity, the best TMs in all runs were able

to sort correctly not only the 10 tapes used for learning, but also the 20 others used for testing, suggesting that general TMs were obtained. We can, therefore, conclude that, more than only generating specific TMs for a particular problem, the proposed approach succeeded in synthesizing algorithms. In fact, in this experiment the best TMs implemented the well-known “bubble sort” algorithm.

Table 3. Training tapes for the unary proper subtraction problem.

Pair #	Input	Target	Meaning
1	1111111101	1111111	$8 - 1 = 7$
2	1111111011	11111	$7 - 2 = 5$
3	1111110111	111	$6 - 3 = 3$
4	1111101111	1	$5 - 4 = 1$
5	1111011111		$4 - 5 = 0$
6	1110111111		$3 - 6 = 0$
7	1101111111		$2 - 7 = 0$
8	1011111111		$1 - 8 = 0$

Proper Unary Subtraction The second problem deals with a TM containing only 0s, 1s and blank characters. There are two strings of contiguous 1s denoting two unary numbers, in such a way that a natural number n is represented by a string of n 1s. It is assumed that there are two numbers separated by a single 0 character, and the objective is to generate a TM able to perform proper subtraction, resulting in a tape containing only a single string of 1s, corresponding to the difference between the numbers. Only 8 pairs of tapes are assumed to be available for training, as shown in Table 3. By “proper” subtraction it is meant that there should be no 1s remaining in the tape when the first operand is smaller than or equal to the second one.

The results of 10 runs are shown in Table 4. Once again the generated TMs succeeded in producing correct results for tapes other than the ones used for training, indicating that general solutions were generated.

5 Enhanced Evolutionary Approach

Although the approach described above seems to suffice for simple problems, the procedure does not scale up well for more complex tasks. In fact, such a non-satisfactory performance was somewhat predictable, since the proposed crossover and mutation operators described above were devised without taking into account particular characteristics of automata generation.

As described, the mutation operator performs a type of local search by changing one or a few symbols of the STT representing an automaton, while crossover

Table 4. Best results for 10 runs for the unary proper subtraction problems.

Run	States	Cost	No. of Gen.	Time (s)
1	5	3850	932	101
2	5	3851	842	115
3	3	3850	810	86
4	5	3850	837	102
5	6	3857	936	94
6	6	3850	917	97
7	10	3861	976	120
8	8	3862	995	110
9	6	3851	918	113
10	8	3859	993	124

is the only way to change the number of states, carrying out a more radical change in the search space. However, since random crossover is likely to result in unmeaningful automata, it is not an effective procedure. As a consequence, the number of states of each automaton remains constant in most cases and, therefore, valuable processing time is spent on local evolution of hopeless automata. For example, suppose that the initial population contains 10 automata for each size from 5 to 14 states, in a total of 100 automata, and that the number of states of the optimal automaton for the problem at hand is, say, 12. If the number of states does not change while evolution takes place, then only 10% of the automata in the population will ever have any chance of ever reaching the optimal configuration.

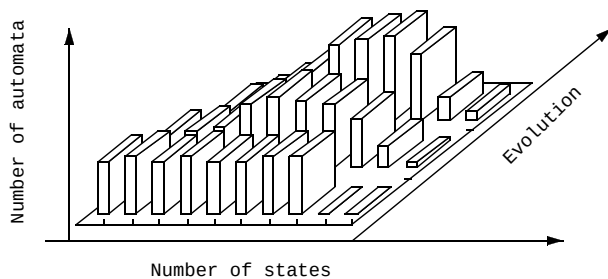


Fig.4. Population shift approach.

5.1 Population Shifting Approach

In this approach, performance statistics are collected at each generation, and operators are devised in such a way to favor the appearance of automata of sizes

close to the sizes of the best performing automata in the previous generation. A pictorial description of this idea is shown in Fig. 4. Beginning from a situation in which the sizes of the automata in the population are approximately uniformly distributed in a given range, the number of automata of each size changes in such a way to concentrate the search on the region of more likely improvement. This approach is implemented by using a mutation operator, as described below.

5.2 New Mutation Operators

First of all, crossover was dropped out due to its inefficiency. The idea of crossover is based on the “building block hypothesis”, which is very unlikely to hold in the case of automata generation. Instead, three mutation operators were developed.

- Mutation¹: This is the same mutation operator described above, with the exception that now multiple changes on the same cell are allowed.
- Mutation²: This operator allows dynamic changes to the number of states of a given automaton. After an automaton is selected to undergo this type of mutation, a state is deleted from or added to it in such a way to make its size closer to the size of the best automata of the previous generation. In the case of deletion, the state to be deleted is chosen as the least visited one, all references to the deleted state are changed randomly inside the new state range, and the corresponding STT becomes one row shorter. In the case of addition of a state, the new state is appended to the end of the STT, the corresponding cells are filled with random values, and one of the cells corresponding to other states is changed to refer to the newly added state.
- Mutation³: This operator kills a given automaton and generates another one with number of states determined from the performance statistics of the previous generation. This determination is done by using a roulette-wheel with slots corresponding to possible number of states, with areas proportional to the average performance of the automata in the previous generation.

5.3 1/5 Heuristics

In the first implementation of the evolutionary approach, the rate of execution of the operators were kept constant for the sake of simplicity. A better approach (at least, in principle) is to set initial rates and have them adapted as evolution takes place, in such a way to optimize performance. Here, this is done by borrowing Rechenberg’s idea of “evolution window” [10, 11].

This idea, developed in the field of “evolution strategy”, states that successful evolution only takes place when the mutation rate falls within a narrow band named evolution window. The “1/5 rule” provides one heuristical procedure to attempt to keep the mutation rate within the evolution window.

In concrete terms, the history of the application of each mutation operator is kept up to date in terms of success rate, that is, the percentage of the total number of applications in which the resulting automaton was fitter than its parent prior to mutation. Whenever this value tends to go above 1/5, the corresponding

mutation rate is changed in such a way to make the search process more global; conversely, success rate below 1/5 is used as an indication that search should be made more local. This 1/5 value is close to theoretical optimal values for a few specific problems, but has been widely applied to a number of general tasks.

In our implementation, making the search more local was interpreted as increasing the probability of mutation¹ while decreasing the probabilities of both mutation² and mutation³. These changes were implemented by addition or subtraction of a fixed value. The enhanced evolutionary approach is summarized in Fig. 5.

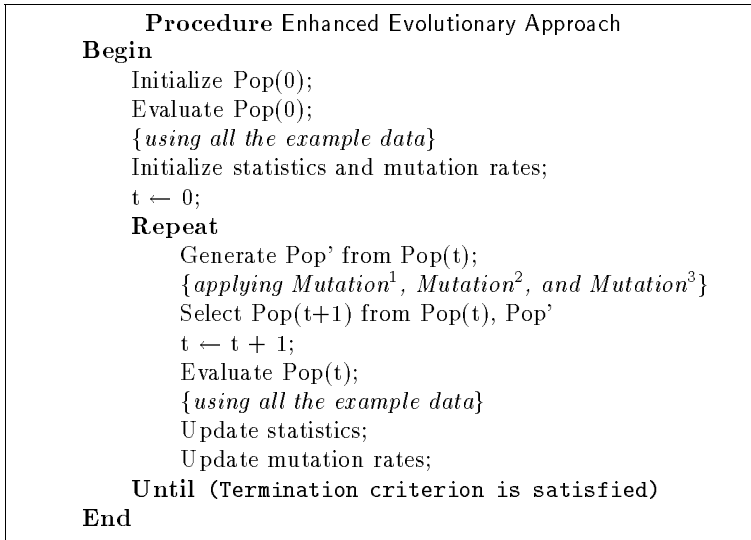


Fig. 5. Summary of the enhanced evolutionary approach.

5.4 Experimental Results

Language acceptance problems were used to evaluate the enhanced evolutionary approach, since such problems often require automata more complex than those needed for the simple problems used in the previous section. Three simple problems, namely, the recognition of a regular, a context-free, and a context-sensitive languages, were set up for experiments.

In all cases, populations of 100 automata were allowed to evolve for 1,000 generations. At the outset, the probabilities of the mutation operators were set to 70%, 25%, and 5% for mutation¹, mutation², and mutation³, respectively, and up to 50% of the cells of a given automaton were allowed to be changed. These values were later adapted by the 1/5 heuristics.

For the three language acceptance problems, the input alphabet was set as $\{a, b, c, \mathcal{B}\}$, where $\mathcal{B}$ represents the blank space. A TM capable of recognizing a language should append a c after the input string if it belongs to the language. Otherwise, it should stop leaving the tape unaltered. In all experiments, 40 tapes were used for search, only half of them containing strings belonging to the languages to be recognized. The other tapes were generated randomly.

Target Languages The first problem was to generate TMs to recognize the regular language awb , where $w \in \{a, b\}^*$. That is, a string should be accepted if it begins with an a , ends with a b , and does not contain any blank space or c . Input strings of length 20 were employed.

The second problem used the context-free language $a^n b^n$, where $n \geq 1$ was employed, and input strings with 40 symbols each were used. Finally, the language $a^n b^n a^n$, for $n \geq 1$, was used as a simple context-sensitive language, and the input strings had 60 symbols each.

Comparative Results Experiments were performed using the simple and the enhanced evolutionary approaches, and results for 100 runs of each approach are shown in Table 5. The results indicate the success ratio in 100 runs and the average generation in which the problems were solved. As expected, the extended approach outperformed the simple one, but the results were still far from the ideal success ratio of 100%. It is reasonable to expect to improve the results by starting from TMs generated by some simple heuristical method and/or by post-processing the obtained TMs using a conventional technique.

Table 5. Comparative results for language acceptance problems.

Problem	Evolutionary Approach			
	Simple		Enhanced	
	succ. (%)	gen.	succ. (%)	gen.
awb	9	660	82	500
$a^n b^n$	41	387	62	429
$a^n b^n a^n$	1	259	38	616

6 Conclusion

In this paper, an approach to evolve automata represented by their state transition tables was proposed. By operating directly on STTs, the method allows for easy analysis and conversion of the evolution results into hardware implementable logic. Two approaches were investigated, namely, a simple GA approach and another implementation employing heuristics of known evolutionary

computing methods. Experimental results for several problems of generation of TMs from examples indicated the feasibility of the proposed methods, particularly the second one.

It may be argued that the second proposed approach can be thought of as an EP method, since there is not crossover and only sophisticated mutations. However, the proposed approach employs neither tournament selection nor Gaussian mutations which are typical of EP methods. Furthermore, the population shift approach plays a significant role in concentrating the automata search in the regions of more likely improvement.

While the results indicate it is possible to evolve TMs from examples, the proposed idea is still inefficient and needs to be enhanced. Many runs failed to solve the target problems in the number of generations allocated. This was caused in part because the number of training examples provided was very low in comparison with the size of the search space. It was necessary to limit the number of examples due to the rapid increase in the processing time, since all examples had to be processed to calculate the fitness of a given TM. One possible via for improvement is devising an estimate fitness measure that does not require the machine to process each training example until it stops or runs out of time. Other avenues for research include the use of more sophisticated initialization procedures, better operators, and post-processing using conventional techniques.

References

1. Clelland, C. H., Newlands, D. A.: PFSA modelling of behavioural sequences by evolutionary programming. In R. J. Stonier and X. H. Yu, *Complex Systems: Mechanism for Adaptation*, IEEE Press (1994) 165–172
2. Collins, R., Jefferson, D.: Ant farm: toward simulated evolution. In C. G. Langton et al., *Artificial Life II*, Addison Wesley (1991)
3. Fogel, D. B.: An introduction to simulated evolutionary optimization. *IEEE Trans. Neural Networks* **5** (1994) 3–14
4. Fogel, D. B.: Evolving behaviors in the iterated prisoner's dilemma. *Evolutionary Computation* **1** (1993) 77–97
5. Fogel, L. J., Owens, A. J., Walsh, M. J.: *Artificial Intelligence through Simulated Evolution*, John Wiley (1966)
6. Goldberg, D. E., Deb, K., Korb, B.: Don't worry, be messy. *Proc. Fourth Int. Conf. Genetic Algorithms* (1991) 24–30
7. Holland, J. H.: *Adaptation in Natural and Artificial Systems*, Univ. of Michigan Press (1975)
8. Iba, H., Kurita, T., deGaris, H., Sato, T.: System identification using structured genetic algorithms. *Proc. 5th Int. Conf. Genetic Algorithms* (1993)
9. Koza, J. R.: *Genetic Programming—On The Programming of Computers by Means of Natural Selection*, MIT Press (1992)
10. Rechenberg, I.: *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*, Frommann-Holzboog Verlag (1973)
11. Rechenberg, I.: Evolution strategy. In J. M. Zurada and R. J. Marks II and C. J. Robinson, *Computational Intelligence: Imitating Life*, IEEE Press (1994) 147–159

Genetic Algorithms: Minimal Conditions for Convergence

Alexandru Agapie

Computational Intelligence Lab., National Institute of Microtechnology
PO Box 38-160, 72225, Bucharest, Romania
E-mail: agapie@imt.pub.ro

Abstract. This paper is concerning the finite, homogenous Markov chain modeling of the binary, elitist genetic algorithm (EGA) and provides a set of minimal sufficient conditions for convergence to the global optimum. The case of a GA where each population would be allowed to mutate only a small number of bits has not been covered yet by the GA's literature, although it commonly appears in practice. The main result presented here shows that the condition of the one-step transition probability by mutation between two arbitrary strings being larger than zero can be relaxed in the sense that it is also sufficient to achieve the transition by a chain of small mutations. Consequently, even one-bit mutations would be sufficient to make the GA globally convergent, because they can be chained to achieve a multi-bit mutation. All this study is performed with respect to the theory of non-negative matrices and their relationship to Markov chains.

1. Motivations for a new approach

Genetic algorithms (GAs) - as introduced by Holland in the seventies [4] - are probabilistic techniques for optimization, operating on populations of strings (called *chromosomes*) coded to represent some underlying parameter set. Some operators, called *selection*, *crossover* and *mutation* are applied to successive generations of chromosomes to create new, better valued populations.

When the schema theorem - established in [4], and consecutively revised [13] - proved to be insufficient for analyzing the GAs, most of the theoretical approaches moved onto the convergence theorems of stochastic processes, and especially to the ergodic theorems from Markov chain theory. The first attempts were concerning the theory of finite, homogenous Markov chain only [2, 3, 5, 14], but they evolved also to the study of infinite [10], or inhomogenous models [1]. However, the finite, homogenous algorithm is most commonly used in practice, so the improvement of its theoretical fundamentals is strongly required.

This study deals with the model of the binary, elitist genetic algorithm (EGA) and provides some sufficient conditions for convergence to the global optimum, regardless of the initialization. It provides a generalization - based on some realistic assumptions - of the similar results (*Theorems 6, 7*) presented in [9].

* All over this paper the term *elitist* is associated to a canonical GA maintaining the best solution found over time, without using it to generate new individuals.

Against the well known hypothesis “ M (the mutation matrix) is positive” , herein is proposed a weaker assumption on the mutation operator, namely: “ M is irreducible with positive diagonal elements”. For example, the irreducibility is satisfied by assuming that mutation can invert only a small number of bits in a generation - thus not all the bits, as required by the positive M condition. Roughly speaking, this paper covers the case of a GA where each population of chromosomes would be allowed to mutate at max one bit, which previous papers do not cover. One must admit that this representation is much closer to the practical GAs. Additionally, the crossover matrix will be required to have positive diagonal elements, also. It is shown that, even with this weaker assumptions, the convergence result for EGA still holds.

In fact, the convergence path from [9]:

EGA $\rightarrow$ M positive $\rightarrow$ M primitive $\rightarrow$ Ergodic theorem for reducible EGA $\rightarrow$ Convergence of EGA

is replaced by:

EGA $\rightarrow$ M irreducible $\rightarrow$ Sub-optimal states are inessential $\rightarrow$ Zero limit theorem for the sub-matrix of inessential states $\rightarrow$ Convergence of EGA.

The major tool used in this study is the *Zero limit theorem for the sub-matrix of inessential states* - from [11]. As one can notice, the ergodic behavior of the EGA model is *not* available any more. However, this fact does not affect the convergence of the algorithm to the optimal states - as a sufficient condition for convergence is: “For each sub-optimal state i , the sum of transition probabilities to all the sub-optimal states tends to zero”. One will see how this statement ensures the desired convergence - using the triangular form of the transition matrix, which is still available for this approach.

A new approach in Markov chain analysis of EGA convergence must first explain its utility, as the convergence for the elitist algorithm was already proved, under the positive mutation matrix assumption [9]. At a closer look, this statement of positivity proves to be not so realistic: consider the extreme case of two states* - say i and j - placed at maximal *Hamming distance* one from the other. Recall that the Hamming distance - denoted H - is defined as the number of different bits between two binary strings. Therefore, $H_{ij}=n\ell$, where n denotes the GA's population size and ℓ - the chromosome's string length. For example let $i=(111...1)$ and $j=(000...0)$, both of length ℓ . One can easily compute (see e.g. [1], [9]) the transition probability from state i to state j , due to mutation only, as $m_{ij}=p_m^{n\ell}$ (p_m denotes the usual mutation probability). If the parameters are set at $p_m \sim 0.1$, $n \sim 20$ and $\ell \sim 20$ (which are very permissible assumptions) one will obtain $m_{ij} \sim 10^{-400}$. From a practical point of view, the positivity of this value is non-realistic: actually, a value of 10^{-400} is rather zero than positive. This statement is supported by statistical considerations - for a detailed discussion on this subject see [6] or [7]. Thus, the problem of relaxing the positivity

* All over this paper a state of the Markov chain will be associated to a population of chromosomes, not to a single chromosome.

assumption on M is a worthwhile challenge, and the objective of this paper is to show that this assumption is a sufficient condition for convergence, but not a necessary one.

Section 2 describes the Markov chain formalism used in the paper, and the GA model - as introduced in [9]. This formalization, together with some classical results from the Markov chain theory, are used in Section 3 for deriving the minimal conditions for convergence. A critique of this attempt completes the paper.

2 The Markov Chain Model

There are several GA approaches based on the ergodic theorem for *irreducible* transition matrix [1, 8, 12] but the idea of applying the ergodic theorem for *reducible* transition matrix is due to Rudolph [9], yielding a convergence result for the elitist model. Going deeper in the Markov chain theory, this paper states that the ergodicity (and thus the reducible form of the transition matrix) is not a necessary condition for the EGA's convergence. Anyway, it remains a sufficient condition for convergence.

This paper entirely adopts the formalism from [9], namely:

1. The optimization task is assumed as: $\max \{f(x) \mid x \in \{0, 1\}^\ell\}$, where f , the *fitness* function, satisfies $f(x) > 0 \forall x$;
2. The simple (canonical) GA consists of an n -tuple (called *population*) of binary strings x_i , each of them of length ℓ ;
 - the transition between successive populations is performed by applying the genetic operators of *crossover*, *mutation* and *selection* - in this order; crossover and mutation are applied with some probabilities - p_c , res. p_m - which are considered fixed all over the algorithm, in the homogenous case;
 - the canonical GA's transition matrix, denoted P , can be described as a product of the matrices corresponding to crossover, mutation and selection: $P = CMS$;
3. The EGA is modeled by a Markov chain in the following manner:
 - the state space is of cardinality $2^{(n+1)\ell}$ - a state is a possible population of the EGA, this is, the GA population supplemented by a *super individual* which does not take part in the evolutionary process. The super individual is artificially maintained from a generation to another, being subject of change only when the algorithm produces a better chromosome. The k chromosome from the current population at time t (let this population be denoted by i) can be accessed by the projection function, $\pi_k^t(i)$. For notational convenience the super individual is placed on the leftmost position in the $(n+1)$ -tuple (position 0, thus accessed by $\pi_0^t(i)$, from the population i , at time t). The transition probabilities of those states containing the same super individual are assumed to be listed one below the other in the transition matrix associated to the EGA (say P^*), and the better the super individual's fitness the higher the position of the corresponding state in P^* ;
 - since the super individual is not affected by crossover, selection and mutation, P^* (the transition matrix associated to the EGA) can be computed as:

$$P^+ = \begin{pmatrix} C & & \\ & C & \\ & & \ddots \\ & & & C \end{pmatrix} \begin{pmatrix} M & & \\ & M & \\ & & \ddots \\ & & & M \end{pmatrix} \begin{pmatrix} S & & \\ & S & \\ & & \ddots \\ & & & S \end{pmatrix} U =$$

$$= \begin{pmatrix} CMS & & \\ & CMS & \\ & & \ddots \\ & & & CMS \end{pmatrix} U,$$

where C , M and S are $2^{n'} \times 2^{n'}$ size matrices, P^+ is of size $2^{(n+1)'} \times 2^{(n+1)'}$ and U is the so called *upgrade* matrix defined as follows: let $b = \arg \max \{f(\pi_k(i)) \mid k=1, \dots, n\} \in \{0, 1\}^{\ell}$ denote the best individual of the population at any state i (regardless of time), excluding the super individual. Then, by definition, $u_{ij}=1$ if $f(\pi_0(i)) < f(b)$ where $j = (b, \pi_1(i), \pi_2(i), \dots, \pi_n(i))$, otherwise $u_{ij}=0$ (i and j are two possible states for the Markov chain, this is, two populations of $(n+1)$ chromosomes). Actually, the upgrade matrix is acting like a first position modifier, by copying the new best individual - in case of improvement/generation, and resting inactive otherwise (see [9] for details). Therefore, its structure is:

$$U = \begin{pmatrix} U_{11} = I & & & \\ U_{21} & U_{22} & & \\ \vdots & \vdots & \ddots & \\ U_{2^{\ell},1} & U_{2^{\ell},2} & \dots & U_{2^{\ell},2^{\ell}} \end{pmatrix}.$$

U is composed of 2^{ℓ} lines (and columns) of $2^{n'} \times 2^{n'}$ -size blocks.

Assuming the existence of only one global optimum for the problem to be solved, it yields $U_{11}=I$ - the $2^{n'} \times 2^{n'}$ unit matrix - whereas all U_{aa} with $a \geq 2$ are unit matrices with some zero diagonal entries. Therefore, the transition matrix for the EGA can be derived as:

$$P^+ = \begin{pmatrix} P & & \\ & P & \\ & & \ddots \\ & & & P \end{pmatrix} \begin{pmatrix} I & & \\ U_{21} & U_{22} & \\ \vdots & \vdots & \ddots \\ U_{2^{\ell},1} & U_{2^{\ell},2} & \dots & U_{2^{\ell},2^{\ell}} \end{pmatrix} =$$

$$= \begin{pmatrix} P & & & \\ PU_{21} & PU_{22} & & \\ \vdots & \vdots & \ddots & \\ PU_{2^\ell,1} & PU_{2^\ell,2} & \dots & PU_{2^\ell,2^\ell} \end{pmatrix}.$$

4. The *convergence definition*: let $Z_t = \max \{f(\pi_k^t(i)), k=1, \dots, n\}$ be a sequence of random variables representing the best fitness within a population represented by state i at time t . A GA converges to the global optimum if and only if $\lim_{t \rightarrow \infty} P\{Z_t = f^*\} = 1$, where $f^* = \max \{f(x) \mid x \in \{0, 1\}^\ell\}$ is the global optimum of the problem.

Note 1. It was proved [8], that the maximal number of different populations is actually less than $2^{n'}$, namely:

$$n_{\text{pop}} = \binom{n + 2^\ell - 1}{2^\ell - 1}. \text{ But for the sake of convenience the cardinality of the state}$$

space will be maintained at the value $2^{n'}$, all over this article. This assumption is equivalent to considering ordered populations of chromosomes in the GA (instead of non-ordered ones) and does not affect the convergence behavior of the EGA model, depicted below.

Definition 1.

- Let i and j be two states from a Markov chain (with transition matrix P). We say that i *leads to* j (denote $i \rightarrow j$) if there is a chain of states $(i, i_1, i_2, \dots, i_k, j)$ s.t. $p_{i,i_1} p_{i_1,i_2} \dots p_{i_k,j} > 0$. (Equivalently: it exists an integer m s.t. the (i,j) element from P^m is positive.)
- A state i from a Markov chain is said to be *essential* if $i \rightarrow j$ implies $j \rightarrow i$, for all j . Otherwise, (if there exist some j s.t. $i \rightarrow j$ but $j \not\rightarrow i$) i is said to be *inessential*.
- A non-negative matrix $A=(a_{ij})_{i,j=1,\dots,n}$ is said to be *stochastic* if $\sum_{j=1,\dots,n} a_{ij} = 1$, for each $i=1,\dots,n$.
- A square matrix $A=(a_{ij})_{i,j=1,\dots,n}$ is said to be *irreducible* if for each pair of states $i, j \leq n$ there is an integer m s.t. the (i,j) element from A^m is positive. (Equivalently, A is irreducible if $i \rightarrow j$ for each pair (i, j) .) Otherwise A is said to be *reducible* (see [11, p. 12, 18] for definition 1. a, b, c, d).
- A square matrix is said to be *diagonal-positive* if all its main diagonal elements are positive.
- A state i is said to be *optimal* (for the EGA problem) if the corresponding population contains the globally optimal individual on its first position - $\pi_0(i)$ (the globally optimal *super individual* from [9]). Otherwise, i is said to be *sub-optimal* (for the EGA problem).

3 Convergence results

The following lemmas set up the foundations of the EGA convergence upon the irreducible mutation matrix assumption. The first one is a simple result involving only non-negative matrices, while the second makes the connection between non-negative matrices and the GA's model - identifying the sub-optimal populations from a GA with the inessential states in the Markov chain.

Lemma 1. Let C , M and S be stochastic matrices, where M is irreducible and S , C are diagonal. Then the product CMS is irreducible. Supplementary, if M is diagonal-positive then CMS is diagonal-positive.

Proof: Let us consider first the product MS . M is irreducible implies that for each pair i, j there is a chain of states $(i, i_1, i_2, \dots, i_k, j)$ s.t. $m_{i,i_1} m_{i_1,i_2} \dots m_{i_k,j} > 0$. Since S is diagonal positive, one gets $m_{i,i_1} s_{i_1,i_1} m_{i_1,i_2} s_{i_2,i_2} \dots m_{i_k,j} s_{j,j} > 0$, too. As $m_{i,j} s_{j,j} \leq (ms)_{i,j}$ for every pair i, j , it yields $(ms)_{i,i_1} (ms)_{i_1,i_2} \dots (ms)_{i_k,j} > 0$, hence MS is irreducible. The left product between a diagonal-positive matrix (C) and an irreducible one (MS) is still irreducible (similar proof), thus CMS yields irreducible.

Supplementary, if M is diagonal-positive, (and S , C are diagonal-positive by assumption) it is obvious that CMS is diagonal-positive, too.

Lemma 2. Let P^* be the transition matrix of the EGA. If $P=CMS$ is irreducible and diagonal-positive, then a state i is sub-optimal for the EGA problem if and only if it is inessential.

Proof: One will treat separately the two implications:

a $\square \square i$ is sub-optimal $\Rightarrow i$ is inessential;

b $\square \square i$ is optimal $\Rightarrow i$ is essential;

In order to prove (a), let $i=(\pi_0(i), \pi_1(i), \dots, \pi_n(i))$ be a sub-optimal state, and let $max_i = \max \{f(\pi_k(i)), k=1, \dots, n\}$. Two different situations may occur:

a.1) $f(\pi_0(i)) \geq max_i$,

a.2) $f(\pi_0(i)) < max_i$.

Case (a.1) allows two possibilities as well:

a.1.1) there is a state k s.t. $f(\pi_0(k)) < max_k$ and $p_{ik} > 0$ (note that the last inequality implies $\pi_0(k) = \pi_0(i)$). Thus, according to the definition of the upgrade matrix U , one has $u_{kj} = 1$, where $j=(b, \pi_1(i), \dots, \pi_n(i))$ and $b = \arg \max \{f(\pi_s(i)), s=1, \dots, n\}$. Hence $p_{ij}^* \geq p_{ik} u_{kj} = p_{ik} > 0$, thus $i \rightarrow j$.

Next, we prove that $j \rightarrow i$: as j is placed in a class of states (by size $2^{n'}$) superior to that of i (because $f(\pi_0(j)) = max_k > f(\pi_0(i))$), and - by definition - matrix P^* is zero above the block diagonal, there will be no possibility for $j \rightarrow i$.

a.1.2) there is a chain of states $(i, i_1, i_2, \dots, i_s, k)$ s.t. the product $p_{i,i_1} p_{i_1,i_2} \dots p_{i_s,k} > 0$, where $i_1, i_2, \dots, i_s$ satisfy $max_{i_1} \leq f(\pi_0(i))$, ..., $max_{i_s} \leq f(\pi_0(i))$, and $\pi_0(i) = \pi_0(i_1) = \dots = \pi_0(i_s)$, and k satisfies $f(\pi_0(k)) < max_k$ and $p_{ik} > 0$ (as in (a.1.1)).

Consequently $u_{i1, i1}=1, u_{i2, i2}=1, \dots, u_{is, is}=1$, thus $p_{i, i1} u_{i1, i1} p_{i1, i2} u_{i2, i2} \dots u_{is, is} p_{is, k} = p_{i, i1} p_{i1, i2} \dots p_{is, k} > 0$, and this case is reduced to (a.1.1). Therefore (a.1) is fulfilled.

Case a.2) If $f(i_0) < \max_i$, let $j = (b, i_1, \dots, i_N)$ (as in a.1.1). One will see that $i \rightarrow j$, yet $j \not\rightarrow i$. Obviously $u_{ij}=1$ and, as $p_{ii} > 0$, it yields $p_{ij}^+ \geq p_{ii} u_{ij} > 0$, so $i \rightarrow j$. As i and j belong to different classes of states, the inverse does not hold, thus i is inessential.

Case (b). First, recall that at least one essential state exists (any Markov chain contains at least one essential state - see [11, p. 16]). From part (a), it yields that this state is optimal. Next, as the class of optimal states is closed (consequence of the zero- above diagonal form of P^*), one can see that any optimal state can **not** lead to an sub-optimal state. Additionally, the behavior of the Markov chain inside the class of optimal states follows the irreducible matrix CMS - thus $i \rightarrow j$ for every pair of optimal states i and j .

Note 2. The hypothesis “ P is diagonal-positive” was used in the proof of case (a.2) only.

Before moving further, let us remind a very important result from the theory of Markov chain and non-negative matrices. This theorem is presented in detail in [11], but an other form (for the so called “non-recurrent” states - one can prove the equivalence *non-recurrent* = *inessential*) may be found also in [6, p. 97], [7] as a corollary of *Doebelin's Formula*.

Let A be the transition matrix of a reducible Markov chain in canonical form, let Q be the square sub-matrix of A associated to the (transitions between the) inessential states and let T be the square sub-matrix associated to the essential states:

$$A = \begin{pmatrix} T & 0 \\ R & Q \end{pmatrix}. \text{ Then the following theorem holds (quoted all over the paper as the:}$$

Zero limit theorem for the sub-matrix of inessential states).

Theorem 1. [11, p. 120] $Q^k \rightarrow 0$ elementwise as $k \rightarrow \infty$, geometrically fast.

Note 3. This result is also a major tool in the proof of the *Ergodic theorem for reducible matrices* ([7, p. 126], [9, Theorem 2]).

In the following, one will see how this result only is sufficient for the EGA convergence, ensuring the absorption of the Markov chain in the set of optimal points (which is a closed class).

The idea behind these convergence results is that even one-bit mutations would be sufficient to make the EGA globally convergent, because they can be chained to achieve a multi-bit mutation. Thus, the condition “ M is strictly positive” (i.e. the one-step transition probability by mutation between two populations being larger than zero) can be relaxed to the condition “ M is irreducible” (i.e. it is sufficient to achieve the transition by a chain of small mutations).

Theorem 2. The EGA modeled by the transition matrix P^* , with matrices C , M , S stochastic and diagonal-positive, and M irreducible, converges to the global optimum, regardless of initialization.

Proof: According to *Lemma 1*, CMS is irreducible and diagonal-positive; by *Lemma 2*, a state from the EGA model is sub-optimal if and only if it is inessential.

Thus $P^+ = \begin{pmatrix} P & 0 \\ R & Q \end{pmatrix}$ with $P=CMS$ a square, irreducible matrix corresponding to

the set of optimal states, and Q a square matrix corresponding to the set of sub-optimal (thus inessential) states.

Then $(P^+)^k = \begin{pmatrix} P^k & 0 \\ R_k & Q^k \end{pmatrix}$ and even if R_k and P^k do not converge to steady

state matrices as $k \rightarrow \infty$ (like in the ergodic case), the zero limit convergence for Q^k - guaranteed by *Theorem 1* - is sufficient for the Markov chain's absorption in the set of optimal states (i.e. the indices of sub-matrix P). Thus $P\{Z_k=f^*\} \rightarrow 0$ as $k \rightarrow \infty$, which is equivalent to $\lim_{k \rightarrow \infty} P\{Z_k=f^*\}=1$. As in [9], the convergence is regardless of initialization (i.e. the choice of the EGA's initial population does not affect the limit behavior). This fact yields from straightforward matrix multiplication:

$p' \begin{pmatrix} A & 0 \\ & \end{pmatrix}_{n \times (n-m)} = \begin{pmatrix} p' A & 0 \\ & \end{pmatrix}_{1 \times (n-m)}$, where p' is an n -length vector

and A is an $n \times m$ matrix.

The following corollary makes the connection to the case of a more realistic GA model - as outlined in Section 1 - this is, to a GA where each population of chromosomes would be allowed to mutate at max a fixed number of bits (say T), less than the string's length.

Recall that H_{ij} stands for the Hamming distance between populations i and j .

Corollary 1. Let the S and C matrices from the Canonical GA model be stochastic and diagonal-positive. Let T , $1 \leq T \leq 2^{n'}$, be an integer. Let the M matrix be stochastic and satisfying $m_{ij}=0$ if $H_{ij} > T$ and $m_{ij} > 0$ elsewhere. Then the corresponding EGA with transition matrix P^* converges to the global optimum.

Proof: We must prove that if M satisfies $m_{ij} > 0$ for each pair (i, j) s.t. $H_{ij} \leq T$, then:
a) M is diagonal-positive and
b) M is irreducible.

Part (a) is obvious, as long as $H_{ii} = 0$ for each i (notice that $T \geq 1$).

For (b): $m_{ij} > 0$ for each pair (i, j) satisfying $H_{ij} \leq T$. Next, let i and j be two populations with $H_{ij} = k$, $1 \leq k \leq 2^{n'}$. One can easily see that it can be found a chain $(i, i_1, i_2, \dots, i_{k-1}, k)$ s.t. $H_{i,i_1}=H_{i_1,i_2}=\dots, H_{i_{k-1},k}=1$ and $m_{i,i_1} m_{i_1,i_2} \dots m_{i_{k-1},k} > 0$. Thus M is irreducible, so all the hypothesis from *Theorem 2* are fulfilled and the EGA will converge.

Note 3. The hypothesis on M from this corollary is a generalization of the *classical* form (i.e. positive) of the mutation matrix: for $T=2^{n\ell}$ one will get the positivity of all the entrances in M .

Note 4. The condition C is *diagonal-positive* is not guaranteed by the assumptions in [9] - where only little attention is paid to the crossover operator. Nevertheless, this requirement is not very restrictive: for example a crossover probability $p_c < 1$ (as it is commonly used in practice) provides the diagonal-positivity of C . (This statement can be easily proved using the transition probabilities form derived in [1] for the *Three operator GA* (i.e. a GA with reproduction, mutation and crossover).)

As in [9, *Theorem 7*], the convergence result from *Theorem 2* can be adapted for the EGA copying the better solution directly after the evaluation phase (before selection). It follows straightforwardly:

Theorem 3. The EGA as in *Theorem 2*, but maintaining the best solution found over time *before* selection, converges to the global optimum, regardless of initialization.

Proof: Let $T=CM$. The transition matrix becomes:

$$P^+ = \begin{pmatrix} T & & & \\ & T & & \\ & & \ddots & \\ & & & T \end{pmatrix} U \begin{pmatrix} S & & & \\ & S & & \\ & & \ddots & \\ & & & S \end{pmatrix} =$$

$$= \begin{pmatrix} CMS & & & \\ TU_{21}S & TU_{22}S & & \\ \vdots & \vdots & \ddots & \\ TU_{2^\ell,1}S & TU_{2^\ell,2}S & \dots & TU_{2^\ell,2^\ell}S \end{pmatrix}$$

According to the conclusion of *Lemma 1*, CMS is irreducible and diagonal-positive.

Next, one can easily prove that an analogous of *Lemma 2* (but with the assumptions “CM irreducible and diagonal-positive, C, M and S diagonal-positive” - instead of “CMS is irreducible and diagonal-positive”) still holds for the new matrix model of the EGA. (The proof is quite similar to that of *Lemma 2*, so it will be omitted.) The conclusion is the same: a state from the EGA model is sub-optimal if and only if it is inessential, thus:

$$P^+ = \begin{pmatrix} P & 0 \\ R & Q \end{pmatrix} \quad \text{where } P=CMS \text{ is the square, irreducible matrix of optimal states,}$$

and Q - the square matrix of inessential states. From this point, the proof of *Theorem 2* can be reproduced identically, providing the globally optimal convergence of the algorithm.

4 Conclusions

The theoretical analysis of the binary GA relies strongly on the theory of Markov chains and non-negative matrices.

The previous papers in this field tackled the convergence behaviour of different models of GAs, yet the models proposed were not so close to a *real* GA. This study is an attempt of surpassing the gap between the theoretical properties and the *real* GA, stating that even relaxing one of the existing theoretical models (to a more realistic one) the convergence result still holds.

Against the well known hypothesis "*M (the mutation matrix) is positive*", herein is proposed a weaker assumption on the mutation operator, namely: "*M is irreducible with positive diagonal elements*". For instance, the irreducibility is satisfied by assuming that mutation can invert only a small number of bits in a generation - thus not all the bits, as required by the positive M condition. Additionally, the crossover matrix will be required to have positive diagonal elements.

Actually, this study proves that everything one needs for the elitist GA's convergence is the irreducibility of the transition matrix. So, one has to look for conditions on matrices C, M, S such that their product is irreducible. The approach given above may be seen either as:

- a different proof for the globally optimal convergence from [9], more direct, as we avoid the *Ergodic theorem* - notice that a primitive matrix is irreducible but not inversely, or as
- a generalization of the usual GA Markov chain model - regarding the non-(strictly)-positive form of the mutation matrix M,

The Markov chain modeling of the other elitist GA - the one *using* the best solution found over time to generate new individuals, not only *maintaining* it - is still in work, but one can anticipate that there will appear no major differences (both in modeling and convergence behavior) versus the above study.

A more important problem is the comparison between different types of convergence and, for the same convergence type, between different convergence rates. This objective requires the use of some constructive convergence results from the theory of probabilities and stochastic processes. The *Zero limit theorem for the sub-matrix of inessential states from a Markov chain* - hardly used in this paper - is such a powerful tool.

Acknowledgements

The author would like to thank G. Rudolph of Dortmund University for his support and for making the things clear in Markov chain modeling of GAs. The author is also grateful to the referees for their helpful comments and suggestions during the revision of the paper.

References

1. T.E. Davis and J.C. Principe, A Markov Chain Framework for the Simple Genetic Algorithm, *Evol. Comp. 1, Vol. 3* (1993) 269-288.
2. A.E. Eiben, E.H.L. Aarts and K.M. Van Hee, Global Convergence of Genetic Algorithms: a Markov Chain Analysis, in: *Parallel Problem Solving from Nature* (Springer, Berlin, 1990) 4-12.
3. D.E. Goldberg and P. Segrest, Finite Markov chain analysis of genetic algorithms, in: *Proc. ICGA '87 (Int. Conf. On Genetic Algorithms)* (Hillsdale, NJ: Lawrence Erlbaum, 1987) 1-8.
4. J.H. Holland, Adaptation in natural and artificial systems, *Ann. Arbor*, (The University of Michigan Press, 1975).
5. J. Horn, Finite Markov Chain Analysis of Genetic Algorithms with Niching, in: *Proc. ICGA '93 (Int. Conf. on Genetic Algorithms)* (1993) 110-117.
6. M. Iosifescu, *Finite Markov chains and applications* (Techn. Publ., Bucharest, 1977 - in Romanian).
7. M. Iosifescu, *Finite Markov Processes and Their Applications* (Chichester: Wiley, 1980).
8. A.E. Nix and M.D. Vose, Modeling genetic algorithms with Markov chains, *Ann. Math. and Artificial Intelligence* 5 (1992) 79-88.
9. G. Rudolph, Convergence Analysis of Canonical Genetic Algorithms, *IEEE Trans. on NN. 1, Vol. 5* (1994) 98-101.
10. X. Qi and F. Palmieri, Theoretical Analysis of Evolutionary Algorithms, parts I-II, *IEEE Trans. on NN. 1, Vol. 5* (1994) 102-129.
11. E. Seneta, *Non-negative Matrices and Markov Chains*, 2nd edition, (Springer, New York, 1981).
12. J. Suzuki, A Markov Chain Analysis on A Genetic Algorithm, in: *Proc. ICGA '93 (Int. Conf. on Genetic Algorithm, Urbana Illinois)* (Morgan Kaufmann, 1993) 46-153.
13. Y. Uesaka, Convergence of algorithm and the schema theorem in genetic algorithms, in: *Proc. ICANGA 1995* 210-213.
14. M.D. Vose, Modeling Simple Genetic Algorithms, in: D. Whitley, ed., *Foundations of Genetic Algorithms II* (San Mateo, CA: Morgan Kaufmann, 1993) 63-73.

An analysis of Punctuated Equilibria in Simple Genetic Algorithms

Sangyeop Oh and Hyunsoo Yoon

Dept. of CS & CAIR, KAIST, Daejeon, Korea
{syoh,hyoon}@camars.kaist.ac.kr

Abstract. In the running of a genetic algorithm, the population is liable to be confined in the local optimum, that is the metastable state, making an equilibrium. It is known that, after a long time, the equilibrium is punctuated suddenly and the population transits into the better neighbor optimum. We adopt the formalization of Computational Ecosystems to show that the dynamics of the Simple Genetic Algorithm is represented by a differential equation focusing on the population mean of a phenotype. Referring to the studies of differential equations of this form, we show that the duration time of metastability is exponential in the population size and other parameters, on the one dimensional bistable fitness landscape which has one metastable and one stable state.

1 Introduction

Genetic Algorithms (GAs) are optimization methods modeled from some operations which are used during the natural reproduction and the natural selection [3]. Since the concepts of GAs are introduced by Holland [?], various GAs show practical successes in various fields. Among them, the Simple Genetic Algorithm (SGA) is the simplest genetic algorithm containing the essential operators : selection, mutation, and crossover.

Like the other optimization methods, SGAs have the problem of metastability that the population is liable to be trapped in a local optimum, making an equilibrium. If there is a better optimum state in the vicinity, the local optimum is the metastable state since the punctuated equilibrium appears [11]. The punctuated equilibrium is the phenomenon in which the system in a metastable state shows a sudden transition into the more stable neighbor state after a long time. Punctuated equilibria are analyzed in various fields including Computational Ecosystems (CEs) and neo-Darwinian evolution models [1, 8].

The analysis of CEs is based on the time derivative of the probability distribution P , where $P(\mathbf{r}, t)$ is the probability that the system state is $\mathbf{r}$ at time t in the ensemble of possible system states. They use dP/dt to find $d\langle z \rangle/dt$ where $\langle z \rangle$ is the ensemble mean of the interested system character z [5].

In the neo-Darwinian model, the population mean $\bar{x}$ of a genetically determined character is governed by the equation $d\bar{x} = F'(\bar{x}) + \varepsilon dB$ where F is a landscape on x , $B(t)$ is a standard Brownian process, and ε is a small constant. They use the theoretical results from diffusion processes [7] to show that

the punctuated equilibria appear in the natural evolution with the exponential duration time of metastability as ε decreases to zero [8].

In this paper, we adopt the formalization of Computational Ecosystems to show that the dynamics of the Simple Genetic Algorithm is represented by a differential equation like $d\bar{x} = F'(\bar{x}) + \varepsilon dB$ focusing on the population mean of a phenotype. Referring to the studies of differential equations of this form, we show that the duration time of metastability is exponential in the population size and other parameters, on the one dimensional bistable fitness landscape which has one metastable and one stable state.

In section 2, the CE and the equation from the diffusion process are described. We analyze the dynamics of the SGA adopting the methods of CEs in section 3. The bistable fitness landscape is introduced and the simulation results are shown in section 4. In section 5 we discuss the results obtained in previous sections focusing on the duration time of metastability. Conclusion and further work are covered in section 6.

2 Background

2.1 Computational Ecosystems

The CE is a very similar model to GAs. A population contains N agents, and each agent chooses one from R resources to get some payment which is determined by the payment function f . f is the function of the chosen resource and the population state which is represented by a vector $\mathbf{r} = (r_1, r_2, \dots, r_R)$ whose i -th element is the ratio of agents choosing the resource i . During a unit time, each agent has the α chances to change the resource to the new one according to ρ_i , which is the probability that resource i is perceived to be the best choice and is the function of $\mathbf{r}$.

The possible population states at time t compose the ensemble represented by $P(\mathbf{r}, t)$ which is the probability of the population state to be $\mathbf{r}$ at t . Considering only one possible change of an agent's resource in a short time interval,

$$\frac{dP(\mathbf{n}, t)}{\alpha dt} = -P(\mathbf{n}, t) \sum_{i \neq j} n_j \rho_i + \sum_{i \neq j} P(\mathbf{n}^{[j, i]}, t) (n_j + 1) \rho_i^{[j, i]} \quad (1)$$

where $\mathbf{n}^{[j, i]}$ is such that $n_j^{[j, i]} = n_j + 1$, $n_i^{[j, i]} = n_i - 1$ and all other elements are the same as $\mathbf{n}$. And ρ_i and $\rho_i^{[j, i]}$ are evaluated at $\mathbf{n}$ and $\mathbf{n}^{[j, i]}$, respectively.

The ensemble mean $\langle r_i \rangle$ of r_i satisfies

$$\frac{d\langle r_i \rangle}{dt} = \alpha (\langle \rho_i \rangle - \langle r_i \rangle). \quad (2)$$

using equation (1) [5].

2.2 Diffusion processes

Consider the one dimensional diffusion process $z(t)$ satisfying

$$dz(t) = F'(z) + \varepsilon dB(t) \quad (3)$$

where $B(t)$ is the standard Brownian process and ε is a small constant. Let F satisfy the following conditions :

- F is a differentiable function defined on $-\infty < z < \infty$,
- there exists $z_1 < z_2 < z_3$ such that F is strictly increasing on $(-\infty, z_1] \cup [z_2, z_3]$ and strictly decreasing on $[z_1, z_2] \cup [z_3, \infty)$, and
- $F(z_1) < F(z_3)$.

If $z(0) = z_1$, then the punctuated equilibrium appears and the duration time T of metastability satisfies

$$T \propto \exp\left(\frac{2(F(z_1) - F(z_2))}{\varepsilon^2}\right) \quad (4)$$

when $\varepsilon \ll 1$. And the transition is unidirectional in the sense that the system remains in the stable state permanently [7]. These results about the duration time can be expanded to the cases that F has more than two peaks or z is multidimensional [7, 2].

3 Analysis

The analysis of genetic algorithms in the genotype is somewhat complex since it considers vector values and matrix operations. We focus on the phenotype x_i of an individual where x_i corresponds to the genotype i .

3.1 Simple Genetic Algorithms

A SGA in this paper deals with a population which consists of N individuals. Each individual is a binary string of L bits, each bit has one of two values 0 and 1, and there are $R = 2^L$ genotypes. The phenotype for the genotype i is x_i and the fitness f is the function on the phenotype domain. The population state is represented by $\mathbf{n} = (n_1, n_2, \dots, n_R)$ or $\mathbf{r} = (r_1, r_2, \dots, r_R)$ where $n_i(t)$ is the number of individuals with the genotype i and $\mathbf{r} = \mathbf{n}/N$.

The population of the next generation is produced from the current one through some SGA operators : *roulette wheel selection*, *1-point crossover* and *simple mutation* [3]. After two individuals are selected from the current population by the roulette wheel selections, the 1-point crossover and the simple mutation are applied to the pair. In the 1-point crossover, each individual is cut at the same point and divided into two substrings, and then the second substrings are exchanged. The cutting point is determined randomly among all the points between two bits. And then, the simple mutation toggles each bit of

individuals with the probability p_m . After the mutations, one of two children is chosen randomly and added into the next generation.

Repeating this process N times, the new generation with the generation gap 1 can be completely produced. For the simplicity, the crossover is not considered in the analysis and is mentioned in section 5.2. The parameters and the functions related to selection, crossover and mutation are superscripted with ^{s} , ^{c} and ^{m} respectively.

3.2 Brownian part

As it is defined in the CE, let $P(\mathbf{r}, t)$ be the probability that the population state is $\mathbf{r}$ at time t . Then the ensemble mean and the population mean of z are represented, respectively as $\langle z \rangle = \sum_{\mathbf{r}} z P(\mathbf{r}, t)$ and $\bar{z} = \sum_i z r_i$.

Let α be the average number of generations per unit time. Then the SGA changes the population state $\alpha \Delta t$ times during Δt . When an individual is chosen after selection, crossover and mutation, the phenotype of the individual can be considered as a random variable X . Using the central limit theorem [10], the population mean $\bar{X}$ of the phenotypes follows

$$\bar{x}(t + \Delta t) - \bar{x}(t) = \langle \bar{x}(t + \Delta t) \rangle - \bar{x}(t) + \alpha \Delta t G \quad (5)$$

where G is a Gaussian random variable with mean 0 and variance $V(\bar{X})$.

Since the dispersion of the ensemble starts from the instantiated state with $\bar{x}(t)$, $\langle \bar{x}(t) \rangle = \bar{x}(t)$. The accumulation of Gaussian random variables, each of which has the variance 1, makes the standard Brownian process. Then, since $V(\bar{X}) = V(X)/N$ by the central limit theorem,

$$d\bar{x}(t) = d \langle \bar{x}(t) \rangle + \alpha \sqrt{\frac{V(X)}{N}} dB(t). \quad (6)$$

Ignoring the crossover, the random variable X of phenotype is composed of X^s and X^m ,

$$X = X^s + X^m \quad (7)$$

where X^s is the result of the selection and X^m is the change due to the mutation. The effects of the mutation is dependent on how the genotype is decoded into the phenotype, and then into the fitness. To obtain a general feature of the simple mutation, we define the phenotype as $x_i = l_i/L$, where l_i is the number of bits with the value 1 in the genotype i . In case of $L = 1$, X^m has the probability distribution with mean $(1 - 2l(t)/L)p_m$ and variance $p_m - (1 - 2l(t)/L)^2 p_m^2$. Thus, in general, X^m has the Gaussian distribution with

$$\begin{aligned} E(X^m) &= (1 - 2l(t)/L)p_m \quad \text{and} \\ V(X^m) &= [p_m - (1 - 2l(t)/L)^2 p_m^2]/L \simeq p_m/L \end{aligned} \quad (8)$$

by the central limit theorem.

Considering only the selection, the variance of the phenotype seems to be dependent on $f^{(1)}/\bar{f}$, where $f^{(m)}$ is the m -th derivative of $f(x)$. Even when the

fitness landscape is flat, the selection makes the variance decrease by the factor of $1/N$ [9], and then

$$0 < V_{steepest} \leq V(X) \leq V_{flat} = \frac{Np_m}{L} \quad (9)$$

where $V_{steepest}$ and V_{flat} are variances when $|f^{(1)}/\bar{f}|$ is maximum and zero, respectively. That is, $V(X)$ is finite within the range of equation (9).

3.3 Ensemble mean part

Consider the case of the generation gap $1/N$, in which a child is produced from the SGA operators, a victim is chosen randomly from the current generation and then it is replaced by the child, producing the next generation. When genotypes of the child and the victim are i and j respectively, this random event corresponds with the resource changing from j to i for an agent in CEs. That is, the SGA with the generation gap $1/N$ is the special case of the CE, when ρ_i is interpreted as the probability that the child with the genotype i would be produced by the SGA operations.

However, if the generation gap is 1, equation (1) can not be used by the SGA since it is the result of considering maximally one change of an agent in a given time interval. Thus we focus on the macroscopic equation (2) whose right hand side is interpreted as $N\alpha$ generations per unit time multiplied by the effect of one generation, $(\langle \rho_i \rangle - \langle r_i \rangle)/N$ on condition that the generation gap is $1/N$. If one generation contains N changes of individuals and they are all relatively independent, the effect of one generation is enlarged to be $(\langle \rho_i \rangle - \langle r_i \rangle)$. Strictly speaking, the changes of individuals in one generation of the SGA is not independent since the victim is selected in round robin. But it makes the same effect to the independent case since the round robin guarantees that each individual has the same probability to be selected as a victim. Modifying the definition of α as the average number of generations per unit time, equation (2) can also be used for the SGA with the generation gap 1.

From equation (2), we can obtain

$$\frac{d\langle \bar{x}(t) \rangle}{dt} = \alpha \sum_{i=1}^R x_i (\rho_i(t) - r_i(t)). \quad (10)$$

Considering only the selection,

$$\sum_{i=1}^R x_i \left(\frac{f(x_i)r_i}{\bar{f}} - r_i \right) = \frac{\overline{(x - \bar{x})f}}{\bar{f}} = \frac{1}{\bar{f}} \sum_{m=0}^{\infty} \frac{f^{(m)}(\bar{x})}{m!} \overline{(x - \bar{x})^{m+1}}. \quad (11)$$

On condition that

$$f(x) \simeq f(\bar{x}) + (x - \bar{x})f^{(1)}(\bar{x}) \quad (12)$$

for each individual in the population, equation (11) becomes

$$\sum_{i=1}^R x_i \left(\frac{f(x_i)r_i}{\bar{f}} - r_i \right) = s_X^2 \frac{\partial \log f(\bar{x})}{\partial \bar{x}} \quad (13)$$

where $s_X^2 = \overline{(x - \bar{x})^2}$, the population variance.

The change due to the mutation is, from equation (8),

$$\left[\frac{d < \bar{x}(t) >}{\alpha dt} \right]^m = p_m(1 - 2\bar{x}^s) \simeq p_m(1 - 2\bar{x}). \quad (14)$$

3.4 Dynamics due to selection and mutation

Let V^s be the some constant representing the selection term of $V(X)$ within the range of equation (9). If we assume that s_X^2 is independent of $\bar{x}$ for the simplicity then the global fitness function F can be defined by

$$F(\bar{x}) = s_X^2 \log f(\bar{x}) + p_m(\bar{x} - \bar{x}^2). \quad (15)$$

This assumption will be mentioned again in section 5.3. Finally, from equations (6), (8), (13) and (14), we can summarize the dynamics of the SGA as

$$\frac{d\bar{x}(t)}{\alpha dt} = \frac{\partial F(\bar{x})}{\partial \bar{x}} + \sqrt{\frac{V^s + p_m/L}{N}} \frac{dB(t)}{dt} \quad (16)$$

considering selections and mutations except the crossover.

4 Simulation

4.1 Bistable landscape

Punctuated equilibria appear if F has the landscape as is shown in section 2.2. In this simulation, we restrict f to have three features. First, F satisfies the conditions in section 2.2 except that F is not differentiable at finite number of points. Secondly, the selection pressures from the barrier are the same for the metastable and stable attractors. And lastly, the condition of equation (12) is satisfied for each individual. Strictly speaking, the third is not always true since there are some generations in which some individuals are in the right side of the barrier and the others remain in the left side. But this dispersion is condensed into a population in one side very quickly and hence we ignore the effect. The phenotype x is defined by $x_i = l_i/L$ as in the case of mutation analysis.

The typical f and F are shown in figure 1, where $d = f(0) - f(1/3)$ is the barrier depth in f landscape. Other fitness function like the generalized deceptive functions [12] could also be used if it has the deceptiveness and the multistep trajectory from the local optimum to the global optimum state, but are not covered in this paper.

Figure 2 shows typical punctuated equilibria appeared in the running of the SGA. The population starts from the state $\bar{x} = 0$, converges quickly into the metastable state, and shows perturbations around it. After a long duration, $\bar{x}$ transits the fitness barrier to the stable state suddenly.

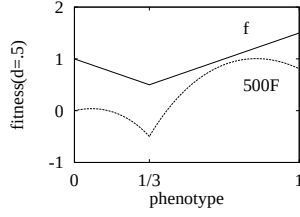


Fig. 1. A fitness function with barrier depth d

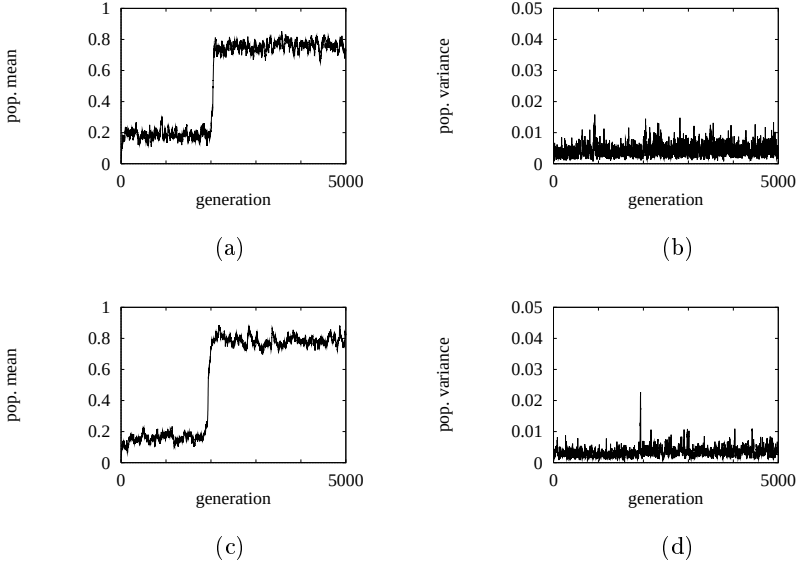


Fig. 2. The punctuated equilibria are shown on the graph of $\bar{x}$ as generations are succeeded. (a) and (b) are $\bar{x}$ and s_X^2 respectively, where $N = 20$, $p_c = 1.0$, $p_m = 0.012$ and $d = 0.5$. (c) and (d) are $\bar{x}$ and s_X^2 respectively, where $N = 30$, $p_c = 0.0$, $p_m = 0.008$ and $d = 0.5$.

4.2 The time of metastability

Beginning with the population state with $\bar{x} = 0$, we record the duration T of metastability varying some parameters, where T is defined as the number of generations till the transition to the stable state occurs. The time required for the transition is so short relative to T that it is ignored. Considered parameters are the population size N , the barrier depth d in f landscape, and the mutation probability p_m .

Figure 3 is the simulation results when $p_c = 0$. Figure 3(a) and 3(b) shows that T is a rapidly increasing function of N and d . And figure 3(c) shows that

T is a rapidly decreasing function of p_m .

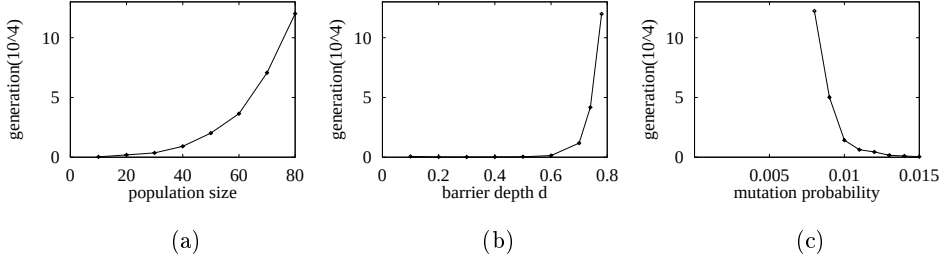


Fig. 3. The duration T of metastability versus some parameters. Each point is an average of 30 runs. Default parameter values are $L = 30$, $N = 40$, $p_c = 0$, $p_m = 0.01$ and $d = 0.7$.

In case of $p_c = 1$, figure 4 is the counterpart of figure 3. This shows that the crossover makes the duration be longer for any parameter set in this simulation environment.

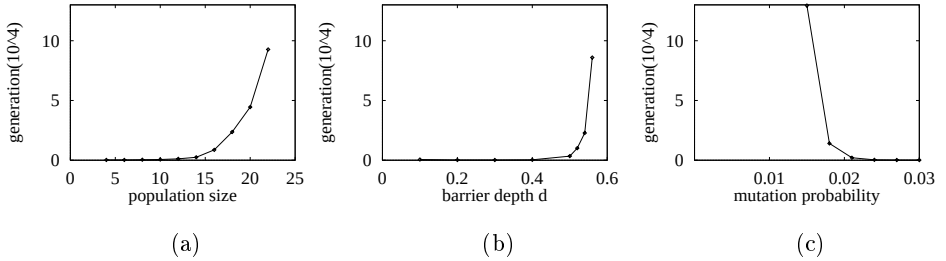


Fig. 4. The duration T of metastability versus some parameters. Each point is an average of 30 runs. Default parameter values are $L = 30$, $N = 40$, $p_c = 1$, $p_m = 0.01$ and $d = 0.7$.

5 Discussion

5.1 Effects of selection and mutation

Since equation (16) has the form of equation (3), the duration T of metastability satisfies

$$T \propto e^{\frac{2DN}{V^s + p_m/L}} \quad (17)$$

by equation (4), where D is the barrier depth in F landscape.

The mean population phenotype $\bar{x}(t)$ oscillates around the equilibrium value x_e which is determined from the non-Brownian part of equation (16), $\partial F(\bar{x})/\partial \bar{x} = 0$. Given p_m and d , the relation between s_X^2 and x_e can be obtained from this equation and is approximately consistent with the graphs of figure 2.

Among the SGA runs in the simulation, there is not any case that the population returns to the metastable state once the punctuated equilibrium occurs, provided that $p_m \ll 1$. This means that the transition is unidirectional.

Figure 3(a) qualitatively confirms that T is exponential in the population size N . But figure 3(b) needs an explanation since the barrier depth d is the quantity of f landscape. Let x_e be the root of $\partial F(\bar{x})/\partial \bar{x} = 0$ in the metastable area. Then, from equation (17),

$$T \propto e^{K[\log f(x_e) - \log(1-d)]} = \left[\frac{f(x_e)}{1-d} \right]^K \quad (18)$$

where K is a constant. That is, the shape of the graph of T is that of $1/(1-d)$ rather than that of e^d . Figure 3(c) shows that why the condition of $p_m \ll 1$ is needed for the punctuated equilibria to be appeared, and is consistent to equation (17).

5.2 Effects of crossovers

In a bistable problem, individuals can be divided into two types according to which basin of attractor they belong to : A -type and B -type, respectively in metastable and stable area. Let r_B and f_B be the ratio and the average fitness of B -type individuals, respectively.

For most generations, $r_B = 0$, and r_B becomes positive at long intervals by crossovers or mutations. When this event occurred at τ -th generation,

$$r_B(\tau + 1) \simeq [r_B^s(\tau)]^2 + \beta r_A^s(\tau) r_B^s(\tau) \quad (19)$$

since the term of $[r_A^s(\tau)]^2$ and the effect of mutation can be ignored. The parameter β is the rate that B -type children are produced from the crossover of one A -type and one B -type parents.

When the phenotype is defined as $x = l/L$, the crossover has the tendency that the x 's of two parents are averaged. If one A -type and one B -type parents crossover then the children would be around the barrier terminating the appearance of the B -type, and hence $\beta \ll 1$. That is, the crossover not only enhances the appearance of B -type individual, but also would eliminate it. Figure 5 shows that the crossover interrupts the appearance of the B -type as a whole if the genotype is decoded into the fitness as is done in section 4. On condition that β is larger than a particular criterion, this interrupt would be replaced by the enhancement. But the crossover is highly dependent on the definitions of x and f , and the generalization is not considered in this paper.

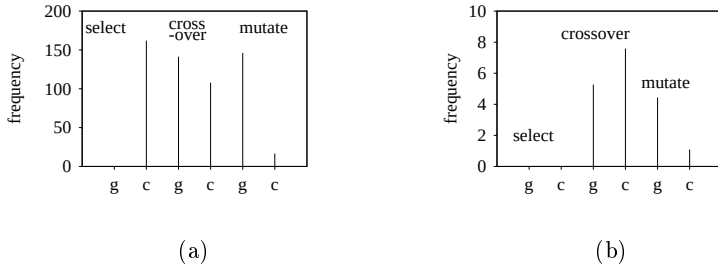


Fig. 5. The individual with the maximal phenotype in the population is traced. It sometimes goes and comes across the particular phenotype criterion value after each SGA operation. The criterion is $1/3$ for (a), and $14/30$ for (b). For the parameters in this simulation, the individuals in $1/3 \leq x \leq 14/30$ disappear by the selections. Hence we focus on (b) to elucidate the contributions of GA operations to the transition into the stable state. The number of goings or comings in a run is counted until the transition occurs, and then averaged over 100 runs. g and c represent ‘going’ and ‘coming’, respectively. Parameter values are $L = 30$, $N = 20$, $p_m = 0.012$ and $d = 0.5$.

5.3 Variance of phenotype

Since the population variance s_X^2 is the sample variance corresponding to the ensemble variance $V(X)$, the sampling of s_X^2 has the distribution with mean $E(s_X^2) = V(X)$ and variance $V(s_X^2) = 2(V(X))^2/(N - 1)$, in which $V(s_X^2)$ represents the sampling perturbation [10].

The selection makes $V(X)$ decrease according to the relative gradient of f , $(\partial f / \partial x) / \bar{f}$. As the relative gradient of f decreases, the selection force decreases and $V(X)$ increases. However the increment is small relative to the change of $\log f$ as shown in figure 6 and this supports the assumption for equation (15).

As the population size N increases, the diversity of genotypes within the population increases but it does not directly mean that the phenotype variance increases. Instead, figure 7 shows two things that, as N increases, the increment of $V(X)$ is negligibly small compared with the linear increment, and the amplitude of sampling perturbations of s_X^2 decreases as predicted above. The first supports the assumption that V^s is approximately independent of N . The second means that, as N increases, the effect of increasing $V(X)$ could be dominated by that of decreasing $V(s_X^2)$ in the small range of N .

6 Conclusion and future work

In this paper, we have analyzed the dynamics of the SGA to get a differential equation about the population mean of phenotype values. It is divided into the Brownian part and non-Brownian part by the central limit theorem, as shown in equation (6). Detailing the equation, we have considered selections and mutations except the crossover for the simplicity, and the result is equation (16).

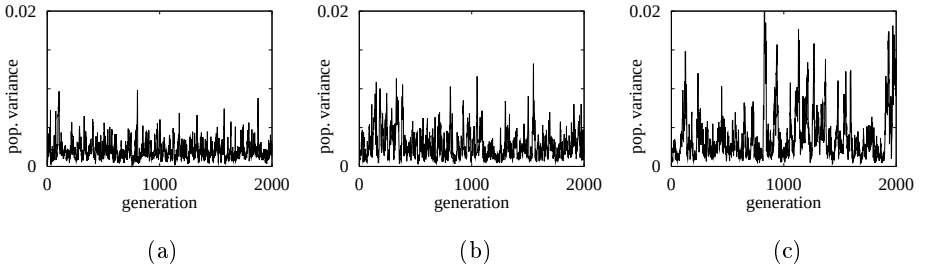


Fig. 6. The population variance s_X^2 of phenotype at each generation. Parameter values are $N = 20$, $p_c = 0$ and $p_m = 0.01$ except that $d = 0$ and $d = 0.5$ for (a) and (b) respectively. The fitness function of (c) is the same as that of (b) except that the gradient is increased to 1500 on $x > 1/3$. The population goes into the stable area after several generations in these cases, where the gradients are 0, 1.5 and 1500 for (a), (b) and (c), respectively.

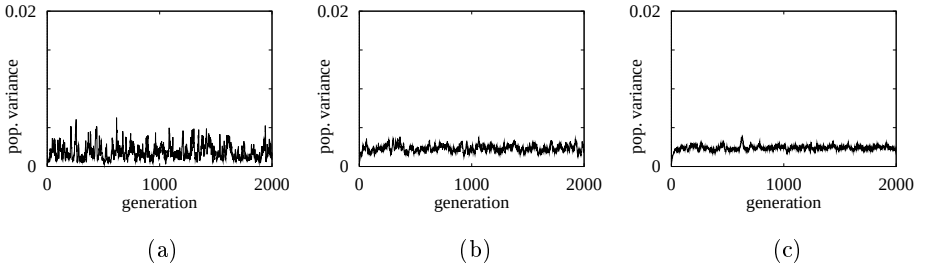


Fig. 7. The population variance s_X^2 of phenotype at each generation. Parameter values are $p_c = 0$, $p_m = 0.005$ and $d = 0.5$, except that $N = 40$, $N = 400$, and $N = 1600$ for (a), (b) and (c), respectively.

The non-Brownian part is the dynamics of the ensemble mean of phenotype values and has been analyzed adopting the method of the CE. The effect of roulette wheel selections is proportional to the population variance and the logarithm of the fitness function in condition of equation (12). The effects of mutations and crossovers are dependent on how the genotype is decoded into the phenotype, and then into the fitness. When the phenotype is defined as $x = l/L$, the effect of simple mutations is proportional to the mutation probability and $(1/2 - \bar{x})$.

For the equation (16) has the typical form which has been analyzed in diffusion processes, we can adopt the theoretical results from them. That is, running the SGA on the bistable global fitness landscape F which is defined as equation (15), (i) the punctuated equilibrium appears, (ii) the duration time of metastability is exponential as shown in equation (17), and (iii) the transition is unidirectional. These theoretical results are qualitatively confirmed by the simulation

results.

These results about the duration time could be expanded to the cases that F has more than two peaks or x is multidimensional [7, 2].

Though this paper shows some interesting results, it has some defects to be supplemented. The major one is that the effect of crossovers is just roughly analyzed. When the phenotype is defined as $x = l/L$, 1-point crossovers make the duration of metastability be longer. The next one is that, we have regarded the variance of the phenotype $V(X)$ to be constant, on a basis of simulation results. For the more accurate analysis, the relation between $V(X)$ and GA parameters such as the population size and the gradient of the fitness landscape should be examined.

Since GAs have direct relations with CEs, the results obtained from the studies of CEs could be applied to GAs. These include the issues about time delay, cooperation, competition, chaos, and so on [6, 4]

References

1. H. A. Ceccatto and B. A. Huberman. Persistence of nonoptimal strategies. *Proceedings of the National Academy of Sciences of the United States of America*, 86:3443–3446, 1989.
2. A. Galves, E. Olivieri, and M. E. Vares. Metastability for a class of dynamical systems subject to small random perturbations. *The Annals of Probability*, 15(4):1288–1305, 1987.
3. D. E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, New York, 1989.
4. Tad Hogg and Bernardo A. Huberman. Controlling chaos in distributed systems. *IEEE transactions on Systems, Man, and Cybernetics*, 21(6):1325–1332, 1991.
5. B. A. Huberman and T. Hogg. The behavior of computational ecologies. In B. A. Huberman, editor, *The Ecology of Computation*. North-Holland, Amsterdam, 1988.
6. J. O. Kephart, T. Hogg, and B. A. Huberman. Dynamics of computational ecosystems. *Physical Review A*, 40(1):404–421, 1989.
7. C. Kipnis and C. M. Newman. The metastable behavior of infrequently observed, weakly random, one-dimensional diffusion processes. *SIAM Journal on Applied Mathematics*, 45(6):972–982, 1985.
8. C. M. Newman, J. E. Cohen, and C. Kipnis. Neo-darwinian evolution implies punctuated equilibria. *Nature*, 315:400–401, May 30 1985.
9. A. Prügel-Bennett and Jonathan L. Shapiro. An analysis of genetic algorithms using statistical mechanics. *Physical Review Letters*, 72:1305–1309, 1994.
10. R. L. Scheaffer and J. T. McClave. *Probability and statistics for engineers*. PWS-KENT, Boston, 1990.
11. M. D. Vose. Punctuated equilibria in genetic search. *Complex Systems*, 5:31–44, 1991.
12. L. Darrell Whitley. Fundamental principles of deception in genetic search. In Gregory J. E. Rawlins, editor, *FOGA*. Morgan Kaufmann, San Mateo, California, 1991.

SGA Search Dynamics on Second Order Functions

Bart Naudts* and Alain Verschoren

Dept. of Mathematics and Computer Science
University of Antwerp, RUCA
Groenenborgerlaan 171, B-2020 Antwerpen, Belgium
e-mail: [bnaudts,aver]@ruca.ua.ac.be

Abstract. By comparing its search dynamics to that of a simple $\mathcal{O}(n^2)$ heuristic, we are able to analyze the behavior of the simple genetic algorithm on second order functions, whose optimization is shown to be an NP-equivalent problem. It appears that both algorithms approach these fitness functions in the same, intuitive way: they start by deciding the obvious and most probable, and then they proceed to make more difficult decisions. Useful information about the optimization problem is, among others, provided by statistical physics: lattice gases can be modeled as second order functions.

1 Introduction

In this note, we analyze the behavior of the simple genetic algorithm (SGA) on instances of a particular NP-equivalent optimization problem. (See [10] for a general introduction to complexity theory.) By choosing such a problem, we are ensured of instances of real world difficulty level, while still having a single and easily definable fitness function class to focus on.

A consequence of this choice is that we have to face the structural difficulties typical to NP-completeness. Firstly, the optimum of a function is something unreachable: exhaustive search is too expensive, and no efficient algorithm can tell a local optimum from a global one. A second problem is that it is very difficult to give a general characterization of difficult instances. E.g., recently a special volume [2] was dedicated to locating difficult instances of the 3SAT problem. Finally, one often has no idea of the characteristics of the fitness landscape, while still knowing that it contains many local optima which are fairly close to the global optimum. Only one path could lead to the global optimum, and there is no information to tell a good path from a bad one.

Throughout this note, we consider fitness functions of the form $f : \mathcal{S} \rightarrow \mathbb{Z}$, where $\mathcal{S}$ is the set of bit strings indexed by $\Lambda = \{0, 1, \dots, \ell-1\}$, i.e., $\mathcal{S} = \{0, 1\}^\Lambda$. Our very basic function class is the class of *first order functions*

$$f(s) = \sum_{i \in \Lambda} g_i(s_i), \quad g_i : \{0, 1\} \rightarrow \mathbb{Z}, \quad s \in \mathcal{S}, \quad (1)$$

* Research assistant of the Fund for Scientific Research – Flanders (F.W.O.) (Belgium)

also called *linear* or *fully separable* functions. Typical is that each locus contributes to the function value independently of the other loci, which makes these simple functions almost perfect for SGA optimization because the uniform crossover operator literally “exchanges properties”.

First order functions occur all over the place. The epistasis measure (e.g., [1],[6]) computes the least squares distance between a fitness function and the class of first order functions, whereas the bit decidability metric [6] is based on the concept of independent decidability of loci. Integer Programming (e.g., [11]) requires a first order function to be optimized, under the restriction of a number of linear relations. In Sect. 2.2, we see that the well known problem from statistical physics of finding the ground state of a lattice gas can be modeled into a fitness function maximization problem. There, the loci represent the sites of the lattice: each site can be occupied by a gas molecule or can be empty, or, alternatively, can be associated with a magnetic spin with either north (e.g., bit value 0) or south pole (e.g., bit value 1). In the latter case, the first order function represents an external magnetic field.

One adds second order components $g_{ij} : \{0,1\}^2 \rightarrow \mathbb{Z}$ to Eq. 1 to obtain the *second order functions*

$$f(s) = \sum_{i \in \Lambda} g_i(s_i) + \sum_{i < j \in \Lambda} g_{ij}(s_i, s_j). \quad (2)$$

The optimization problem associated with the second order functions will be denoted as 2ORD. It is a restriction of this class (hence a subproblem of 2ORD) which will be the subject of our analysis.

As we will see in the next section, a fair number of real world optimization problems can be embedded into 2ORD. This connection with extensively studied problems is important as a source of algorithms to compare with, a source of theoretic results which provides insight in the structure of the class, and, if nothing else, as a source of difficult instances. Two additional motivations for the choice of 2ORD are that it is probably the simplest yet very rich function class beyond the first order functions, and that there is no encoding from an abstract problem space to the space of bit strings involved. Also, note that Kauffman’s *NK*-landscapes [4] with $K = 1$ and $N = \ell$ can essentially be embedded in the substantially richer class 2ORD.

It has been proved in [7] that 2ORD is an NP-equivalent problem. In view of this complexity, one may wonder whether any metric based on first order functions can grasp the richness of the second order functions and render it using one single value. Simple as it is, though, the bit decidability measure provides valuable (but by no means exhaustive!) information on the GA-hardness of the instances.

The principal aim of this note is to show that the search dynamics of the SGA can be predicted for many functions in U-2ORD-01, an NP-equivalent subclass of 2ORD. This is done by comparing the dynamics of the SGA to those of a very intuitive $\mathcal{O}(\ell^2)$ heuristic, which tries to optimize U-2ORD-01 functions by deciding one locus at a time. One observes that the SGA and the heuristic

follow the same strategy, in particular, one finds a very high correlation between the order in which both algorithms decide the values of the loci. Moreover, the dynamics of the heuristic is a very good indicator of GA-hardness, the essential reason being the fact that it faces the same structural difficulty as the SGA does.

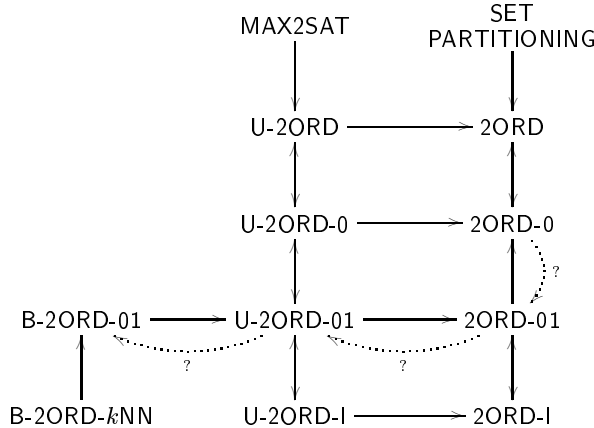


Fig. 1. Overview of inclusions and reductions. An arrow from one problem to another indicates a reduction or inclusion, e.g., U-2ORD can be reduced into U-2ORD-0, while U-2ORD-0 is trivially included in U-2ORD.

2 U-2ORD-01 and Relatives

Before sketching the big picture, let us first motivate the naming scheme for the function classes and associated optimization problems introduced in this note. The U- prefix stands for “unsigned”, e.g., U-2ORD is the subset of 2ORD instances which only have non-negative second order components $g_{ij}(\cdot, \cdot)$. The suffix -01 indicates that second order terms will only be included in the sum when $s_i = 0$ and $s_j = 1$; similarly the -0 suffix excludes all $g_{ij}(1, \cdot)$ terms. The -I suffix stands for “interaction”: terms are only included when $s_i \neq s_j$. More formally,

$$\text{2ORD-0:} \quad f(s) = \sum_{i \in A} g_i(s_i) + \sum_{i < j \in A} g_{ij}(s_j) \cdot (1 - s_i), \quad (3)$$

$$\text{2ORD-01:} \quad f(s) = \sum_{i \in A} g_i(s_i) + \sum_{i < j \in A} g_{ij} \cdot (1 - s_i)s_j, \quad (4)$$

$$\text{2ORD-I:} \quad f(s) = \sum_{i \in A} g_i(s_i) + \sum_{i < j \in A} g_{ij} \cdot (s_i + s_j - 2s_i s_j). \quad (5)$$

Here, the g_i and g_{ij} take integer values. Note that none of the annotations affect the first order terms.

Via MAX2SAT it is proved in [7] that optimizing U-2ORD-01 functions is an NP-equivalent problem. Figure 1 shows the necessary reductions, as well as some other problems and reductions: from SET PARTITIONING into 2ORD, from U-2ORD-l into U-2ORD-01, etc. Even two processor network balancing can be modeled into U-2ORD, although the arbitrary processor version is much more interesting. See [8] for more details.

The following two subsections complete the overview of optimization problems and function classes related to 2ORD.

2.1 Subclasses of U-2ORD-01

The B- (for *binary*) prefix indicates that the second order components g_{ij} take values in $\{0, 1\}$, while the first order components keep their larger range. The fact that a number of very difficult U-2ORD-l instances can be written as B-2ORD-01 instances (as in Sect. 4), leads us to conjecture that B-2ORD-01 is NP-equivalent.

The B-2ORD-2NN class consists of those functions of B-2ORD-01 where $j-i > 2$ implies that $g_{ij} = 0$. Hence its name: in the worst case a locus is dependent on its two *nearest neighbors*. Instances of B-2ORD-2NN can be optimized polynomially by a simple *divide and conquer* algorithm. In general, if one restricts the span of the edges to a constant distance $k \geq 1$, then the resulting optimization problem U-2ORD- k NN remains polynomial.

2.2 Interaction models

The most interesting instances of interacting particle systems [5] can be modeled in the class U-2ORD-l, where a component g_{ij} contributes to the function value when s_i and s_j have opposite values. Finding the ground state of an anti-ferromagnetic model then accords to maximizing a particular fitness function.

Example. Let us consider the anti-ferromagnetic one dimensional Ising model (see [12], e.g.) with cyclic boundary conditions. The model is represented by an instance of U-2ORD-l; for $\ell = 6$ we get

$$f(s) = (s_0 + s_1 - 2s_0s_1) + (s_1 + s_2 - 2s_1s_2) + (s_2 + s_3 - 2s_2s_3) + (s_3 + s_4 - 2s_3s_4) + (s_4 + s_5 - 2s_4s_5) + (s_5 + s_0 - 2s_5s_0). \quad (6)$$

This fitness function can be visualized by

$$*_0 \longleftrightarrow *_1 \longleftrightarrow *_2 \longleftrightarrow *_3 \longleftrightarrow *_4 \longleftrightarrow *_5 \quad (7)$$

Each vertex in this picture represents a locus; the label of the vertex indicates the locus index. A bidirectional edge between vertices i and j indicates that $g_{ij} = 1$; components not represented by an edge in the picture have a value of zero.

The asterisk ‘*’ in each label is a place-holder for a bit value. While the picture in Eq. 7 represents f , the following picture represents f together with the string 010101, which happens to be one of the two global optima:

$$0_0 \xleftrightarrow{\quad} 1_1 \xleftrightarrow{\quad} 0_2 \xleftrightarrow{\quad} 1_3 \xleftrightarrow{\quad} 0_4 \xleftrightarrow{\quad} 1_5 \quad (8)$$

By changing their first order component, we can reduce U-2ORD-I instances into U-2ORD-01 instances, which brings them within the domain of the heuristic. A detailed proof of this fact may be found in [7]; the following example illustrates the main technique:

Example. Let $A = \{0, 1\}$ and consider the simple U-2ORD-I instance

$$f(s) = a(s_0 + s_1 - 2s_0s_1), \quad (9)$$

with $a \in \mathbb{Z}$. We may rewrite this as

$$f(s) + a = as_0 + a(1 - s_1) + 2a(1 - s_0)s_1, \quad (10)$$

to recognize in the right hand side an instance of U-2ORD-01. As the constant term a in the left hand side can be ignored in the context of function optimization, we have performed a correct reduction; it is visualized by

$$*_0 \xleftrightarrow{v} *_1 \quad \Longrightarrow \quad \begin{array}{c} \nearrow v \\ *_0 \xrightarrow{2v} *_1 \nearrow v \end{array} \quad (11)$$

The picture also shows three more features of our system of visualizing second order functions. Firstly, when visualizing 2ORD-01 instances, we use unidirectional edges. Secondly, if an edge is labeled, it represents a second order component g_{ij} with the value of that label, instead of the value 1. Lastly, the first order components are visualized by edges which do not connect but point to resp. from a vertex, indicating non-zero $g_i(1)$ resp. $g_i(0)$ values.

Another interesting example is the two dimensional anti-ferromagnetic Ising model on a square lattice. Figure 2 describes it as an instance of U-2ORD-I and of U-2ORD-01.

The well established field of statistical physics gives a number of indications why finding ground states can be difficult. We conclude this section by putting three of them, *dimensionality*, *symmetry* and *frustration*, in the context of second order function optimization.

Dimensionality refers to the degree of connectedness of the edges of a function visualization. We have already come across an optimization problem in which one dimensional interaction models can be embedded: U-2ORD- k NN. Here, the span of the edges is restricted to the constant k . In statistical physics, one dimensional models can be dealt with analytically; we see that the corresponding

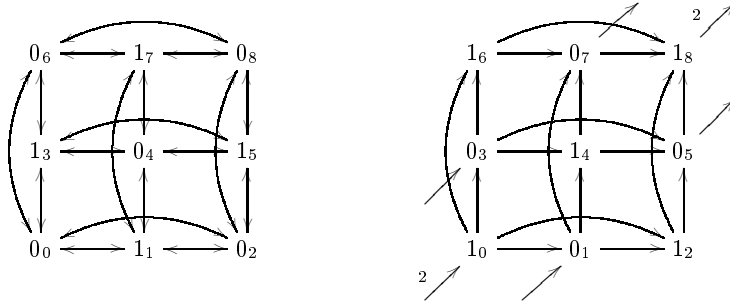


Fig. 2. The $d=2$ anti-ferromagnetic Ising model on a square lattice, with 3^2 sites, as an instance of U-2ORD-I and U-2ORD-01. This model has two ground states, i.e., two global optima: the two checkerboard patterns shown together with the model.

optimization problem is polynomial. It is visually clear that the model of Eq. 6 is one dimensional and that that of Fig. 2 is two dimensional. Interaction models become easier with the dimensionality increasing, the two dimensional models being the hardest. We will observe this behavior in Sect. 5, when we consider U-2ORD- k NN instances for k near ℓ .

It is easy to see that the Ising models are highly symmetrical, which is a potential source of hardness, mainly because sequential optimization algorithms do not have a clue of where to start the optimization process. The effects of symmetry are most easily seen in conjunction with the third hardness indicator, frustration.

Frustration occurs, e.g., in the model visualized in Fig. 3 below. The figure shows a two dimensional anti-ferromagnetic Ising model, this time not on a square but on a triangular lattice. It is beyond the scope of this note to formally define the notion of frustration, but we can easily illustrate it as follows.

The two ground states of the Ising model on a square lattice are checkerboard patterns, i.e., for each square in the model, one finds optimal solutions

$$\begin{array}{cc}
 0_2 \longleftrightarrow 1_3 & 1_2 \longleftrightarrow 0_3 \\
 \updownarrow & \updownarrow \\
 1_0 \longleftrightarrow 0_1 & 0_0 \longleftrightarrow 1_1
 \end{array} \tag{12}$$

A sequential algorithm would start optimizing, e.g., in the lower left corner, by choosing a value of 0 for locus 0. Then it would decide to put 1's at loci 1 and 2, to make sure that it gets the components g_{01} and g_{02} in the fitness sum. The decisions made by this sequential algorithm turn out to be successful, since by putting a 0 at locus 3, the algorithm has found the global optimum.

The symmetry of the square lattice allows the algorithm to start at any point, and reach one of the two global optima by proceeding in a greedy or hill climbing sort of way. We say that all paths lead to the global optimum.

This equilibrium can easily be disturbed by adding a diagonal component, to obtain the repeated subproblem of the Ising model on a triangular lattice

$$\begin{array}{ccc}
 0_1 & \longleftrightarrow & ?_2 \\
 \downarrow & \nearrow & \\
 1_0 & &
 \end{array} \tag{13}$$

Our sequential algorithm now faces a problem: after setting the value of 1 at locus 0, the combinations (0,0), (0,1), and (1,0) for loci 1 and 2 all result in the same fitness value. But only one of these choices might be on the path leading to the global optimum if this triangle is only a small subproblem.

Example 2 of Sect. 4 also exhibits this typical form of frustration.

3 The Heuristic

The SGA, with uniform crossover and ordinary mutation, does not seem to have great problems with optimizing randomly sampled U-2ORD-01 functions. But, neither does the simple hill climbing heuristic described below.

Algorithm 1 (Heuristic for U-2ORD-01). Let $f \in \text{U-2ORD-01}$ be the fitness function. This heuristic tries to optimize f by deciding the value of one locus at a time. Initially all loci are “free”; this is formalized by using a control set F , initialized as $F = A$. The heuristic will stop when there are no free loci anymore, i.e., when $F = \emptyset$. A locus i which is not free anymore is called “fixed”; it is fixed to the value $h_i \in \{0, 1\}$. The outcome of the heuristic is the string h with values h_i .

Thus the set F registers the evolution of the algorithm. Each of the free loci is given a score, and the best one will be chosen to be fixed. The score of a locus is based on a study of its incoming and outgoing edges. In what follows, we will define the sets of incoming and outgoing edges of a free locus, and divide them into three subsets, without continually referring to the dependency on F .

Let T_0^i be the set of labeled edges which start in a free locus $i \in F$, and T_1^j the set of labeled edges which end in a free locus $j \in F$. More formally,

$$\begin{aligned}
 T_0^i &= \{j > i; g_{ij} > 0\}, & \text{for } i \in F, \\
 T_1^j &= \{i < j; g_{ij} > 0\}, & \text{for } j \in F.
 \end{aligned} \tag{14}$$

Each set of edges T_b^k is then divided into three groups: Let

$$\begin{aligned}
 L_0^i &= \{j > i; g_{ij} > 0, j \notin F \text{ and } h_j = 0\}, \\
 L_1^j &= \{i < j; g_{ij} > 0, i \notin F \text{ and } h_i = 1\},
 \end{aligned} \tag{15}$$

which might be called the “lost edges” of i resp. j . On the other hand,

$$\begin{aligned} C_0^i &= \{j > i; g_{ij} > 0, j \notin F \text{ and } h_j = 1\}, \\ C_1^j &= \{i < j; g_{ij} > 0, i \notin F \text{ and } h_i = 0\} \end{aligned} \quad (16)$$

are the “promising edges”. Obviously $C_b^k \cap L_b^k = \emptyset$ for all values b , free loci $k \in F$ and states of F . Finally we define the “free edges” of $k \in F$, for each b , as

$$F_b^k = T_b^k \setminus \{C_b^k \cup L_b^k\}. \quad (17)$$

Now if one sets, for each free locus $i \in F$,

$$\begin{aligned} \delta_0^i &= 2g_i(0) + 2 \sum_{i < j \in C_0^i} g_{ij} + \sum_{i < j \in F_0^i} g_{ij}, \\ \delta_1^i &= 2g_i(1) + 2 \sum_{i > j \in C_1^i} g_{ji} + \sum_{i > j \in F_1^i} g_{ji}, \end{aligned} \quad (18)$$

and $\Delta_i = |\delta_0^i - \delta_1^i|$, then the heuristic picks one of the loci with maximal Δ as the next locus to be decided, i.e., it picks an $n \in F$ such that

$$\Delta_n = \max_{j \in F} \Delta_j. \quad (19)$$

The algorithm proceeds by determining the value of h_n , according to the sign of $\delta_0^n - \delta_1^n$: put $h_n = b$ if $\delta_b^n - \delta_{1-b}^n > 0$. One additional rule comes in to decide the value if the difference of the δ ’s is 0: if $n < \ell/2$ then $h_n = 0$, otherwise $h_n = 1$. The algorithm moves on to the next state, by setting $F \leftarrow F \setminus \{n\}$. $\square$

The heuristic contains three parameters. First there is the factor with which the number of promising edges and first order components in the calculation of δ_b^i is multiplied. A value of 2 is chosen because intuitively one sees that the odds of changing a free edge into a lost edge could equal those of changing it into a promising one. A second “parameter” is the rule for picking a locus when there is more than one optimal Δ_i . In practice random selection is used. The third “parameter” is the additional rule for deciding the value h_n when $\Delta_n = 0$. Again, one sees that, at least for randomly sampled instances, loci 0 up to, but not including, $\ell/2$ are best set to 0, and the others to 1.

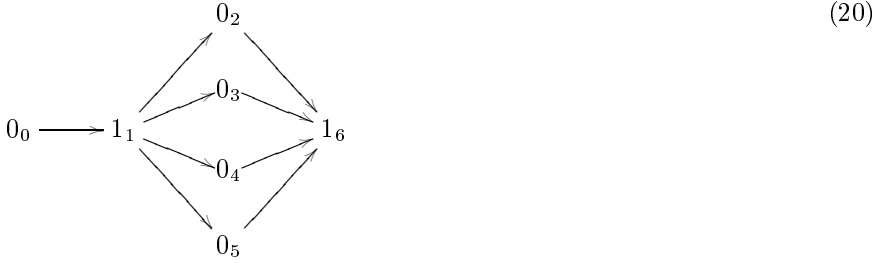
The multiplication factor 2 is used both for the promising edges and the first order component. This leads to another way of viewing the workings of the heuristic. When the value of a locus is decided, the locus is removed and its influence, in the form of promising and lost edges, is transferred to the first order component of the free loci. In other words, the search problem is reduced to a problem with fewer loci but with a modified first order component.

Although one observes that the heuristic optimizes a great number of randomly sampled U-2ORD-01 functions, it certainly cannot do all of them, as we will now show.

4 Where the Heuristic Gets Lost

Example 1 shows a B-2ORD-01 instance which completely misleads the heuristic. It shows that one cannot give a bound on the error of the heuristic. As a matter of convenience, we will add a super-index to the symbol Δ_i used in the algorithm to indicate the number of loci which have been fixed so far.

Example 1. Let $A = \{0, \dots, 6\}$, and let us visually define a B-2ORD-01 instance, together with its optimum:

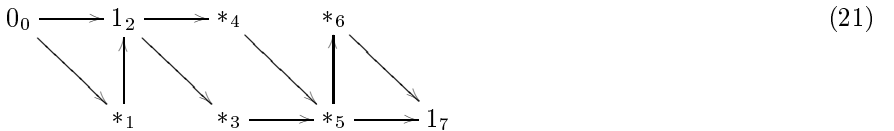


Initially, all loci and edges are free, and the loci have Δ_i^0 values of 1, 3, 0, 0, 0, 0 and 4 respectively. Hence locus 6 gets fixed to 1 ($h_6 = 1$). The loci 2–5 now have one free and one promising edge, which results in $\Delta_2^1 = \dots = \Delta_5^1 = 1$, while we still have $\Delta_0^1 = 1$ and $\Delta_1^1 = 3$. Hence the next step is to fix locus 1 to 0, which leads away from the optimum, because the component g_{01} will not be included in the fitness sum, and the components g_{26} , g_{36} , g_{46} and g_{56} would have been included anyway.

In practice the heuristic starts by fixing the values of all loci which have either only incoming or only outgoing edges to 1 resp. 0. Even then the heuristic is lead away from the optimum: after fixing loci 0 and 6, Δ_1^2 is still greater than Δ_2^2 , Δ_3^2 , Δ_4^2 and Δ_5^2 .

Another typical way of misleading the heuristic is to force it to decide between a number of loci which all have $\Delta = 0$. If then only one “path”, or correct choice, leads to the global optimum, there is no information of where to go, and any search algorithm is forced into exhaustive search.

Example 2. Consider the following frustrated B-2ORD-2NN instance, together with the outcome of the algorithm after three steps:



Initially, the Δ_i^0 values are 2, 0, 0, 0, 0, 0, 0, and 2 respectively. Let us say that the algorithm fixes locus 0 to 0. Now both locus 1 and locus 2 have one promising

edge, which results in $\Delta_1^1 = 1$, $\Delta_2^1 = 1$ and the other Δ_i^1 's remaining unchanged. Locus 7 is fixed to 1, and we obtain $\Delta_1^2 = \Delta_2^2 = \Delta_5^2 = \Delta_6^2 = 1$, and $\Delta_3^2 = \Delta_4^2 = 0$. Assume that locus 2 gets fixed to 1, then $\Delta_1^3 = 0$ and $\Delta_3^3 = \Delta_4^3 = \Delta_5^3 = \Delta_6^3 = 1$. Locus 1 is not important anymore.

If the algorithm now makes the mistake of fixing locus 5 to value 0, then it is lead away from the optimum. Alternatively, assume that locus 4 is fixed to 0. Then $\Delta_3^4 = \Delta_6^4 = 1$ and $\Delta_5^4 = 0$, and the algorithm is lead to the optimum, regardless of further choices.

In this context the Ising models of Figs. 2 and 3 prove to be nice examples. Already in the first step, the heuristic is faced with zero Δ_i 's for all loci! In the square model, all paths lead to the optimum (one can see the fitness landscape as a hyper-paraboloid), whereas in the triangular model, a large number of paths mislead the heuristic.

The case where all loci initially have a Δ_i of 0 shows how symmetry can obstruct the search of the heuristic. This is the reason why the heuristic is fairly good at optimizing random samples: they don't contain too much symmetry. Random samples typically also have a very high degree of connectedness, which indicates a high dimension.

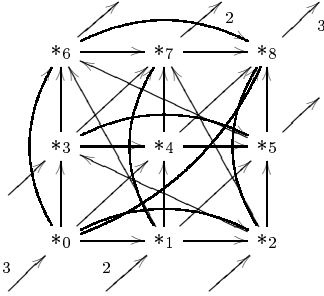


Fig. 3. The $d=2$ anti-ferromagnetic Ising model on a triangular lattice, with 3^2 sites, as an instance U-2ORD-01.

5 The Heuristic and the Simple Genetic Algorithm

Before we can start comparing the workings of the heuristic and the SGA, we have to decide on a stopping criterion for the latter, bearing in mind that we cannot know the maximum of our fitness functions at all. A natural choice is to stop the SGA when its population is dominated by a string with fitness value at least the value provided by the heuristic. This also allows for an efficiency comparison between the two algorithms.

5.1 The SGA's Fixing Chronology

Given the evolution $(\mathcal{P}_t)_{t=0}^T$ of an initial population $\mathcal{P}_0$ of the SGA, we can extract *whether* and *when* a certain locus got fixed to a final position if we assume that when the SGA stops at time T , a certain string has taken over a large proportion of the population. Let us call this string m , and define, for each locus i , the set

$$A_i = \left\{ 0 \leq t \leq T; \forall t \leq u \leq T. \frac{|\{s \in \mathcal{P}_u; s_i = m_i\}|}{|\mathcal{P}_u|} \geq \frac{7}{8} \right\}. \quad (22)$$

Obviously the value $7/8$ is a parameter we can freely choose. It seems a good idea to set it well beyond $1/2$.

We say that locus i got fixed if $A_i \neq \emptyset$, in which case its fixing time $I_{\text{SGA}}(i)$ is defined to be the earliest evolution step found in A_i , i.e., $I_{\text{SGA}}(i) = \min A_i$. If $A_i = \emptyset$, the locus did not get fixed; by convention we set its fixing time to the last step in the evolution of the population: $I_{\text{SGA}}(i) = T$.

Consider this example of a fixing chronology (short for ‘set of fixing times’):

$$5 \quad 29 \quad 28 \quad 5 \quad 5 \quad 35 \quad 6 \quad 9 \quad 23 \quad 10 \quad 10 \quad 35, \quad (23)$$

produced by an SGA with $\ell = 12$ which ran for 35 iterations. It shows that sometimes the SGA fixes a number of loci simultaneously, as it processes the loci in parallel. We also see that there can be a significant number of iterations between two fixing times.

5.2 The Heuristic's Fixing Chronology

Although the heuristic is sequential in nature, and its fixing times can be defined in a trivial way, it can also be thought of as processing a number of loci in parallel, namely those which have equal Δ_i values.

To take this effect into account, we modify the way the heuristic's chronology is computed.

Algorithm 2 (Fixing Times of the Heuristic). Initially, each locus i 's fixing time $I_{\text{H}}(i)$ is undefined, and a counter k is set to 0. Every time a locus is removed from F , the counter k is changed and a number of fixing times are set, in the following way.

At stage F , let Γ_F be the set of loci i which have an optimal Δ_i , and out of which one locus γ is picked to be fixed. If $I_{\text{H}}(\gamma)$ has already received a value, then k is set to $I_{\text{H}}(\gamma) + 1$. Next, the fixing times $I_{\text{H}}(\lambda)$ are set to k for all $\lambda \in \Gamma_F$ with undefined fixing time. Finally, k is incremented by 1. $\square$

As an example, let us apply this algorithm to obtain the fixing chronology associated with the heuristic's evolution of Example 2 in Sect. 4.

Example. At the initial stage $F = A$, two loci have an optimal Δ_i : $\Delta_0^0 = 2$ and $\Delta_7^0 = 2$, so $\Gamma_A = \{0, 7\}$. Locus 0 is chosen and since the fixing times of loci 0 and 7 are still undefined, they are set to 0, and the counter k is incremented from 0 to 1. After the first locus has been fixed, we see that $\Delta_7^1 = 2$ is still the greatest of the Δ_i^1 , resulting in $\Gamma_{A \setminus \{0\}} = \{7\}$. Since $I_H(7)$ has already been set, we set k to $I_H(7) + 1 = 1$, and increment it once more to obtain $k = 2$. In the third step, $\Gamma_{A \setminus \{0, 7\}} = \{1, 2, 5, 6\}$, and since their fixing times are still undefined, we set $I_{H(1)} = I_{H(2)} = I_{H(5)} = I_{H(6)} = k = 2$, and we increment k by one.

If we assume that the heuristic proceeded by fixing loci 4, 6, 5, 3 and 1, we obtain the fixing chronology 0, 2, 2, 3, 3, 2, 2, 1.

5.3 Results

Let us start by giving an overview of all parameters involved. The SGA, with uniform crossover (rate 0.7), and ordinary bit-flip mutation, (rate 0.003), ranking selection and a population size of 128 strings, reaches the heuristic's value if, within 1000 iterations, a string with at least this value takes up at least 50 percent of the population. Each (average, standard deviation) pair in Tables 1 and 2 is the result of averaging the observed correlation over 20 instances. The correlations are obtained by averaging both the heuristic's and the SGA's chronologies over 10 runs.

Randomly sampled B-2ORD-01 instances are defined by letting $g_{ij} = 1$ with probability p . Table 1 shows observed averages and standard deviations of the correlation between the chronologies for different p values. We also observed that the SGA almost always reached the value found by the heuristic, that it usually needed less than 100 iterations, and that different runs of the heuristic almost always produced the same chronology.

To obtain Table 2, we set the range of the U-2ORD- k NN components to $\{0, \dots, 15\}$. The correlation values, the percentage of successful SGA runs and the bit decidability values all indicate that the lower the span of the edges, hence the lower the degree of connectedness and the flatter the landscape, the harder the functions are for the SGA.

Finally, it is of interest to compare the heuristic and the SGA on the highly symmetrical models from statistical physics. As has been announced, the triangular model is a lot harder to optimize than the square model: not even once out of 50 runs, the SGA could reach the heuristic's value, whereas the square model resulted in 28 successful runs out of 50. Moreover, all heuristic runs on the square model resulted in the same optimum, whereas only 33 out of 50 did so for the triangular model. Computing the correlation is not all that useful for these two models, since we know that all Δ 's are initially 0. However, we observe relatively high correlations, simply because the fact that each different SGA run produces a different chronology, which, in the end, averages to a flat chronology. Both models share the worst value $\tau = 1$.

Table 1. Average and standard deviation of the correlation between the chronologies of heuristic and SGA, for randomly sampled B-2ORD-01 instances. The parameter p gives the probability of g_{ij} being 1.

p	$ A = 32$		$ A = 80$	
	avg	std dev	avg	std dev
0.1	0.74	0.06	0.67	0.11
0.15	0.76	0.10	0.69	0.12
0.2	0.83	0.09	0.80	0.12
0.3	0.86	0.09	0.83	0.07
0.4	0.85	0.09	0.85	0.09
0.5	0.87	0.12	0.84	0.11
0.6	0.91	0.08	0.88	0.03
0.7	0.90	0.08	0.90	0.03
0.8	0.92	0.06	0.87	0.08
0.9	0.93	0.02	0.91	0.02
0.95	0.93	0.02	0.90	0.02
0.99	0.93	0.02	0.90	0.02
1.0	0.94	0.01	0.91	0.01

Table 2. Average and standard deviation of the correlation between the chronologies of heuristic and SGA, for randomly sampled U-2ORD- k NN instances with components uniformly drawn from the range $\{0, \dots, 15\}$. We also give the percentage of SGA runs which reached the heuristic's value, and the percentage of runs amongst them with higher value than the heuristic's. Finally we stress the strong correlation with the bit decidability metric τ , introduced in [6].

k	avg	std dev	pct. success	pct. better	τ
2	0.38	0.15	32	52	0.87
3	0.41	0.12	20	59	0.85
5	0.46	0.12	27	57	0.82
7	0.54	0.14	29	72	0.79
10	0.51	0.15	31	65	0.75
15	0.58	0.13	50	70	0.67
20	0.69	0.07	57	61	0.59
25	0.68	0.16	59	45	0.51
40	0.73	0.10	62	37	0.26
50	0.79	0.07	96	4	0.15
60	0.84	0.03	96	0	0.15
70	0.84	0.10	93	4	0.15
79	0.90	0.03	98	0	0.14

5.4 Heuristic versus SGA

Which algorithm is best? It is important to note that we stop the SGA when it reaches the heuristic's value, while it might have found better values after some more iterations. Even if we take this into account, we are lead to believe that the heuristic is a lot more efficient in providing us with a decent approximation of the optimum, at least on a majority of reasonable instances. But, we have no guarantee at all that the SGA will always do better than the heuristic on the more difficult instances.

6 Conclusion

Having observed the (sometimes very high) correlation between the chronologies of the heuristic and the SGA, we have a means of explaining and predicting the search dynamics of the SGA. More precisely, for a large class of U-2ORD-01 instances we can algorithmically predict *how* the SGA proceeds from a random population to one containing the first local optimum it comes across. Often, this local optimum is the one found by the heuristic.

While other researchers often consider fitness functions with high or varying epistatic interaction, we focus on second order interaction. We prove that the related optimization problem 2ORD is NP-equivalent, hence certainly rich enough for general analysis. Moreover, the connection with statistical physics, a very well established field, and with some other search problems, proves to be a very valuable resource.

From the theoretical point of view, the next step now is to formalize the correspondence between the two algorithms. This might be achieved by constructing a chain of intermediate algorithms, closely related to each other. E.g., one might be able to prove the correspondence between the heuristic and a more probabilistic version of it, which adheres closer to the SGA. Or, one might study the correspondence between the correlation values and the bit decidability values. It is also interesting to study the way the SGA parameters, like crossover and mutation rate, selection scheme, population size etc., influence the chronologies produced by the algorithm.

On the practical side, it would be instructive to know if the dashed arrows in Fig. 1 can be proved or disproved, maybe allowing for some other real world problems to be reduced to the domain of the heuristic.

Acknowledgements

The authors wish to thank J. Naudts (University of Antwerp, Dept. of Physics) for pointing out the connection with statistical physics, and all fruitful discussions which followed after that. They are also grateful for the comments of the two anonymous referees.

References

1. Davidor, Y. (1991): "Epistasis Variance: A Viewpoint on GA-Hardness". In: G.J.E. Rawlins ed., *Foundations of Genetic Algorithms*, Morgan Kaufmann Publishers, San Francisco, Ca., pp. 23–35.
2. Hogg, T., Hubermann, B., Williams, C., eds. (1996): "Special Volume on Frontiers in Problem Solving: Phase Transitions and Complexity". *Artificial Intelligence*, vol. 82.
3. Jones, T., Forrest, S. (1995): "Fitness Distance Correlation as a Measure of Problem Difficulty for Genetic Algorithms". In: L.J. Eshelman ed., *Proceedings of the Sixth International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, San Francisco, Ca., pp. 184–192.
4. Kauffman, S.A. (1989): "Adaptation on Rugged Fitness Landscapes". In: D.L. Stein, ed., *Lectures in the Sciences of Complexity*, Addison–Wesley, Reading, Ma., pp. 527–618.
5. Liggett, T.M. (1985): "Interacting Particle Systems". Springer Verlag, Heidelberg, Germany.
6. Naudts, B., Suys, D., Verschoren, A. (1997): "Epistasis as a Basic Concept in Formal Landscape Analysis". In: Th. Baeck, ed., *Proceedings of the 7th International Conference on Genetic Algorithms*, Morgan Kaufmann, San Francisco, Ca., pp. 65–72.
7. Naudts, B., Verschoren, A. (1997): "Second Order Functions for the SGA". In preparation.
8. Naudts, B. (1998): "Network Balancing: SGA versus Hillclimber". In preparation.
9. Naudts, B. (1998): "A Theoretical Approach to GA-Hardness". Ph.D. thesis, University of Antwerp, RUCA, Belgium. In preparation.
10. Papadimitriou, C.H. (1993): "Computational Complexity". Addison–Wesley, Reading, Ma.
11. Salkin, H.M. (1975): "Integer Programming". Addison–Wesley, Reading, Ma.
12. Thompson, C.J. (1972): "Mathematical Statistical Mechanics". Macmillan, New York.

Asymptotical Convergence Rates of Simple Evolutionary Algorithms under Factorizing Mutation Distributions

Günter Rudolph

Universität Dortmund, Fachbereich Informatik, LS XI, D-44221 Dortmund

Abstract. The standard choice for mutating an individual of an evolutionary algorithm with continuous variables is the normal distribution. It is shown that there is a broad class of alternative mutation distributions offering local convergence rates being asymptotical equal to the convergence rates achieved with normally distributed mutations. Such mutation distributions must be factorizing and the absolute fourth moments must be finite. Under these conditions an asymptotical theory of the convergence rates of simple evolutionary algorithms can be established for the entire class of distributions.

1 Introduction

The standard choice to represent mutations in evolutionary models dealing with continuous quantities is the normal distribution. This choice is usually justified by the central limit theorem: Since mutations in nature are caused by a variety of physical and chemical influences that are not identifiable or measurable to a degree that allows for a deterministic model, these influences are considered as independent random perturbations whose normed sum approaches a normal random variable in the limit, provided that the first two absolute moments of the distributions of these random perturbations are finite and that the so-called Lindeberg condition is obeyed. Therefore it is not surprising that evolutionary algorithms with continuous search space model mutations by normally distributed random variables as well. But the biological original needs not necessarily be the best choice when mutations play the role of an exploration operator—as it is the case in evolutionary algorithms (EAs). It was noted several times [1, 2, 3] that mutation distributions with slowly (i.e., not exponentially) decreasing tails should offer a larger probability to escape from local optima (also see fig. 1 & 2). Although this claim is certainly correct if the variance is held fixed, it is still an open question whether this theoretical property carries over to practical EAs employing an auto-adaptive adjustment of the variances. But this question will not be addressed here. Instead, it is investigated to which extent non-normal mutation distributions may affect the local convergence behavior of evolutionary algorithms.

Two simple evolutionary algorithms will be studied here: The $(1+1)$ -EA and the $(1, \lambda)$ -EA. The first one generates a single offspring by mutation and accepts the offspring only if it is better than the parent, whereas the latter one generates

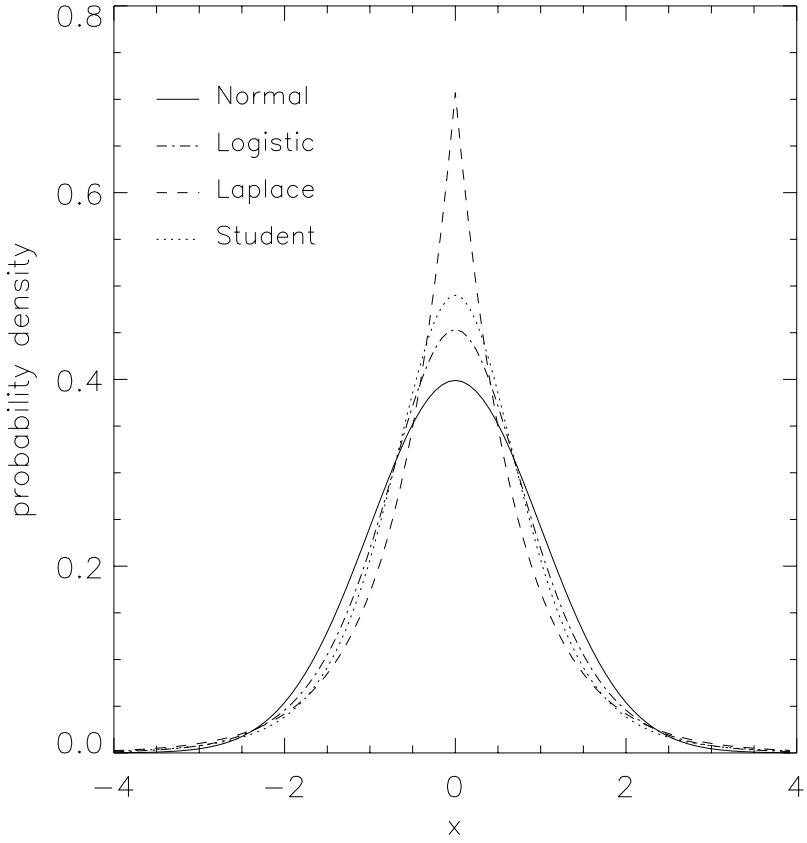


Fig. 1. The shape of some symmetrical univariate distributions with zero mean and unit variance.

distribution	density function
Normal	$\exp(-x^2/2)/\sqrt{2\pi}$
Logistic	$\pi \operatorname{sech}^2(\pi x/\sqrt{12})/\sqrt{48}$
Laplace	$\exp(- x \sqrt{2})/\sqrt{2}$
Student (d=5)	$8(1+x^2/3)^{-3}/\sqrt{27\pi^2}$

Table 1. Probability density functions of some symmetrical univariate distributions with zero mean and unit variance.

$\lambda \geq 2$ offspring independently with the same mutation distribution and chooses the best offspring among the λ offspring to serve as new parent (regardless of

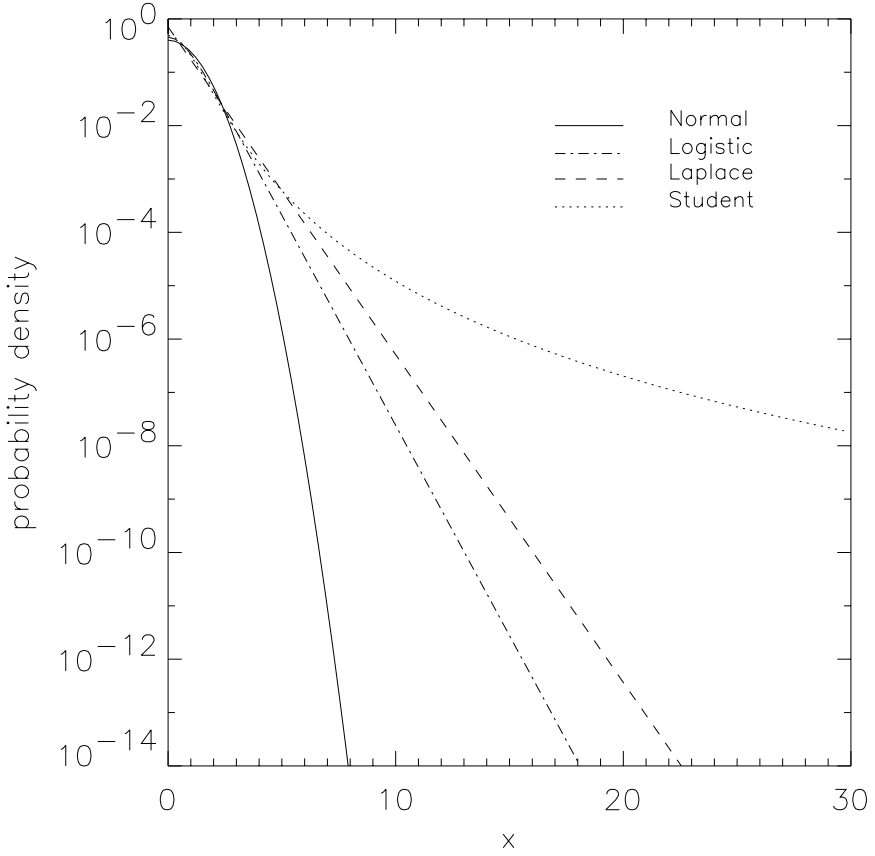


Fig. 2. The decay of the right tails of the symmetric Normal, Logistic, Laplace, and Student distribution. The tails of the first three distributions decline exponentially whereas the tail of the Student distribution (with 5 degrees of freedom) follows a power law.

the quality of the old parent). If $\theta \in \mathbb{R}^n$ denotes the current position of the EA in the search space, then a mutation is modeled by adding a random vector $\mathbf{Z}$ that must fulfill some conditions (details will follow shortly). Thus, an offspring $\mathbf{X}$ is represented by the random variable $\mathbf{X} = \theta + \mathbf{Z}$.

The test problem is the minimization of the objective function $f(\mathbf{x}) = \mathbf{x}' \mathbf{x}$ with $\mathbf{x} \in \mathbb{R}^n$. It will be assumed that n is large ($n \geq 100$). This test function reflects to some extent the case of a local optimum, and it is usually used to assess the local convergence behavior of evolutionary algorithms. To be comparable to previous work, this common practice is followed here.

2 Asymptotical Results

The *fundamental assumption* made in the remainder is that it will be postulated that the product moments of random vector $\mathbf{Z}$ do exist up to order 4. Further conditions on $\mathbf{Z}$ are given in the definition below which specifies the distribution class of random vector $\mathbf{Z}$.

Definition 1. The distribution of random vector $\mathbf{Z}$ is termed a *mutation distribution* if $\mathbb{E}[\mathbf{Z}] = \mathbf{0}$. In this case, random vector $\mathbf{Z}$ is called a *mutation vector*. A mutation distribution is said to be *factorizing* if the joint probability density function of the mutation vector $\mathbf{Z}$ can be written as

$$f_{\mathbf{Z}}(z_1, \dots, z_n) = \prod_{i=1}^n f_{Z_i}(z_i)$$

with $f_{Z_1}(\cdot) = \dots = f_{Z_n}(\cdot)$ where n denotes the dimension. $\square$

Let $\mathbf{Z}$ possess a factorizing mutation distribution. Since the random objective function value of an offspring is given by

$$f(\theta + \mathbf{Z}) = \sum_{i=1}^n (\theta_i + Z_i)^2$$

each of the summands above is mutually independent to the remaining ones. As a consequence, the objective function value is representable by a sum of independent random variables. If such a sum is appropriately normed, then its distribution converges to some limit distribution as $n \rightarrow \infty$. This fact will be exploited to develop an asymptotical theory with regard to the convergence rates. In order to obtain the desired norming constants some preparatory results are necessary.

Lemma 2. Let Z be a symmetrical random variable with $\mathbb{E}[Z^{2k-1}] = 0$ for $k \in \mathbb{N}$ and set $X = \theta + Z$ with $\theta \in \mathbb{R}$. Then $\mathbb{E}[X^2] = \theta^2 + \mathbb{E}[Z^2]$ and $\mathbb{V}[X^2] = 4\theta^2 \mathbb{E}[Z^2] + \mathbb{V}[Z^2]$. $\square$

The proof of this lemma is trivial and therefore omitted while the next result is an immediate consequence of the lemma above.

Proposition 3. Let $Z_1, \dots, Z_n$ be independent and identically distributed symmetrical random variables with $\mathbb{E}[Z_i^{2k-1}] = 0$ for $i = 1, \dots, n$ and $k \in \mathbb{N}$. If $X_i = \theta_i + Z_i$ and $S_n = \sum_{i=1}^n X_i^2$ then

$$\begin{aligned} \mathbb{E}[S_n] &= \|\theta\|^2 + n \mathbb{E}[Z_1^2] \\ \mathbb{V}[S_n] &= 4\|\theta\|^2 \mathbb{E}[Z_1^2] + n \mathbb{V}[Z_1^2] \end{aligned}$$

where $\theta \in \mathbb{R}^n$ and $\|\cdot\|$ denotes the Euclidean norm. $\square$

The central limit theorem (see [4], p. 262) ensures that the distribution of the appropriately normed random scalar product $S_n = \mathbf{X}' \mathbf{X} = \|\mathbf{X}\|^2$ converges weakly to the standard normal distribution. Thus, since

$$\frac{S_n - \mathbb{E}[S_n]}{\mathbb{V}[S_n]^{1/2}} \longrightarrow N \sim N(0, 1)$$

as $n \rightarrow \infty$ one obtains

$$\begin{aligned} S_n &\approx \mathbb{E}[S_n] + \mathbb{V}[S_n]^{1/2} \cdot N \\ &= \|\theta\|^2 + n \mathbb{E}[Z^2] + (4 \|\theta\|^2 \mathbb{E}[Z^2] + n \mathbb{V}[Z^2])^{1/2} \cdot N. \end{aligned} \quad (1)$$

Let $\eta^2 = \mathbb{V}[Z] = \mathbb{E}[Z^2]$ and suppose that $\mathbb{V}[Z^2] = a \eta^4 = a \mathbb{V}[Z]^2$ for some $a > 0$. Then the random variable $S_n/\|\theta\|^2$ can be written as

$$\frac{S_n}{\|\theta\|^2} \approx 1 + \frac{1}{n} \left[\gamma^2 + \left(4 \gamma^2 + \frac{a \gamma^4}{n} \right)^{1/2} \cdot N \right] \quad (2)$$

where $\gamma = n \eta / \|\theta\|$. After having established this approximation one can begin to calculate the expected asymptotical progress rates for the $(1+1)$ -EA and the $(1, \lambda)$ -EA provided that the objective function is $f(\mathbf{x}) = \|\mathbf{x}\|^2$. At first consider the $(1+1)$ -EA. Assume the current position is $\theta \in \mathbb{R}^n$. Since the $(1+1)$ -EA only accepts improvements the relative progress is given by

$$\mathbb{E} \left[\max \left\{ \frac{\|\theta\|^2 - \|\theta + \mathbf{Z}\|^2}{\|\theta\|^2}, 0 \right\} \right] = \mathbb{E} \left[\max \left\{ 1 - \frac{S_n}{\|\theta\|^2}, 0 \right\} \right].$$

It will be useful to normalize the relative progress by the dimension n . This quantity will be called *normalized progress*. Owing to eqn. (2) one obtains the normalized progress

$$\mathbb{E} \left[\max \left\{ n \left(1 - \frac{S_n}{\|\theta\|^2} \right), 0 \right\} \right] \approx \mathbb{E} [\max\{-\gamma^2 + \gamma \sqrt{4 + a \gamma^2/n} \cdot N, 0\}]. \quad (3)$$

Proposition 4. Let $\eta^2 = \mathbb{V}[Z] = \mathbb{E}[Z^2]$ and suppose that $\mathbb{V}[Z^2] = a \eta^4 = a \mathbb{V}[Z]^2$ for some $a > 0$. If $n \gg 1$ then the expected normalized progress rate of the $(1+1)$ -EA is asymptotically given by

$$h(\gamma, a, n) = 2 \gamma \sqrt{1 + \frac{a \gamma^2}{4n}} \varphi \left(\frac{\gamma}{2} \sqrt{\frac{4n}{4n + a \gamma^2}} \right) - \gamma^2 \Phi \left(-\frac{\gamma}{2} \sqrt{\frac{4n}{4n + a \gamma^2}} \right)$$

with $\gamma = n \eta / \|\theta\|$ and where $\varphi(\cdot)$ and $\Phi(\cdot)$ denote the probability density and distribution function of the standard normal distribution, respectively.

Proof: Let $W = -\gamma^2 + \gamma \sqrt{4 + a \gamma^2/n} \cdot N$ with $N \sim N(0, 1)$. The expected normalized progress as given in eqn. (3) becomes $\mathbb{E}[\max\{W, 0\}]$. Since $\max\{W, 0\} = W \cdot 1_{(0, \infty)}(W)$, where $1_A(x)$ is the indicator function of set A , one obtains

$$\mathbb{E}[\max\{W, 0\}] = \mathbb{E}[W \cdot 1_{(0, \infty)}(W)] = \int_0^\infty \frac{w}{\gamma \sqrt{4 + a \gamma^2/n}} \varphi \left(\frac{w + \gamma^2}{\gamma \sqrt{4 + a \gamma^2/n}} \right) dw$$

where $\varphi(\cdot)$ is the probability density function of the standard normal distribution. The determination of the integral yields the desired result. $\square$

In principle, the same kind of approximation was presented in [5] for the special case of normally distributed mutations. Additionally, it was argued that the term $a\gamma^2/n$ in eqn. (3) becomes small for large n so that this term can be neglected. As a consequence, the random variable W reduces to $\widetilde{W} = -\gamma^2 + 2\gamma \cdot N$ and the expected normalized progress becomes

$$\tilde{h}(\gamma) = 2\gamma \cdot \varphi(\gamma/2) - \gamma^2 \cdot \Phi(-\gamma/2)$$

attaining its maximum $\tilde{h}(\gamma^*) = 0.404913$ at $\gamma^* = 1.224$ which is exactly the same result established 20 years earlier by Rechenberg [6]. Since all factorizing mutation distributions (with finite absolute moments) in Proposition 4 only distinguish from each other by the constant a , an analogous argumentation for an arbitrary factorizing mutation distribution leads to the result that the normalized improvement is asymptotically equal for all factorizing mutation distributions. Evidently, this kind of approximation is too rough to permit a sound comparison of the progress offered by different factorizing mutation distributions.

distribution	a	γ^*	$h(\gamma^*, a, 100)$
Normal	2	1.24389	0.40801
Logistic	16/5	1.25648	0.40992
Laplace	5	1.27639	0.41289
Student ($d = 5$)	8	1.31273	0.41811

Table 2. Optimal expected normalized progress rates for the $(1 + 1)$ -EA for some factorizing mutation distributions in case of dimension $n = 100$ under the assumption $E[\max\{n(1 - S_n/\|\theta\|^2), 0\}] \equiv h(\gamma, a, n)$.

Table 2 summarizes the optimal expected normalized progress rates for some factorizing mutation distributions under the assumption that the approximation of Proposition 4 is exact. The surprising observation which can be made from Table 2 is that the normal distribution is identified as yielding the least progress compared to the other distributions, provided that the assumption $h(\gamma, a, n) \equiv E[\max\{n(1 - S_n/\|\theta\|^2), 0\}]$ holds true. The validity of this assumption, however, deserves careful scrutiny since the norming constants $a_n = E[S_n]$ and $b_n^2 = V[S_n]$ used in the central limit theorem do not necessarily represent the best choice for a rapid approach to the normal distribution. In fact, there may exist constants α_n, β_n obeying $\beta_n \sim b_n$ and $\alpha_n - a_n = o(b_n)$ that lead much faster to the limit [7, p. 262]. As a consequence, it may happen that the ranking of the distributions in Table 2 is reversed after using these (unknown) constants. Thus, unless the error of the approximation of Proposition 4 has been quantified, this kind of approximation is also too rough to permit a sound ranking of the

mutation distributions. Nevertheless, the small differences in Table 2 provide evidence that (at least for $n \geq 100$) every factorizing mutation distribution offers a local convergence rate being comparable to that of a normal distribution.

The quality of the approximation in Proposition 4 can be checked in case of normally distributed mutations. As shown in [5], the random variable $V_n = S_n/\|\theta\|^2$ follows a noncentral χ^2 distribution with probability density function

$$f_{V_n}(v; \delta) = \frac{\delta^2}{2} v^{(n-2)/4} \exp\left(-\frac{\delta^2(v+1)}{2}\right) I_{n/2-1}(\delta^2 \sqrt{v}) \cdot 1_{(0, \infty)}(v)$$

where $I_m(\cdot)$ denotes the m th order modified Bessel function of the first kind and where $\delta = \|\theta\|/\eta$ is the noncentrality parameter. Since $V_n > 0$ one obtains $\max\{n(1 - V_n), 0\} = n(1 - V_n) \cdot 1_{(0,1)}(V_n)$ and hence

$$g(n, \delta) = \mathbb{E}[\max\{n(1 - V_n), 0\}] = n \int_0^1 (1 - v) f_{V_n}(v; \delta) dv. \quad (4)$$

This integral can be evaluated numerically for any given n and δ . Since $\delta = \|\theta\|/\eta$ and $\gamma = n\eta/\|\theta\|$ it remains to maximize the function $g(n, \delta) = g(n, n/\gamma)$ with respect to $\gamma > 0$. For example, in case of $n = 100$ a numerical optimization leads to $\gamma^* = 1.224$ with $g(n, n/\gamma^*) = 0.4049$. Figures 3 & 4 show that the optimal variance factor γ^* and the optimal normalized progress $g(n, n/\gamma^*)$ quickly stabilizes for increasing dimension n . In fact, the theoretical limits are almost reached for $n = 30$.

A similar investigation might be made for other mutation vectors $\mathbf{Z}$ with factorizing mutation distributions, if the distribution of $S_n = \sum_{i=1}^n (\theta_i - Z_i)^2$ were to be known. But this does not seem to be the case. For this reason and realizing that the knowledge of the true limits is of no practical importance, it is refrained from taking the burden of determining the density of S_n for other mutation vectors.

Even numerical simulations do not easily lead to a statistically supported ranking: Although the average of the outcomes of random variable

$$Y = \max\{n(1 - S_n/\|\theta\|^2), 0\}$$

is an unbiased point estimator of the expectation, there is neither a standard parametric nor standard nonparametric test permitting a statistically supported decision which mean is the largest among the random variables Y generated from different mutation distributions. For example, the parametric t -test presupposes at least approximative normality of Y whereas the nonparametric tests require the continuity of the distribution function of Y . Neither of these requirements is fulfilled, so that it would be necessary to develop a specialized test for this kind of random variables. This is certainly beyond the scope of this paper.

Instead, the attention is devoted to the expected progress rates of the $(1, \lambda)$ -EA. Since this EA generates $\lambda \geq 2$ offspring independently with the same distribution and accepts the best among them, the expected progress is simply

$$\mathbb{E}\left[\max_{i=1, \dots, \lambda} \{\|\theta\|^2 - \|\theta + \mathbf{Z}_i\|^2\}\right].$$

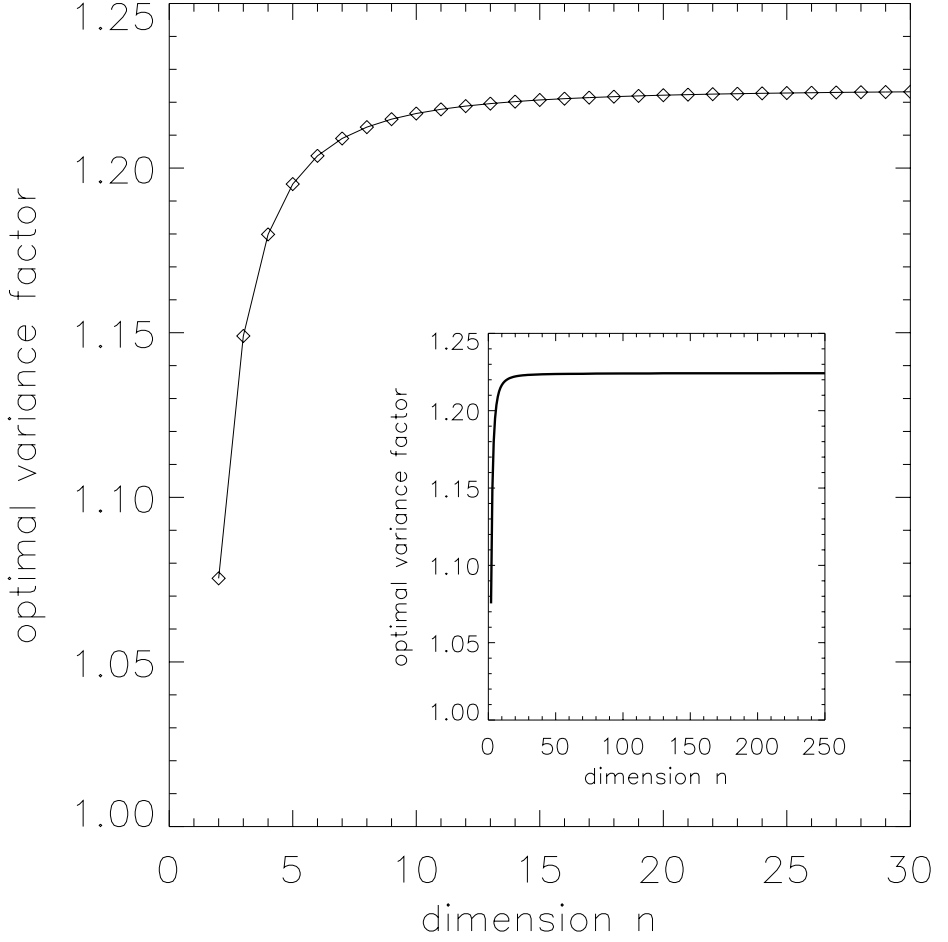


Fig. 3. The optimal variance factor γ^* in case of normal mutation vectors for increasing dimension n .

Following the lines of Proposition 4 and owing to eqn. (2) the normalized expected progress is approximately

$$h(\gamma, a, n) = -\gamma^2 + \gamma \sqrt{4 + a \gamma^2/n} \cdot \mathbb{E}[N_{\lambda:\lambda}] \quad (5)$$

where $N_{\lambda:\lambda}$ denotes the maximum of λ independent and identically distributed standard normal random variables. Let $c_\lambda = \mathbb{E}[N_{\lambda:\lambda}]$. Then the optimal expected normalized progress rate of the $(1, \lambda)$ -EA is attained at

$$\gamma^* = \left(\frac{2 c_\lambda^2}{1 - a c_\lambda^2/n + \sqrt{1 - a c_\lambda^2/n}} \right)^{1/2}$$

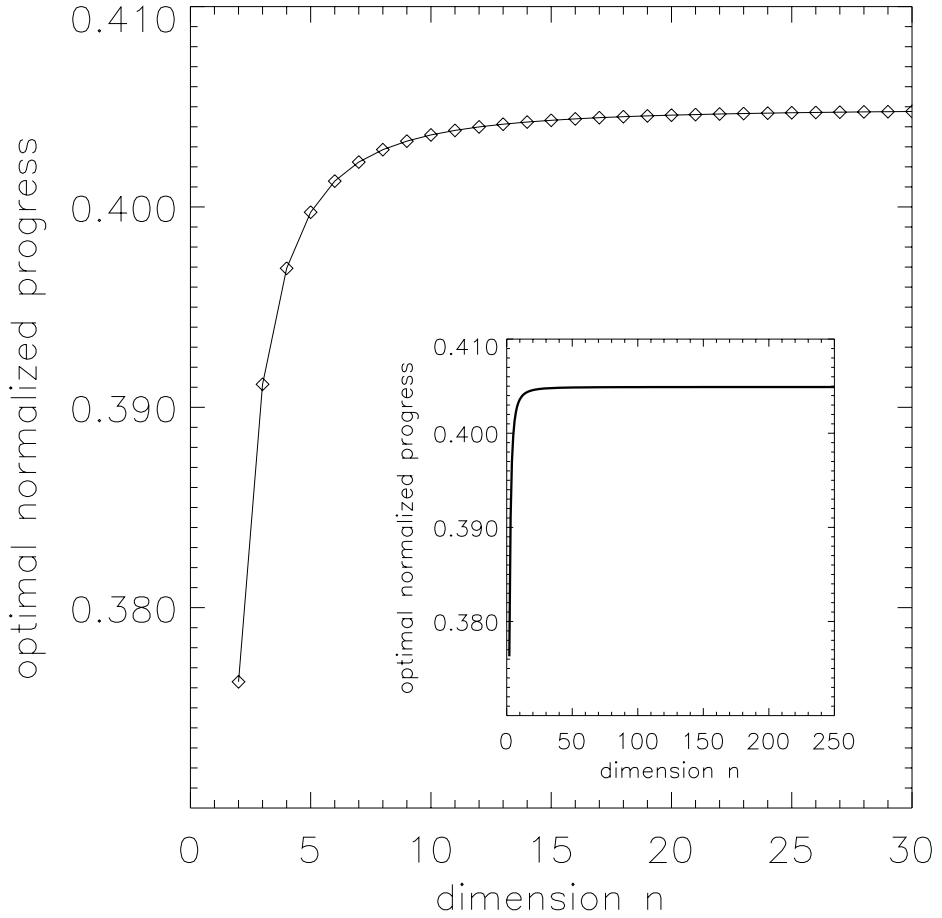


Fig. 4. The optimal normalized progress $g(n, n/\gamma^*)$ in case of normal mutation vectors for increasing dimension n .

which reduces to $\tilde{\gamma}^* = c_\lambda$ as $n \rightarrow \infty$. In general, the relation $h(\gamma^*, a, n) > c_\lambda^2$ is valid. Moreover, $h(\gamma, a + \epsilon, n) > h(\gamma, a, n)$ for arbitrary $\gamma > 0$ and $\epsilon > 0$ which follows easily from eqn. (5). Consequently, the expected progress becomes larger for increasing $a > 0$, provided that the approximation given in (2) holds with equality. But it has been seen in case of the $(1+1)$ -EA that this approximation does not permit a sound ranking of the distributions. At this point there might arise the question for which purpose the approximations presented in this paper are good for at all. The answer is given in the next section.

3 Conclusions

Under the conditions of the central limit theorem an asymptotical theory of the expected progress rates of simple evolutionary algorithms has been established. If the mutation distributions are factorizing and possess finite absolute moments up to order 4, then each of these distributions offer an almost equally fast approach to the (local) optimum. The optimal variance adjustment w.r.t. fast local convergence is of the type $\eta_k = \gamma \|\mathbf{X}_k - \mathbf{x}^*\|/n$ for each of the distributions considered here. This implies that the self-adaptive adjustment of the “step sizes” originally developed for normal distributions needs not be modified in case of other factorizing mutation distributions. In the light of the theory developed in [8] it may be conjectured that these results carry over to population-based EAs *without* crossover or recombination.

Finally, notice that Student’s t -distribution with d degrees of freedom converges weakly to the normal distribution as $d \rightarrow \infty$ whereas it is called the Cauchy distribution for $d = 1$. All results remain valid for $d \geq 5$. Lower values of d cannot be investigated within the framework presented here, since it was presupposed that the absolute moments of $\mathbf{Z}$ are finite up to order 4. If these moments do not exist the central limit theorem does not hold true. Rather, then there emerges an entire class of limit distributions [9] as already mentioned in [1]. But this case is beyond the scope of this paper and it remains for future research.

Acknowledgment

Besides my thanks to all anonymous reviewers whose suggestions led to several improvements, special thanks must be addressed to one of the reviewers who detected a severe flaw in my argumentation following Proposition 4. Therefore, the part after Proposition 4 was completely revised. As a result, the message of this paper is completely different from the original (wrong) one.

Finally, it should be mentioned that this work is a result of the *Collaborative Research Center “Computational Intelligence” (SFB 531)* supported by the German National Science Foundation (DFG).

References

1. C. Kappler. Are evolutionary algorithms improved by large mutations? In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Parallel Problem Solving From Nature—PPSN IV*, pages 346–355. Springer, Berlin, 1996.
2. X. Yao and Y. Liu. Fast evolutionary programming. In L. J. Fogel, P. J. Angeline, and T. Bäck, editors, *Proceedings of the Fifth Annual Conference on Evolutionary Programming*, pages 451–460. MIT Press, Cambridge (MA), 1996.
3. X. Yao and Y. Liu. Fast evolution strategies. In P. J. Angeline, R. G. Reynolds, J. R. McDonnell, and R. Eberhart, editors, *Proceedings of the Sixth Annual Conference on Evolutionary Programming*, pages 151–161. Springer, Berlin, 1997.

4. W. Feller. *An Introduction to Probability Theory and Its Applications, Vol. 2*. Wiley, New York, 2nd edition, 1971.
5. T. Bäck, G. Rudolph, and H.-P. Schwefel. Evolutionary programming and evolution strategies: Similarities and differences. In D. B. Fogel and W. Atmar, editors, *Proceedings of the 2nd Annual Conference on Evolutionary Programming*, pages 11–22. Evolutionary Programming Society, La Jolla (CA), 1993.
6. I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog Verlag, Stuttgart, 1973.
7. Y. S. Chow and H. Teicher. *Probability Theory*. Springer, New York, 1978.
8. G. Rudolph. *Convergence Properties of Evolutionary Algorithms*. Kovač, Hamburg, 1997.
9. B. V. Gnedenko and A. N. Kolmogorov. *Limit Distributions for Sums of Independent Random Variables*. Addison-Wesley, Reading (MA), revised edition, 1968.

Adaptation on the Evolutionary Time Scale: A Working Hypothesis and Basic Experiments

Ralf Salomon and Peter Eggenberger

AI Lab, Computer Science Department, University of Zürich
Winterthurerstrasse 190, 8057 Zurich, Switzerland
FAX: +41-1-635 68 09; Email: {salomon|eggen}@ifi.unizh.ch

Abstract. In the pertinent literature, an ongoing discussion can be found about whether evolutionary algorithms are better suited for optimization or adaptation. Unfortunately, the pertinent literature does not offer a definition of the difference between adaptation and optimization. As a working hypothesis, this paper proposes adaptation as tracking the moving optimum of a dynamically changing fitness function as opposed to optimization as finding the optimum of a static fitness function. The results presented in this paper suggest that providing enough variation among the population members and applying a selection scheme is sufficient for adaptation. The resulting performance, however, depends on the problem, the selection scheme, the variation operators, as well as possibly other factors.

1 Introduction

Evolutionary algorithms (EAs) are a class of stochastic optimization and adaptation techniques that have been successfully applied in diverse areas, such as machine learning, combinatorial problems, VLSI design, and numerical optimization [16]. EAs provide a framework that consists of genetic algorithms (GAs) [3, 7], evolutionary programming (EP) [5, 6], and evolution strategies (ESs) [12, 15]. A comparison of these different methods can be found in [2].

Each of these evolutionary algorithms is designed along different methodologies. Despite their differences, all evolutionary algorithms are heuristic population-based search procedures that incorporate random variation and selection. Typically, such population-based search procedures generate offspring in each generation. A fitness value (defined by a fitness or objective function) is assigned to each offspring. Depending on their fitness, each population member is given a specific survival probability. Most evolutionary algorithms can be described in the following generic form:

- Step 0: Initialize and evaluate the fitness of each individual of the population
- Step 1: Select the parents according to a selection scheme (e.g., roulette wheel, linear ranking, or truncation selection)
- Step 2: Recombine and mutate selected parents with a specific operator
- Step 3: Go to Step 1

EP and ESs place particular emphasis on the behavioral link between parents and offspring. This link is realized by applying small, typically Gaussian or Cauchy-distributed perturbations to all parameters x_i that specify an object. Furthermore, the amount of perturbation is self-adapted as to maximize the expected progress of the fitness value in each generation. Therefore, EP and ESs are both predominantly designed for function optimization.

GAs have also been successfully applied to various optimization tasks [7, 16]. Recent results [13], however, have tempered that success in terms of efficacy. As an explanation, the research community has been reminded [4] that Holland's original research aimed at designing and implementing robust adaptive systems as opposed to developing sophisticated optimization techniques, even though GAs can be *used as* function optimizers [4].

The dilemma of current research is that even though the pertinent literature on EAs, e.g., [4, 10] and others, considers adaptation different from optimization, it does not provide any definition for adaptation or for the mentioned difference. Even though the literature on EAs does not provide a (working) definition, it seems common agreement that adaptation is concerned with time varying fitness landscapes as opposed to time invariant fitness functions typically used in optimization. This paper aims at formulating a *working hypothesis* for adaptation on the evolutionary time scale. To this end, Section 2 reviews some biologically inspired definitions, since adaptation has a long tradition in biology. It then proposes a working hypothesis from the point of view of technical applications.

Various practical tasks can be used to examine the adaptive behavior of different EAs. A common feature of most technical applications is that the object at hand is specified by a set of parameters $x_1, \dots, x_n$, with n denoting the number of parameters. In order to concentrate on the behavior of different EAs, several artificially designed test functions are used for experimentation, since complex, real-world tasks involve too many additional task-related problems. Furthermore, artificial test functions have the advantage that they can be fully understood and controlled. The experimental setup, including the used algorithms, test functions, and all parameter settings, is summarized in Section 3. Section 4 then presents the results of these basic experiments. Then Section 5 discusses related research [1], and finally, Section 6 provides a conclusion including a short discussion.

2 Adaptation: A Working Hypothesis

Adaptation has a long tradition in biology. In nature, each individual is forced to adapt to its environment. These adaptation forces originate from climate changes, limited food sources, behavioral changes of prey, the appearance of new predators, the competition among individuals of the same species, and so forth. According to McFarland [9], biologists usually distinguish between four different forms of adaptation: (1) evolutionary adaptation through genetic mechanisms in the very long term, (2) physiological adaptation to, for example, climate change or changes in food quality, (3) sensory adaptation to compensate changes in the strength of a particular stimulation, and (4) adaptation by learning to adapt

to a wide variety of different types of environmental changes. These four levels coincide with the explanation given in Merriam Webster’s dictionary.

Even though the explanations given above provide a good categorization, they are not specific enough for a technical definition. Holland [8, p. 3] argues *“that adaptation, whatever its context, involves a progressive modification of some structure or structures. These structures constitute the grist of the adaptive process, being largely determined by the field of study. Careful observation of successive structural modifications generally reveals a basic set of structural modifiers or operators; repeated action of these operators yields the observed modification sequences.”* and later (p. 4), he argues that *“since a given structure performs differently in different environments – the structure is more or less fit – it is the adaptive plan’s task to produce structures which perform “well” (are fit) in the environment confronting it.”* Holland’s discussion proposes a close relationship between adaptation and fitness, but does not offer a conceptional difference between adaptation and plain fitness or function optimization.

A different perspective on adaptation is provided in [19, p. 105]: *Evolutionary Theory*: *“Adaptation must be defined as the origination and development of particular morphophysiological properties, whose importance to an organism is unambiguously associated with the various general or particular environmental conditions.”* and later (p. 118), it is said that *“Adaptation is the morphophysiological manifestation of the relationships between an organism and the environment recognized through their changes.”* In this view, it can be assumed that the species already exist and that their individuals have sufficient fitness to survive. Then, adaptation concerns an organism with respect to its particular environmental conditions. Accordingly, this paper proposes the following *working hypothesis* for adaptation on the evolutionary time scale.

Definition: Assume a population p that has converged to the neighborhood of the (global) optimum. Adaptation on the evolutionary time scale then refers to a fitness function in which the location of the optimum $\mathbf{x}^o$ is changing over time, i.e., all components of $\mathbf{x}^o$ are functions of the time $x_i^o = x_i^o(t)$. A population or an individual adapts to its environment, if it successfully tracks the changing location of the optimum.

By this definition, optimization and adaptation are two different tasks. The goal of optimization is to find the optimum, whereas the goal of adaptation is to track the changing location of that optimum. The definition of adaptation does not specify *why* the location of the optimum is changing.

3 Methods

This paper focuses on evolution strategies¹ and genetic algorithms. Therefore, this section first describes these algorithms in more detail and it then discusses further details of the experimental setup.

¹ For evolutionary programming, the same results can be expected as for evolution strategies, since both schemes differ, with respect to the scope of this paper, only in the self-adaptation scheme of the step-size (mutation strength) σ .

Both $(\mu \nmid \lambda)$ -ES selection schemes maintain a population of μ parents and generate λ offspring in each generation. A (μ, λ) -ES indicates that the μ new parents are selected only from the λ offspring, whereas in a $(\mu + \lambda)$ -ES, the μ parents are selected from the union of parents and offspring. For further details see [2]. The (μ, λ) -ES is currently recommended [2] for pure function optimization.

In contrast to traditional GAs, ESs encode each real-valued parameter as a floating-point number, and they apply mutation to *all* parameters simultaneously, i.e., $p_m = 1$. Mutations are typically implemented by adding $(0, \sigma)$ -normally distributed random numbers. A key feature of ESs is that they self-adapt the mutation strength σ . In its simplest form, each individual is enhanced by a step size σ , which is global with respect to the individual's parameters and local with respect to the population members. Each offspring inherits its step size from its parent(s). This step size is typically modified by taking the product of itself with a log-normally-distributed random number prior to mutation. By this means, the step size is self-adapted: those offspring survive that have the best adapted step size. For further details of different step size schemes see [2, 15]. In more elaborate forms, ESs maintain one step-size for each parameter or even maintain a whole $n \times n$ matrix with $n(n - 1)/2$ independent parameters to generate correlated mutations. In all experiments reported in this paper, only one global step-size was used for each individual, and the step-sizes were always initialized with $\sigma = 0.001$.

As a representative for GAs, the breeder genetic algorithm (BGA) [11] was chosen for the following four reasons. First, the BGA is especially tailored to continuous function optimization. Second, the BGA yields superior results in the field of continuous function optimization [11, 13]. Third, the BGA encodes all real-valued parameters x_i as floating-point numbers, which has been shown to be more efficient than bit-coding schemes [14]. Fourth, the BGA uses standard parameter settings, such as a small mutation probability $p_m = 1/n$ and a high recombination probability $p_r = 0.5$, and uses an elitist (λ, μ) truncation selection scheme, which has been proven very efficient [17]; elitism means that the best individual obtained so far is preserved for the next generation. The BGA proposes the following mutation operator (see [11] for further details):

$$\delta = A \sum_{i=0}^{15} \alpha_i 2^{-i}, \quad \alpha_i \in \{0, 1\}, \quad (1)$$

where A specifies the mutation range, which might be reduced during evolution, and each α_i is set to the value 1 with probability $1/16$ and 0 with probability $15/16$. On average, the BGA's mutation operator picks only one non-zero α_j yielding $\delta = 2^{-j}$ for some j with $0 \leq j \leq 15$.

The main differences between ESs and GAs (including the BGA) are that GAs do not feature any self-adaptation of the mutation strength A and that GAs prefer small mutation rates in the order of $p_m = 1/n$, as typically recommended [2, 13, 16]. In essence, both algorithms differ in the mutation probability, i.e., $p_m = 1$ (ES) and $p_m = 1/n$ (BGA), and the self-adaptation/non-adaptation of the mutation strength. Other GA variants would use bit-string representations

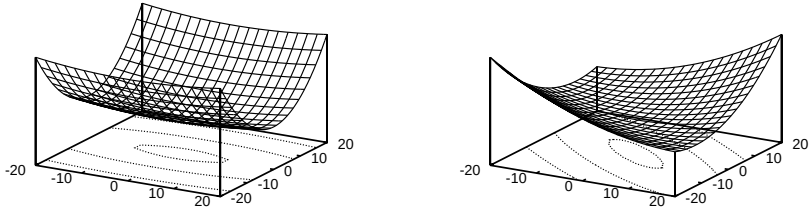


Fig. 1. 3D and contour plot of the function $f(x_1, x_2) = x_1^2 + 10x_2^2$ with (right figure) and without (left figure) a rotation of the coordinate system.

as opposed to floating-point numbers as a third difference to ESs, but bit-string representations yield worse performance under standard implementations [14].

Rather than using a complex real-world task, this paper concentrates on the following three widely-used test functions: the sphere $f_0(\mathbf{x}) = \sum_{i=1}^n x_i^2$, the quadratic function $f_1(\mathbf{x}) = \sum_{i=1}^n 2^i x_i^2$ with different eigenvalues 2^i along the coordinate axis x_i , and Rastrigin's function $f_6(\mathbf{x}) = \sum_i [x_i^2 - \cos(2\pi x_i) + 1]$. These three test functions seemed sufficient, since the task is not to discover the global optimum, but to track the location of the moving global optimum. Therefore, all individuals have been initialized around the global optimum with $(0,0.1)$ -Gaussian-distributed noise.

It has been shown [13] that the efficacy of many GA experiments originates from the decomposability property $f(\mathbf{x}) = \sum_{i=1}^n f_i(x_i)$ of most of the widely-used test functions, which are only $O(n)$ hard, since all parameters x_i can be optimized independently of all others. For these test functions, GAs using floating point representations scale with $O(n \ln n)$ and GAs using bit coding representations scale with $O(n^k \ln n)$ with k denoting the number of bits per parameter x_i [13, 14]. Furthermore, it has been shown [13] that the performance of GAs progressively degrades, if a rotation of the coordinate system is applied. As can be seen from Figure 1, a rotation does not change the function's structure but its orientation in the search space. Therefore, this paper compares the adaptation performance on the original test functions $f(\mathbf{x})$ and the rotated versions $f(\mathbf{M}\mathbf{x})$, where $\mathbf{M}$ denotes a pure rotation matrix.

In each generation, the location of the optimum, which is initially at $\mathbf{x}^o = \mathbf{0}$, is relocated by adding a speed quantity to each component $x_i^o \leftarrow x_i^o + s$ resulting in a total amount of change of $s\sqrt{n}$. The same amount s is used for all components x_i for the following reasons. First, changing some components not at all or just slightly, would mean that adaptation takes place only in a subspace. Second, changing only a few variables, would favor GAs too much, since small mutation rates $p_m = 1/n$ induce a strong search bias along the coordinate axes (see also [13]). Third, choosing a moving direction different from the $(1, 1, \dots, 1)^T$ would not lead to different results for the following two reasons. First, in the experiments not only the original test functions but also their rotated version have been used. Since the rotation angles have been uniformly chosen at random, the effective moving direction is arbitrary in the n -dimensional search space

with different effective moving speeds s_i along each coordinate axis of the rotated coordinate system. And as is discussed in Section 4, the rotation does not significantly effect the obtained performance. Second, quadratic functions with different Eigenvalues λ_i along each coordinate axis forces the population center toward a line $\lambda_1 x_1 = \lambda_2 x_2 = \dots \lambda_n x_n = c$ due to fitness selection (and not distance selection). Therefore, such functions are a challenge for ESs that use only one global step size σ .

It should be noted that the concept of tracking a moving optimum has an important consequence for the design of the GAs' mutation operator. Since GAs do not feature a self-adaptation of the mutation strength denoted by A in the BGA, the designer has to set the mutation strength to an appropriate value. In the experiments reported in Section 4, the mutation strength was set to $A = 100s$ in order to ensure small as well as large enough mutations.

In the experiments presented in the next section, the speed quantity s was varied between 0.001-10. It should also be mentioned that due to the relocation of the optimum in each generation, all individuals including the parents are re-evaluated. This unusual procedure is necessary, since it does not make sense to preserve an evaluated fitness value from one generation to another; the fitness function is changing as opposed to standard optimization tasks.

Current research on evolution strategies [2] recommends a (μ, λ) selection scheme for pure function optimization with a static fitness functions. The main advantage of choosing the new parents only from the offspring (non-elitism) is that it increases the chance of escaping from narrow, local optima due to noise or erroneous fitness evaluation. In this paper, however, a $(\mu + \lambda)$ -selection scheme has been chosen for the following reasons. First, it features elitism and therefore, makes a comparison with GAs easier, which also feature elitist selection schemes. Second, since all population members including all parents are evaluated in each generation, erroneous fitness evaluation is not a problem. Furthermore, due to the moving optimum, each population member will eventually be removed, since its fitness value starts steadily decreasing at a certain generation. In additional control experiments (see also Section 4) on the test functions described above, both schemes did not yield any significant difference.

4 Results

This section presents a summary of the most important results. All performance Figures 2 to 4 do not show any fitness values but the Euclidean distance of the population center to the current location of the moving optimum $\mathbf{x}^o(t)$. A lower limit of this performance measure would be zero, which could be obtained only if a population would know to which location the optimum is moving. In practice, this performance measure is greater than zero and depends on the moving speed s as well as the number of dimensions n . As should be obvious, increasing the number of dimensions n increases the difficulty of both the optimization and the adaptation task if the population size is kept constant as is done in this paper.

(20+40)-ES

(20+40)-GA

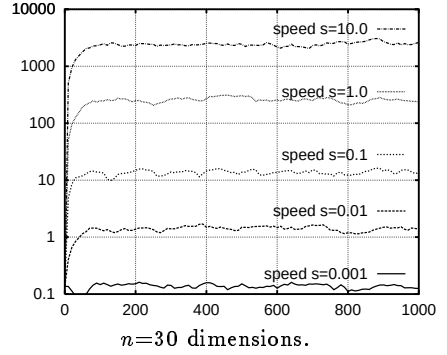
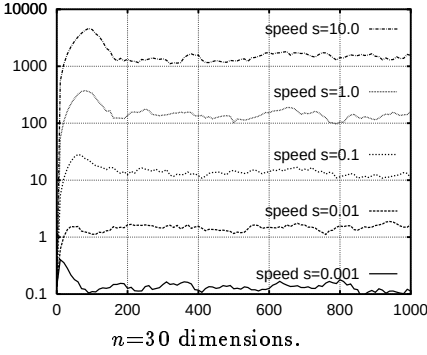
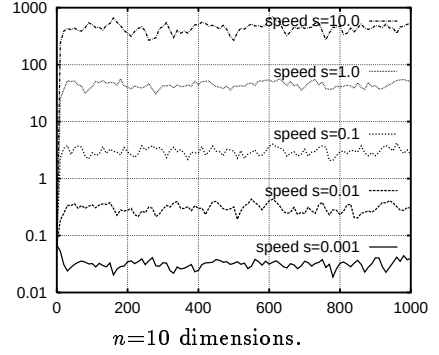
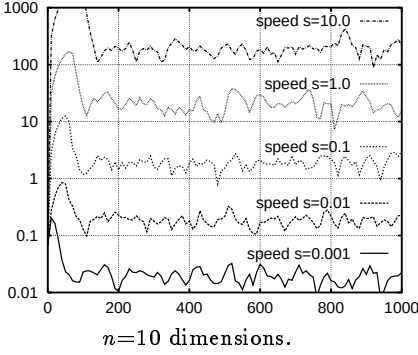


Fig. 2. Comparison of a (20+40)-ES (left column) and a (20+40)-GA (right column) when applied to the sphere $f_0(\mathbf{x}) = \sum x_i^2$ with $n = 10$ (upper row) and $n = 30$ (lower row) dimensions.

Each of the Figures 2 to 4 compares a (20+40)-ES (left column) with a (20+40)-GA (right column) when applied to functions f_0 , f_1 , and f_6 with dimensions varying between 3 and 30. From Figure 2, it can be seen that both procedures exhibit almost identical performance on f_0 . For function f_0 , no coordinate rotation has been applied, since f_0 is already rotationally invariant. It should be noted that in 30 dimensions, for example, the optimum travels a total distance of $s\sqrt{30}$ per generation with s denoting the speed constant.

Figure 3 compares both algorithms when applied to the quadratic function $f_1(\mathbf{x}) = \sum_{i=1}^n 2^i x_i^2$ with a rotation of the coordinate system. It can be seen that in this case, the ES adapts approximately half an order of magnitude closer to the moving optimum than the GA does. Compared to the rotated case, a (20+40)-GA exhibits slightly better performance when the rotation is not applied (not shown due to space limitations). From the last row of Figure 3, it can be seen that none of both algorithms is able to sufficiently track the optimum

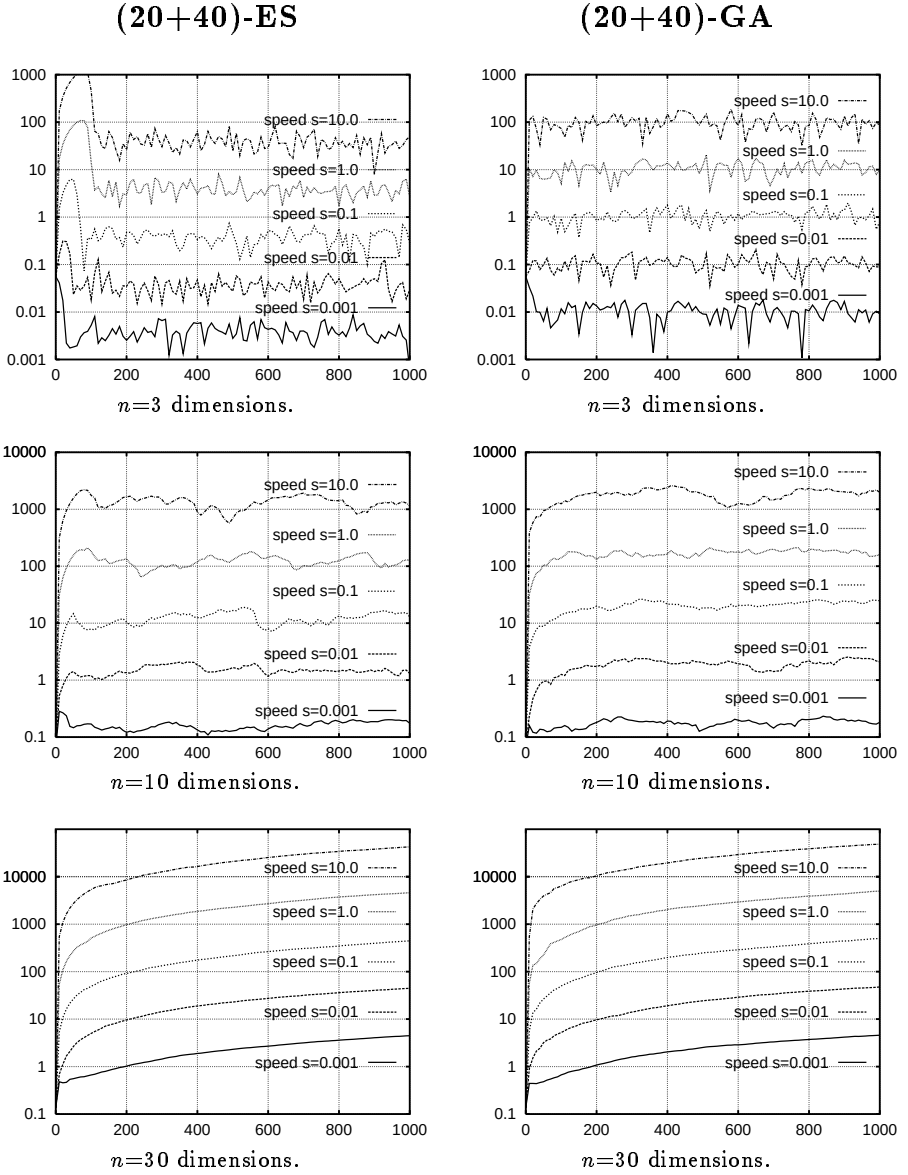


Fig. 3. Comparison of a (20+40)-ES (left column) and a (20+40)-GA (right column) when applied to the quadratic function $f_1(\mathbf{x}) = \sum_{i=1}^n 2^i x_i^2$ with $n = 3$ (upper row), $n = 10$ (middle row), and $n = 30$ (lower row) dimensions.

in 30 dimensions. It might be that due to the different eigenvalues 2^i along each parameter x_i , the population size is not sufficient for that many dimensions.

Figure 4 compares the tracking performance of both algorithms when ap-

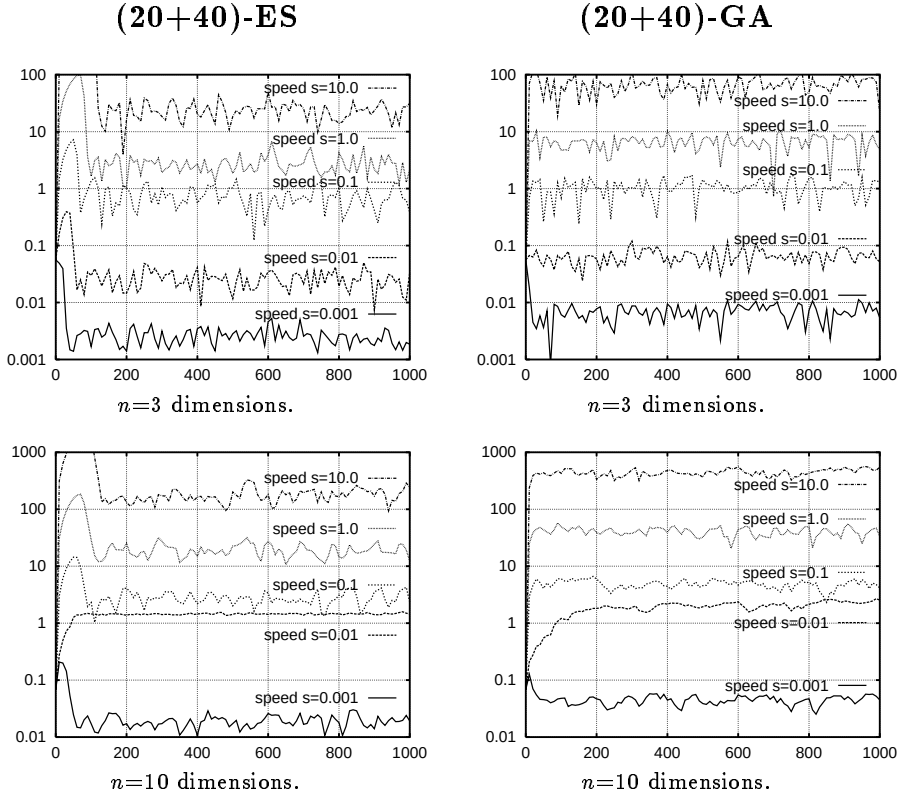


Fig. 4. Comparison of a (20+40)-ES (left column) and a (20+40)-GA (right column) when applied to Rastrigin's function $f_6(\mathbf{x}) = \sum_i [x_i^2 - \cos(2\pi x_i) + 1]$ with $n = 3$ (upper row) and $n = 10$ (lower row) dimensions.

plied to Rastrigin's function $f_6(\mathbf{x}) = \sum_i [x_i^2 - \cos(2\pi x_i) + 1]$. Again, it can be seen that the ES performs slightly better than the GA. The problem with this function is that the adaptation task turns into an optimization task for high speed values. If the moving speed is too high, the global optimum is moving far away from *all* population members from one generation to the next. Then, no population member is in the local basin of attraction of the global optimum, and the offspring have to find the new location of this basin of attraction.

Figures 2 to 4 show a peculiar behavior of the ES in the first few generation. In this time, the ES self-adapts the mutation strength σ to an optimal value that is in the order of the optimum's moving speed. This time span increases logarithmically with the speed due to the multiplicative correction of the step-size.

Figure 5 shows a comparison of a (μ, λ') and a $(\mu + \lambda)$ selection scheme when applied to the sphere with $n = 10$ dimensions and speed $s = 0.1$. In order to consider the same number of function evaluations in each generation, the number

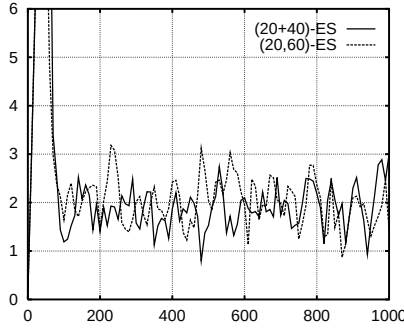


Fig. 5. Comparison of a (20+40)-ES and a (20,60)-ES when applied to the sphere with $n = 10$ dimensions and speed $s = 0.1$.

of offspring λ' of the (μ, λ') has to be set to $\lambda' = \mu + \lambda$. As can be clearly seen from this figure, both schemes yield the same statistical results.

Additional experiments have investigated the influence of recombination to the tracking behavior of both algorithms. In these experiments, both algorithms consistently yielded worse performance *with* recombination than without recombination. This result contradicts the common knowledge that recombination would be beneficial. In case of a moving optima, however, recombination cannot be beneficially exploited, since the combination of existing “good” solutions will not help in subsequent generations where the optimum is moving further.

5 Related Research

Similar research was independently done and has already been presented in [1]. Even though both papers ([1] and this paper) aim at a similar goal, the following three significant differences can be identified. First, the research presented in [1] aims at investigating the behavior of different adaptation schemes for the mutation strength in evolutionary programming. Accordingly, no other evolutionary algorithms were considered in [1]. By contrast, this paper focuses on investigating the behavior of different evolutionary algorithms, evolution strategies and genetic algorithms, with respect to their different forms of applying mutations, i.e., $p_m = 1$ and using a self-adaptation scheme for the mutation strength σ versus $p_m = 1/n$ without using any adaptation scheme for the mutation strength. Second, in [1] only the quadratic function with three parameters $x_{1 \leq i \leq 3}$ was investigated; no other test functions and no other search dimensions were considered. This paper considers three different test functions, the sphere, the quadratic function, and Rastrigin’s function, with many different properties, such as local optima and different Eigenvalues. Third, [1] has used quite different ways of moving the optimum. In his paper, the optimum was moved on a straight line, like in this paper, as well as in a circular and a chaotic way. The last two options have not been considered here, but as already discussed in Section 3,

they are covert to a large extent by considering a large variety of test functions. In addition, the results presented in [1] do not suggest a significant influence of the moving path to the resulting performance. Furthermore, has used three different update forms of the optimum's location; updating was done after 1, 5 or, 10 generations. As has been shown in [1], updating the optimum's location after 5 or 10 generations leads to a rough performance, especially for the investigated self-adaptation scheme. In [1], the self-adaptation scheme exhibits more problems, if in a circular trajectory, the update occurs after 5 or 10 generations, since the mutation strength is self-adapted for each particular parameter x_i ; these n mutation strengths have to be updated very frequently resulting in a "bumping" behavior. The update of a changing optimum after some generations has also been investigated in [18]. However, there the location of the optimum has been moved only after the entire population has been converged to the old optimum. Such task are, in the sense of this paper, rather an optimization task.

The results presented in [1] coincide with the results presented in this paper. As has been shown in the previous section, the self-adaptive mutation scheme of the ESs requires some initial time to adjust the mutation strength to the appropriate value. With this respect, GAs yield smoother results, since the appropriate mutation strength is given by the designer or by a designer-chosen heuristic. In conclusion, the research presented in [1] is complementary to the research presented in this paper.

6 Conclusions

This paper has proposed that adaptation should refer to tracking a moving optimum in dynamically changing fitness functions as opposed to function optimization as an optimum-finding task. The experiments presented in this paper have compared the tracking performance of the evolution strategies and genetic algorithms. It can be summarized that the ES performs slightly better than the GA and that in *sharp contrast* to pure function optimization (see also [13]), the application of a coordinate rotation has almost no effect on the tracking performance of both algorithms. Future research will be devoted to the investigation of different population sizes and different Eigenvalues for each coordinate axis.

From the practical point of view, ESs have the great advantage of self-adapting the mutation strength, which takes place whenever it is required. When using a GA by contrast, the designer has to find a reasonable estimate of the moving speed in order to determine the mutation strength, which is denoted by A in the BGA. This problem is not that severe, since the BGA's mutation operator yields robust behavior within a few orders of magnitude (see also [11]).

As a second, more general result, it should be noted that the particular implementation and parameter setting has much less influence on the tracking behavior than it would have on the optimization behavior. It seems essential that an algorithm provides a sufficient amount of variation and that it applies a suitable selection scheme.

References

1. Angeline, P., 1997, Tracking Extrema in Dynamic Environments, in: *Proceedings of the Sixth Annual Conference on Evolutionary Programming VI*, P.J. Angeline, R.G. Reynolds, J.R. McDonnell, and R. Eberhart (eds.) (Springer-Verlag), pp. 335-345.
2. Bäck, T., Schwefel, H.-P., 1993, An Overview of Evolutionary Algorithms for Parameter Optimization, *Evolutionary Computation* 1(1):1-23, 1993.
3. De Jong, K.A., 1975, An Analysis of the Behavior of a Class of Genetic Adaptive Systems, Ph.D. Thesis (University of Michigan).
4. De Jong, K.A., 1993, Genetic Algorithms are NOT Function Optimizers, in: *Foundations of Genetic Algorithms 2*, L.D. Whitley (ed.) (Morgan Kaufmann Publishers), pp. 5-17.
5. Fogel, D.B., 1995, *Evolutionary Computation: Toward a New Philosophy of Machine Learning Intelligence*, (IEEE Press).
6. Fogel, L.J., 1962, "Autonomous Automata", *Industrial Research*, 4:14-19.
7. Goldberg, D.E., 1989, *Genetic Algorithms in Search, Optimization and Machine Learning*, (Addison-Wesley).
8. Holland, J., 1992, *Adaptation in Natural and Artificial Systems: an Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, (MIT Press).
9. McFarland, D., 1990, What it means for Robot Behaviour to be Adaptive, in: *From Animals to Animats, Proceedings of the First International Conference on Simulation of Adaptive Behavior*, J.-A. Meyer and S.W. Wilson (eds.) (MIT Press), pp. 22-28.
10. Menczer, F., Belew, R.K., Willuhn, W., 1995, Artificial life applied to adaptive information agents, in: *AAAI Spring Symposium Series: Information gathering from heterogeneous, distributed environments*. C. Knowblock, A. Levy, S.-S. Chen, and G. Wiederhold (eds.) (AAAI).
11. Mühlenbein, H., Schlierkamp-Voosen, D., 1993, Predictive Models for the Breeder Genetic Algorithm I. *Evolutionary Computation* 1(1):25-50.
12. Rechenberg, I., 1973, *Evolutionsstrategie*. (Frommann-Holzboog).
13. Salomon, R., 1996, Reevaluating Genetic Algorithm Performance under Coordinate Rotation of Benchmark Functions, *BioSystems* 39(3):263-278.
14. Salomon, R., 1996. The Influence of Different Coding Schemes on the Computational Complexity of Genetic Algorithms in Function Optimization, in: *Proceedings of The Fourth International Conference on Parallel Problem Solving from Nature (PPSN IV)*, H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel (eds.) (Springer-Verlag), pp. 227-235.
15. Schwefel, H.P., 1995, *Evolution and Optimum Seeking* (John Wiley and Sons).
16. Srinivas, M., Patnaik, L., 1994, Genetic Algorithms: A Survey, *IEEE Computer*, pp. 17-26.
17. Thierens, D., Goldberg, D., 1994, Convergence Models of Genetic Algorithm Selection Schemes, in: *Proceedings of Parallel Problem Solving from Nature 3*, Y. Davidor, H.P. Schwefel, and R. Männer (eds.) (Springer-Verlag), pp. 119-129.
18. Vavak, F., Fogarty T.C., 1996, A Comparative Study of Steady State and Generational Genetic Algorithms, in: *Evolutionary Computing: AISB Workshop*, LNCS 1143, T.C. Fogarty (ed.) (Springer Verlag), pp. 297-304.
19. Volkenstein, M.V., 1994, *Physical Approaches to Biological Evolution* (Springer-Verlag).

The Frequency Assignment Problem: A Look at the Performance of Evolutionary Search

Christine Crisan* and Heinz Mühlenbein**

GMD - German National Research Center for Information Technology
D - 53754 Sankt Augustin
Germany

Abstract. The performance of evolutionary search on the Frequency Assignment Problem is the topic of ongoing research. In this paper the performed evaluation is done in two steps. First, the fitness landscape to be optimized is analyzed. Secondly, the performance of different search procedures is described. The insight gained from the fitness landscape analysis is related to the performance of the different search procedures.

1 Introduction

With the perennial increase in the demand of mobile telephone services and the restricted frequency spectrum dedicated to these services, the problem of optimal usage of frequencies has become a major step in planning and operating a digital *mobile telecommunication network (MTN)*. A *MTN* divides the area to be served into cells. A cell is defined to be the geographical area around a *base station (BS)*, where the field strength received from the antenna of the corresponding *BS* is high enough to guarantee a good communication. Each base station has to be assigned a number of frequencies on which the wireless communication occurs [1]. The number of frequencies to be assigned to each base station is approximately defined by the number of users to be served in that area. This leads to an inhomogeneous distribution of frequency requirements. The assignment of frequencies to base stations must satisfy several electromagnetic compatibility constraints such as the *co-channel*, *adjacent-channel* and *co-site* constraint. The first two constraints refer to identical or, regarding the frequency spectrum, adjacent frequencies which may be used only by base stations which cannot interfere with one another. The *co-site* constraint implies that frequencies used at a certain base station must be at a predefined minimum distance with regard to the frequency spectrum.

The union of the frequency requirements from each cell can be viewed as a joint set of frequency requirements. Depending on which cell a requirement belongs to, the assignment of a frequency for that requirement has to satisfy different constraints. The *Frequency Assignment Problem (FAP)* as described in this paper consists of assigning a frequency to each requirement subject to

* email: Christine.Crisan@gmd.de

** email: Heinz.Muehlenbein@gmd.de

a set of constraints. Simultaneously the total number of different frequencies used by an assignment has to be minimized. The *FAP* can be formulated as a discrete optimization problem. It is a generalization of the graph coloring problem, hence NP-hard. Note that the two demands, finding frequency assignments which satisfy the electromagnetic constraints on the one hand and of frequency assignments which make use of as few as possible frequencies on the other hand, are orthogonal, which makes the problem even harder.

Modern digital cellular networks consist of more than 4000 base stations with a requirement of at least two frequencies per base station, which results in optimization problems of dimensions greater than 8000. Therefore, efficient optimization procedures have to be developed in order to find high quality frequency assignments. Several heuristic optimization methods for different instances of the *FAP* have already been proposed [2, 3, 4].

In this paper we take a bottom up approach to the problem in order to favor the design of efficient search procedures. This means that we take a look at the search space and then look at the performance of several search operators within an evolutionary search environment. In this paper we restrict the analysis of the search space to the consideration of the density of states, which is a measure of the hardness of an optimization problem [6] and take a look at the distribution of locally optimal solutions.

The paper is structured as follows: first we give a brief description of the *FAP*, then we analyze the search space. Finally we present several search operators and relate the obtained results to the knowledge gained about the search space.

2 Frequency Assignment Problems

There are many ways to define different instances of the *FAP* [5]. The special case of the *FAP* described in this paper can be described as follows.

Given are a set of requesters $R = \{r_i\}_{i=1,\dots,n}$ and a set $F = \{f_i\}_{i=1,\dots,m}$ of frequencies. For each requester r_i , $i = 1, \dots, n$ a set $D_i \subseteq F$ of frequencies which may be assigned to the respective request is given. For each pair of requesters (r_i, r_j) there is a minimum required distance given. This defines the minimal difference between frequencies which will be assigned to the requesters r_i and r_j . For each pair of requesters (r_{2k-1}, r_{2k}) , $k = 1, \dots, n/2$ which we will refer to as *parallel links*, this distance has to be exactly $d = 238$. This is the so called *equality constraint*.

Required is an assignment of frequencies to the requesters so that each requester is assigned a valid frequency and no interference constraint, i.e. minimal distance, is violated. Furthermore, the total number of different frequencies used in an assignment should be minimal. The two requirements, minimizing the interference as well as the number of frequencies used are orthogonal, which makes the problem even harder.

The *FAP* can also be described with help of an undirected weighted graph $G = (V, E, w)$, with V a set of vertices, $E \subseteq V \times V$ a set of edges and $w : E \longrightarrow \mathbb{R}$

Table 1. Description of the considered problem instances

Instance	n	# Constr.	ED	Min/Max
SCEN01	916	5548	0.0132	16/48
SCEN02	200	1235	0.0620	14/44
SCEN03	400	2760	0.0345	14/48
GRAPH01	200	1135	0.0570	18/44
GRAPH02	400	2245	0.0281	14/44
GRAPH03	200	1086	0.0545	?/44

a weighting function. For each requester r_i there is a vertex $v_i \in V$ and for each constraint (r_i, r_j) there is an edge $(v_i, v_j) \in E$. The function w assigns to each edge the minimum distance imposed by the corresponding constraint.

The considered instances are presented in Table 1. For each problem the dimension n , the total number of constraints (# Constr.), the edge density (ED) of the corresponding graph and the number of frequencies used in the best known solutions (Min) versus the total number of available frequencies (Max) are given. For the problem instance GRAPH03 the optimum is still unknown.

The *FAP* instances SCEN01, SCEN02, and SCEN03 are real-life instances which were provided by the Centre d'Electronique de l'Armement, France within the CALMA project on *Combinatorial Algorithms for Military Applications*. The instances GRAPH01, GRAPH02 and GRAPH03 were made available by the research group at the Delft University of Technology which was involved in the CALMA project. Although they were randomly generated, they have the same characteristics as the real-life instances. For a more detailed description of these instances see [9, 12].

3 Search Space Analysis

There have been many attempts to find measures which characterize a fitness landscape and give an indication about the difficulty of the problem [6, 7, 8].

In this paper we investigate two measures, the density of states and the distribution of locally optimal solutions.

First we describe the coding used, define a cost function and a neighborhood structure on the search space and show the induced metric.

3.1 The Search Space

Coding We are coding the *FAP* as an n -dimensional integer vector. A value f_i of the i th component means that frequency f_i is assigned to the request r_i , $i = 1, \dots, n$. Since for all considered instances the constraint $|f_{2k-1} - f_{2k}| = 238$, $k = 1, \dots, n/2$, has to hold, we define the domain of definition to consist only of solutions which already fulfill this requirement. The range of valid frequencies is the same for each parallel link. Since for each link there are exactly $2d_{2k}$

different assignments, where d_{2k} is the number of different possible frequencies for the requests $2k - 1$ and $2k$, the dimension of the resulting search space is $2^{n/2} \prod_{i=1}^{n/2} d_{2i}$. This coding, for which solutions violating the equality constraint are not considered, reduces the search space drastically and as we have found out leads to a faster convergence. The dimension of the search space which also contains solutions which do not satisfy the equality constraint is $\prod_{i=1}^{n/2} d_{2i}^2$.

Cost Function For a solution, i.e. frequency assignment, s , the fitness function is defined as follows

$$\mathcal{F}(s) = w_1 \cdot NOV(s) + w_2 \cdot NOF(s) + penalty(s) \quad (1)$$

with $NOV(s)$ and $NOF(s)$ the number of violated constraints and the number of different frequencies used by the solution. The penalty function favors solutions which assign some of the frequencies seldom. It is defined as follows

$$penalty(s) = \sum_{i=1}^m \sqrt{UF(f_i)} \quad (2)$$

with $UF(f_i)$ the number of times the frequency f_i is used within the solution.

Neighborhood Next we define a neighborhood structure on the search space induced by the coding. We define two solutions $s = (f_1, \dots, f_n)$ and $s' = (f'_1, \dots, f'_n)$ to be neighboring to each other *iff* they differ in exactly one parallel link assignment, i.e.

$$\exists! k \in (1, n/2) : f_{2k-1} \neq f'_{2k-1} \wedge f_{2k} \neq f'_{2k} \quad (3)$$

$$\forall j \neq k \quad f_j = f'_j \quad (4)$$

We define the neighborhood of a solution s as the set of all its neighboring solutions s' .

Metrics on the search space The coding used and the defined neighborhood induce a metric defined by the normalized Hamming distance between two solutions. The normalized Hamming distance between two solutions $s = (f_1, \dots, f_n)$ and $s' = (f'_1, \dots, f'_n)$ is defined as the total number of different assignments divided by the maximum number of different possible assignments.

$$h_d(s, s') := \frac{1}{n} \sum_{i=1}^n sgn(|f_i - f'_i|) \quad (5)$$

where sgn is the signum function. Notice that the normalized Hamming distance is always even, since only solutions which fulfill the equality constraint are considered.

3.2 Density of states

The density of states as introduced in [6, 7] is defined by the number of different solutions with a certain fitness value. Knowledge about the density of states helps with the estimation of the fitness value for the problem optimum and gives a measure of the difficulty for finding the global optimum. The capability of estimating the fitness value of the optimum with the help of the density of states concept arises from the fact that an approximation of the lowest fitness value (in case of a minimization) is given by the states for which the density is zero. Within the optimization process, the probability of still finding a better solution decreases at the same rate as the values of the density of states decrease. A rapid decrease of the density of states values points to a difficult optimization problem.

The density of states has to be estimated since the extent of the search space makes an exact computation impossible. For obtaining an approximation of the density of states we used a Metropolis algorithm. Starting with a random solution s , a new solution s' from its neighborhood is obtained, i.e. the assignment of exactly one parallel link is changed. If the fitness value of the new solution is better, s' is the new solution, otherwise an inferior solution is accepted with probability $e^{-\frac{\Delta\mathcal{F}}{T}}$, where $\Delta\mathcal{F}$ is the change in the fitness value. In this manner the algorithm converges to a Boltzmann distribution from which the density of states can be approximated [7]. So, in equilibrium the density of states observes the Boltzmann distribution

$$P_{eq}(\mathcal{F}) \sim n(\mathcal{F}) \cdot e^{-\frac{\mathcal{F}}{T}} \quad (6)$$

where $P_{eq}(\mathcal{F})$ is the equilibrium density, $\mathcal{F}$ is the energy function, T the temperature and $n(\mathcal{F})$ the density of states with the energy $\mathcal{F}$ [7].

Figure 1 shows the estimated values for the density of states for the instances SCEN01 and SCEN02. For SCEN02 there are a lot of locally optimal solutions near the global optimum. In the vicinity of the optimum the density of states decreases. The Metropolis algorithm detected configurations with a fitness value as low as 16. It estimates that there are about 1800 configurations with this fitness value. The fitness value of the global optimum is 14. For SCEN01 the distribution is different. The smallest fitness value found by the Metropolis algorithm was 22. It is estimated that there are about 2000 configurations with this fitness value. The global optimum has the fitness value 16. This shows that SCEN01 is more difficult to solve, but both problem instances are rather easy optimization problems.

3.3 Hamming Distance

Next we studied the similarity between locally optimal solutions. On the basis of the previously defined normalized Hamming distance we compute the normalized Hamming distance of a solution s with respect to other M considered solutions

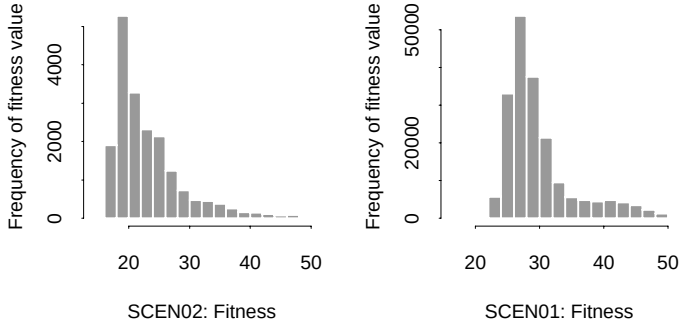


Fig. 1. Density of states

$s_1, \dots, s_M$ [11] as follows

$$h_d(s) = \frac{1}{M} \sum_{i=1}^M h_d(s, s_i). \quad (7)$$

Figure 2 shows the normalized Hamming distance of locally optimal solutions for the problems SCEN01 and SCEN02. For SCEN02 we computed 1000 global optimal solutions. For SCEN01 we computed 100 locally optimal solutions. The values of the Hamming distance are almost maximal, i.e. the solutions are very different. Even if we restrict the analysis to the best 20% of the locally optimal solutions, there is not much similarity. In fact, for SCEN01 the normalized Hamming distance of the best local optimum to the 20% best locally optimal solutions is about 0.87.

This result is very different from a corresponding search space analysis for the Traveling Salesman Problem [11]. There the normalized Hamming distance was measured as about 0.25. This means that on average two good solutions have 75% elements in common.

The analysis of the search space shows that different instances of the *FAP* can be very different with respect to the density of states. The Hamming distance space measure shows that local optima are randomly distributed in the search space. In this case the solutions do not have building blocks, so that recombination can be an ineffective search operator. This result has been confirmed by simulations. So we decided to look first at good mutation operators.

4 Comparison of Search Operators

We used an iterated (1+1) evolutionary search algorithm in order to analyze different search operators. The algorithm is shown in Figure 3. The algorithm uses just one parent, which creates an offspring. If the offspring is better than the parent, it replaces the parent. If the algorithm gets stuck in a local optimum,

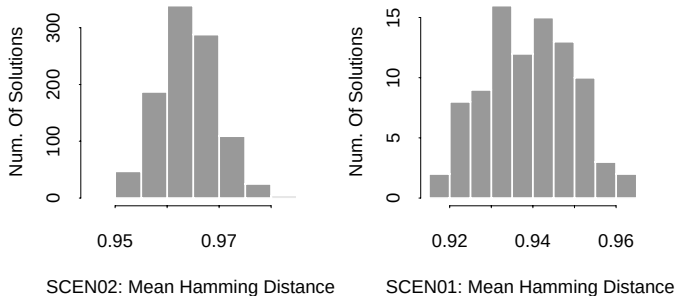


Fig. 2. Hamming Distance

it starts the search from a slightly modified point (**modify**(s)). The evaluation function is the one defined in Equation 1 with $w_1 = w_2 = 1$.

4.1 Operators

Different operators were implemented. **Random Mutation** (RM), which randomly chooses a variable for mutation and also randomly changes its assignment to a new frequency from the domain of valid frequencies. **Interference Mutation** (IM), which randomly chooses a variable for mutation, whose assignment violates at least one constraint and assigns a new frequency to it. Furthermore an **Extended Interference Mutation** (EIM) was implemented. EIM builds up a candidate list of variables which, by changing their assignment, might lead to a better solution. One variable from the candidate list is randomly chosen for mutation and assigned a new frequency. The candidate list contains the variables which violate at least one constraint and all other variables which are involved with the first variables in a constraint. Finally we developed an **Extended Interference and Frequency Mutation** (EIFM) which is similar to EIM, but the new assigned frequency is chosen among the frequencies used most often in the solution. For all defined operators, the following holds. After changing the assignment of a variable r_i , the assignment of the variable belonging to the corresponding parallel link, which is r_{i+1} or r_{i-1} , is also changed. This means that the equality constraint always holds.

All these operators perform only minimal changes on a solution, transforming it into a new solution from its neighborhood. Let $\mathcal{N}_{RM}$, $\mathcal{N}_{IM}$, $\mathcal{N}_{EIM}$ and $\mathcal{N}_{EIFM}$ be the neighborhood functions of the defined operators and $\mathcal{N}_{RM}(s)$, $\mathcal{N}_{IM}(s)$, $\mathcal{N}_{EIM}(s)$ and $\mathcal{N}_{EIFM}(s)$ their corresponding neighborhoods for a given solution

```

begin
  ITER = 0;
   $s \leftarrow \text{generate\_initial\_solution}$ 
   $\text{fit}_s \leftarrow \text{evaluate}(s)$ 
  iterate while (ITER < MAX_ITER)
    ITER++;  $\text{num\_iterations} \leftarrow 0$ ;
    modify( $s$ )
    evolutionary_search
      if ( $\text{num\_iterations} > \text{maximum\_iterations}$ )
        goto iterate
       $\text{num\_iterations}++$ 
       $s' \leftarrow \text{mutate}(s)$ 
       $\text{fit}_{s'} \leftarrow \text{evaluate}(s')$ 
      if ( $\text{fit}_{s'} < \text{fit}_s$ )
         $s \leftarrow s'$ 
      if ( $\text{terminate\_condition}$ )
         $\text{best\_solution} \leftarrow s$ 
        goto end
      else
        goto evolutionary_search
      end_evolutionary_search
    end

```

Fig. 3. Iterated Evolutionary Search Algorithm

s . Then

$$\mathcal{N}_{IM}(s) \subseteq \mathcal{N}_{EIM}(s) \subseteq \mathcal{N}_{EIFM}(s) \subseteq \mathcal{N}_{RM}(s) \quad (8)$$

All the neighborhood functions are connected, i.e. from any starting solution s an optimal solution can be reached by applying the corresponding neighborhood function. Thus, the operators may be compared with one another.

4.2 Results

Table 2 shows the obtained results. For each operator 30 runs were performed. *FE* gives the mean number of function evaluation and *Best* gives the number of utilized frequencies and the amount of interference in the best found solution. The fact that by using just RM, fairly good solutions could be obtained shows once again that a large number of locally optimal solutions exist. Nevertheless, very good solutions which are contained in the area of the search space just after the curve of the density of states decreases can not be obtained with the RM

Table 2. Obtained results

Instance	RM		EIM		EIFM		RM+1PC	
	<i>FE</i>	<i>Best</i>	<i>FE</i>	<i>Best</i>	<i>FE</i>	<i>Best</i>	<i>FE</i>	<i>Best</i>
SCEN01	1.000E+06	18/6	1.000E+06	20/1	1.000E+06	16/2	1.000E+06	18/4
SCEN02	5.935E+04	14/0	5.289E+04	14/0	1.187E+04	14/0	3.538E+05	14/0
SCEN03	1.000E+06	16/1	1.000E+06	16/0	9.866E+05	14/0	1.000E+06	16/1
GRAPH01	4.674E+05	18/0	8.574E+04	18/0	4.021E+05	18/0	1.000E+06	22/0
GRAPH02	1.000E+06	16/0	1.000E+06	14/1	9.901E+05	14/0	1.000E+06	16/0
GRAPH03	1.000E+06	30/33	1.000E+06	30/37	1.000E+06	28/38	1.000E+06	28/39

operator. IM was more efficient, but it also fails to find the optimum. EIM finds the optimum just for the problem SCEN02, which seems to be the easiest of all the problems. The best results by far can be obtained by using the EIFM operator. The operator just tunnels its way to the attractor of a good optimum by simultaneously minimizing the number of violated constraints and reducing the number of different utilized frequencies by a solution.

We also conducted experiments with a larger population size, using RM and IM combined with one of the well known crossover operators: 1 Point, 2 Point and Uniform Crossover and an elitist selection procedure. The experiments were conducted in order to gain first insights in the power of crossover for the *FAP*. Note that for the runs where Crossover was used, much more function evaluations were necessary.

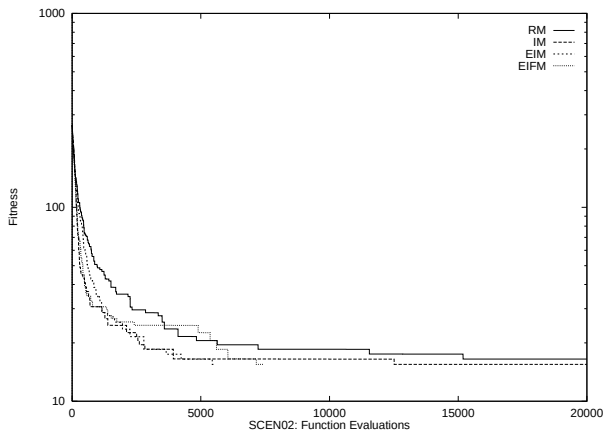


Fig. 4. SCEN02: Fitness Progress for different Mutation operators

Figures 4 and 5 show the typical progress of the fitness (i.e. cost) of the best

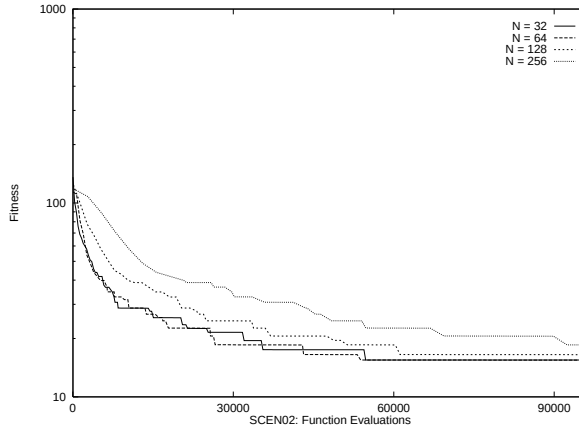


Fig. 5. SCEN02: Fitness progress for RM and 1Point Crossover

solution within a run for SCEN02. In Figure 4 different mutation operators are shown, while in Figure 5 a combination of mutation and crossover for different population sizes N is shown.

The usage of crossover makes the optimization more robust as it explores the search space more thoroughly. On its way to the optimum, local optima of the form (NOF, NOV) (see Equation 1) with $NOV = 0$ and decreasing NOF are reached. The fitness value decreases more slowly compared to the decrease when using just mutation.

The results obtained with the iterated $(1+1)$ algorithm with the EIFM operator are superior to those reported in [10], where results obtained with a standard and a meta genetic algorithms are presented. In [9] results obtained with Tabu Search are reported. There the authors only give the time needed in order to find a solution, rather than the number of function evaluations. Nevertheless, EIFM seems to find the solution more rapidly than Tabu Search, which suggests that the memory used by the Tabu Search is not necessary for these instances of the Frequency Assignment Problem.

5 Conclusions and Future Work

Evolutionary Search proves to be a powerful optimization method for solving the *FAP*. Slightly modifying a solution trapped in a local optimum and restarting the search from that point makes the search more robust. In the parts of the search space where many locally optimal solutions exist crossover shows to be a useful operator but for the *FAP* instances investigated in this paper an algorithm using only a problem specific mutation performed best.

The problem instances defined in [12] proved to be easy for our algorithm. We are now working on real world problems defined by the German mobile

telecommunication provider TMobil. These *FAP* problems have a more application oriented fitness function and the size of the problems is much larger. For Germany up to 6000 base stations have to be considered, which results in optimization problems of dimensions greater than 12000. Due to a non disclosure agreement we are currently not allowed to publish any results pertaining these data.

References

1. Balston, D.M., Macario, R. C. V. (Eds): Cellular Radio Systems. Artech House (1993)
2. Crompton, W., Hurley, S., Stephens, N.M.: A Parallel Genetic Algorithm for Frequency Assignment Problems. Proc. of IMACS SPRANN. (1994) 81–84.
3. Dorne, R., Hao, J.-K.: An Evolutionary Approach for Frequency Assignment in Cellular Radio Networks. Proc. of IEEE Intl. Conf. on Evolutionary Computation. (1995) 539–544.
4. Hao, J.-K., Dorne, R.: Study of Genetic Search for the Frequency Assignment Problem. Artificial Evolution, LNCS 1063. (1996) 333–344.
5. Hale, W.H.: Frequency Assignment: Theory and Application. Proc. of the IEEE, Vol. 68, No. 12 (1980) 1498–1573.
6. Rose, H., Ebeling, W., Asselmeyer, T.: The Density of States - a Measure of the Difficulty of Optimisation Problems. Parallel Problem Solving from Nature - PPSN IV. (1996) 208–217.
7. Asselmeyer, T., Ebeling, W., Rose, H.: Smoothing representation of fitness landscapes - the genotype-phenotype map of evolution. <http://summa.physik.hu-berlin.de/~torsten/>
8. Jones, T., Forrest, S.: Fitness Distance Correlation as a Measure of Problem Difficulty for Genetic Algorithms. Proceedings of the 6.th ICGA. (1995) 184–192.
9. Bouju, A. et al.: Tabu search for radio links frequency assignment problem. Applied Decision Technologies, London. UNICOM Conf. (1995).
10. Smith, G.D. et al.: EUCLID CALMA Radio Link Frequency Assignment Project; Technical Annexe T- 2.1:A - Supplementary Report.
11. Mühlenbein H.: Evolution in Time and Space - The Parallel Genetic Algorithm. Foundations of Genetic Algorithms. (1991) 316–337.
12. Description of the CALMA Benchmark Problems. <http://dutiosd.twi.tudelft.nl/~rlfap/>

A critical and empirical study of epistasis measures for predicting GA performances: a summary

S. Rochet¹ G. Venturini² M. Slimane² E.M. El Kharoubi²

¹ IUT,

Dép. Génie Electrique,
150, Avenue du Maréchal Leclerc,
13300 Salon de Provence, France.

² Laboratoire d'informatique,
Université de Tours,
64, Avenue Jean Portalis,
37200 Tours, France.

venturini,slimane@univ-tours.fr

Tel: (+33)-2-47-36-14-14,

Fax: (+33)-2-47-36-14-22

Abstract. Epistasis measures have been developed for measuring statistical information about the difficulty of a function to be optimized by a genetic algorithm (GA). We give first a review of the work on these measures such as the epistasis correlation. Then we try to relate the epistasis correlation to the overall performances of a binary GA on a set of 14 functions. The only relation that seems to hold strongly is that a high epistasis correlation implies GA-easy, as indicated by the GA theory of deceptiveness. Then, we show that changing the representation of the search space with transformations that improve epistasis measures does not imply the same increasing in the genetic algorithm performances. These both empirical studies seem to indicate that the generality of epistasis measures is limited.

1 Introduction

Characterizing the difficulty of functions is a key point of the genetic algorithms (GAs) theory (see a survey in (Venturini et al. 1997)). The theory of deceptiveness was probably the first one to be studied (Goldberg 1987) (Whitley 1991). It has, however, several limitations (Grefenstette 1992) like for instance dealing only with static fitness computed on the whole search space rather than dynamic fitness computed on the population. Recently, several statistical measures have been developed for characterizing the difficulty of functions for GAs, and thus for predicting the performances of the GA. The fitness/distance correlation evaluates the correlation that may exist between the distance of individuals to the closest global optimum of the fitness function f and their fitness (Jones and Forrest 1995). This measure however has the drawback of requiring the knowledge of f global optima, which is of course not realizable in practice, or not

realistic because in this case one does not need any search method to find the optimum. Other statistical measures are for instance the operator correlation (Manderick et al. 1991) which can be used in virtual GAs (Grefenstette 1995). This measure evaluates for instance the correlation that may exist between the fitness of parents and their offspring with various operators. Such operators can thus be selected according to this correlation. Finally, epistasis measures (Davidor 1991) approximate the fitness function with another function computed by considering the fitness of schemata independently from each others. This bit by bit approximation measures the dependence that may exist between bits in the representation.

In this paper, we describe with more details epistasis measures and we concentrate on the use of such measures for predicting the performances of the GA. Epistasis measures have the advantage of being easily computable from a set of points generated randomly in the search space. For this reason, they seem to be more interesting than other measures such as fitness/distance correlation. We aim here at answering one main question about epistasis measures, a question which as far as we know has not been answered yet: are these measures useful for predicting the performances of the GA in practice? We provide at least two empirical answers to this question. The first one consists in computing the epistasis of several standard functions and in running the GA on these functions. One can then try to relate the epistasis with the observed performances of the GA. As will be seen, this relation is not strong for the tested functions. Then, the second answer is provided by using epistasis measures in order to guide the search for a better binary representation that could improve the GA performances. Our tests suggest that the epistasis measure of a sample of points can be improved but that the GA performances remain at the same level.

The paper is organized as follows: section 2 provides an introduction to epistasis measures. Section 3 describes our first tests where epistasis measures are compared to GA performances on a set of 14 binary functions. Section 4 introduces the basic techniques for changing the representation in a binary GA and provides experimental results. Section 5 concludes on the usefulness of epistasis measures.

2 Overview of epistasis measures

From initial work on epistasis measures for GAs (Davidor 1991), one may define the variance of epistasis as follows: let X denote the fitness function that goes from $S = \{0, 1\}^l$ to R . To compute the epistasis variance, one must generate a set S_a of binary strings $s_1, \dots, s_n$ and evaluate the fitness of those strings $X(s_1), \dots, X(s_n)$. Then one must compute a function W that approximates X by using the information contained in the sample of n strings. A binary string of length l can be denoted by $s = b_1, b_2, \dots, b_l$ where $b_i \in \{0, 1\}$. The value of W for this string equals:

$$W(s) = [X(b_1 * \dots *) - X(* \dots *)] + [X(* b_2 * \dots *) - X(* \dots *)] + \dots$$

$$+[X(*... * b_l) - X(*...*)] + X(*...*)$$

where $X(*... * b_i * ...)$ denotes the fitness of schema $*... * b_i * ...$ computed on strings of S_a , i.e. the mean fitness of strings in S_a that belong to the schema. W is thus computed by assuming that the fitness function X is a simple function that is separable, i.e. fitness is assigned to bits independently. Of course, this is almost always not the case in practice. So, the difference or similarity between W and X can be viewed as respectively a complexity or simplicity measure of X . Davidor has introduced the variance of epistasis as a measure of difficulty of X . This consists simply in computing $W(s_1), \dots, W(s_n)$ and then in computing the variance of $(W(s_i) - X(s_i))_{i \in [1..n]}$. This measure has, among other things, the drawback of being sensitive to the scale of X .

The epistasis variance can be normalized by the function variance on S_a (Manela and Campbell 1992), but the use of a correlation allows an even better comparison of the epistasis measures between two functions. Recently, Rochet has introduced such a measure of simplicity, based on a correlation, which has the advantage of being comparable from one function X to another and is always positive. It consists in computing the correlation between W and X . This epistasis measure has been extensively studied in (Rochet 1997). We have shown formally that epistasis measures are in fact error measures between the best linear approximation of X and X itself. Therefore such measures can be generalized to continuous representation as in (Rochet et al. 1996).

Other work has been done dealing with epistasis measures like theoretical studies (Reeves and Wright 1995). One should expect in practice that such a measure evaluates the difficulty of a function X for the GA: this comes from the fact that epistasis measures are measures of X linearity, and that the GA performs well on linear functions such as GA-easy functions (Wilson 1991). On such linear functions, the role of recombination is then optimal because the recombination of low order building block with high fitness leads to even better individuals. This is the hypothesis that we try to test in the following, from a practical point of view which should allow direct applications to real-world problems.

3 Relation to GA performances

3.1 Experimental settings

In this paper, we have used 14 binary functions as represented in table 1. The first 7 functions are standard ones in GAs evaluations as mentioned in (Whitley et al. 1995). Functions f_{11} (Ackley 1987), f_{12} (Wilson 1991), f_{13} (Goldberg et al. 1989) and f_{14} (Whitley 1991) are also standard ones in the theory of deceptiveness. The GA that we have used is a standard binary generational GA with binary tournament selection. In the following, p_{mut} , p_{cross} , $|pop|$ and G will denote respectively the probability of mutation and crossover, the population size and the number of generations. We have used one point crossover (1X).

	Function	Intervals of x_i	x_i coded on
1	$x_1^2 + x_2^2 + x_3^2$	$[-5.12, 5.11]$	10 bits
2	$100(x_1^2 - x_2)^2 + (1 - x_1)^2$	$[-2.048, 2.047]$	12 bits
3	$50 + \sum_{i=1}^5 (x_i^2 - 10\cos(2\pi x_i))$	$[-5.12, 5.11]$	10 bits
4	$1 + \sum_{i=1}^2 \frac{x_i^2}{4000} - \prod_{i=1}^2 \cos(\frac{x_i}{\sqrt{i}})$	$[-5.12, 5.11]$	10 bits
5	$1 + \sum_{i=1}^5 \frac{x_i^2}{4000} - \prod_{i=1}^5 \cos(\frac{x_i}{\sqrt{i}})$	$[-5.12, 5.11]$	10 bits
6	$0.5 + \frac{\sin^2(\sqrt{x_1^2 + x_2^2}) - 0.5}{1 + 0.001(x_1^2 + x_2^2)}$	$[-100, 100]$	10 bits
7	$(x_1^2 + x_2^2)^{0.25} (1 + \sin^2 50(x_1^2 + x_2^2)^{0.1})$	$[-100, 100]$	10 bits
8	$ x_1 + 2x_2 + 3x_3 + 4x_4 $	$[-100, 100]$	10 bits
9	$x_1 + 2x_2 + 3x_3 + x_4^2$	$[0, 100]$	10 bits
10	$x_1 x_2 x_3 + x_2 x_3 x_4$	$[0, 100]$	10 bits
11	Ackley's Onemax 30 bits	not applicable	n. a
12	Wilson's non-deceptive 30 bits	n. a.	n. a.
13	Goldberg et al., fully deceptive 30 bits	n. a.	n. a.
14	Whitley fully deceptive 40 bits	n. a.	n. a.

Table 1. The 14 functions used in our tests. Note that the last four functions usually correspond to maximization problems, and we have changed them simply by $Constant - f$.

3.2 Initial epistasis computation

For each function and for sample of points of various sizes, we have firstly computed the epistasis correlation defined in section 2. The results are presented in table 2. A first comment is that the epistasis can be correctly approximated with samples of small size for functions 2, 9, 10, 11, 12, 13 and 14. For the other functions, the true and correct values converge only when the sample is large enough. Therefore, we have used in the following only the values of the last columns (2000 or 3000 points) of table 2. From this table, and if one assumes that the epistasis is correlated with the difficulty of functions for the GA, then one could be tempted to classify the functions in the following way : “easy” functions with high epistasis correlation would be functions 9, 10, and 11, “hard” functions with low epistasis correlation would be functions 1, 3, 4, 5, 6, 7, and 8, and “medium” functions with moderate epistasis correlation would be functions 2, 12, 13 and 14.

3.3 Relation to overall performances

If one considers now the performances of the GA on the 14 functions, then maybe the previous classification of functions holds too for the GA. To test this, we have run the binary GA with several parameters settings. These were $|pop| = 100$, $p_{cross} = 0.8$, $p_{mut} = 0.02, 0.05, 0.1$ and $G = 50, 100$. This amounts to 6 different sets of parameters which were tested over 10 trials. The performance of the GA is

Function	Epistasis correlation computed with				
	300 points	500 points	1000 points	2000 points	3000 points
1	0.294 (0.035)	0.248 (0.047)	0.172 (0.024)	0.125 (0.015)	0.100 (0.016)
2	0.524 (0.038)	0.527 (0.021)	0.514 (0.022)	0.503 (0.009)	0.501 (0.006)
3	0.393 (0.035)	0.304 (0.036)	0.213 (0.010)	0.157 (0.020)	0.127 (0.011)
4	0.252 (0.056)	0.197 (0.033)	0.135 (0.020)	0.094 (0.017)	0.078 (0.006)
5	0.395 (0.021)	0.309 (0.022)	0.213 (0.019)	0.162 (0.006)	0.128 (0.010)
6	0.257 (0.034)	0.178 (0.035)	0.150 (0.015)	0.095 (0.016)	0.074 (0.013)
7	0.241 (0.033)	0.196 (0.032)	0.138 (0.013)	0.088 (0.013)	0.081 (0.010)
8	0.329 (0.025)	0.262 (0.033)	0.182 (0.009)	0.139 (0.014)	0.114 (0.011)
9	0.925 (0.009)	0.936 (0.008)	0.953 (0.002)	0.960 (0.002)	0.963 (0.001)
10	0.854 (0.007)	0.865 (0.008)	0.874 (0.006)	0.877 (0.005)	0.878 (0.003)
11	0.962 (0.008)	0.974 (0.008)	0.986 (0.002)	0.993 (0.002)	0.996 (0.001)
12	0.479 (0.030)	0.471 (0.032)	0.444 (0.029)	0.447 (0.018)	0.443 (0.015)
13	0.436 (0.036)	0.434 (0.029)	0.402 (0.023)	0.380 (0.016)	0.383 (0.015)
14	0.591 (0.039)	0.580 (0.039)	0.555 (0.020)	0.543 (0.021)	0.551 (0.008)

Table 2. Epistasis correlation obtained for the 14 functions and averaged over 10 samples. Numbers in parenthesis are the standard deviations.

then measured by the obtained minimum averaged over these 10 runs. However, to compare the results and to compute a more useful performance measure, one must relate this performance to a basis which depends on each function. To achieve this, the performance of the GA on a given function is in fact related to the performance of a simple random generation of strings. Given a value of $|pop|$ and G , the GA evaluates a given number of strings, here 5000 or 10000. Thus, we have used a simple random generation of exactly the same number of strings as for the GA. This method, denoted by RANDOM in the following, consists simply in keeping the best randomly generated string. Its results have also been averaged over 10 runs.

A measure of a function difficulty for the GA can be computed by the ratio $DIFF = \frac{RAN-GA}{RAN}$ where RAN and GA denote respectively the minima found by RANDOM and the GA. This kind of measures are commonly used in combinatorial optimization in order to compare methods. If $DIFF$ is close to 1, then the GA performs much better than RANDOM. On the other hand, if $DIFF$ is negative, then the GA performs very poorly compared to RANDOM. Typical results obtained with $|pop| = 100$, $p_{cross} = 0.8$, $p_{mut} = 0.1$ and $G = 100$ are presented in table 3. From this table and according to the values of $DIFF$, one could classify the functions as follows: easy functions with high values of $DIFF$ would be functions 1, 9, 10, and 11, while hard functions with low values of $DIFF$ would be functions 7, 13 and 14. The first remark here is that functions with a very high epistasis correlation (like functions 9, 10 and 11) are indeed easy for the GA in practice. So the relation that seems to hold experimentally is “high epistasis correlation $\Rightarrow$ easy for GA”. Hopefully, this is the results which is

Function	Epistasis (3000 points)	$DIFF = (RAN - GA)/RAN$	RAN	GA
1	0.100	0.760	0.074	0.018
2	0.501	0.532	0.003	0.002
3	0.127	0.357	18.095	11.636
4	0.078	0.447	0.155	0.086
5	0.128	0.328	3.736	2.509
6	0.074	0.406	0.020	0.012
7	0.081	0.255	1.288	0.960
8	0.114	1.000	0.039	0.000
9	0.963	0.694	44.045	13.464
10	0.878	1.000	0.000	0.000
11	0.996	0.667	4.800	1.600
12	0.443	0.500	6.800	3.400
13	0.383	0.243	22.200	16.800
14	0.551	0.280	45.000	32.400

Table 3. Results obtained by the GA with $|pop| = 100$, $p_{cross} = 0.8$, $p_{mut} = 0.1$ and $G = 100$ in comparison with a random search and averaged over 10 runs.

provided by the theory: linear functions are non-deceptive functions. Now, if one is looking for typical examples of low epistasis correlation and high GA performances, then functions 1 and 8 comply with this specification. So a low epistasis correlation is not a good indication about the true difficulty of the function.

Now, let us test the relation between the epistasis correlation and the functions' difficulty on the 6 sets of parameters. For this purpose, an overall measure for the 14 functions and for a given set of parameters is to compute the correlation that may exist between the epistasis measure (with 3000 points) and the measure of difficulty $DIFF$. Let us denote this correlation by $Corr(Epist, DIFF)$. For instance, in the data represented in table 3, this correlation equals 0.36. Table 4 computes this correlation for the 6 sets of parameters. This correlation seems to be increasing with the probability of mutation. It also seems to be higher at the beginning of the GA search rather than at the end. In all cases, this correlation is not very high and is not really significant.

4 Remapping the binary search space with epistasis measures

4.1 Principle

We consider in this section that it could be possible, at a low computational cost, to find a better binary representation of a search space than the initial representation, by using the epistasis measure. We thus look for a new representation of the search space where the correlation of epistasis would be closer to 1 than

$ pop $	p_{cross}	p_{mut}	G	$Corr(Epist, DIFF)$
100	0.8	0.02	100	-0.07
100	0.8	0.05	100	0.25
100	0.8	0.1	100	0.36
100	0.8	0.02	50	0.04
100	0.8	0.05	50	0.30
100	0.8	0.1	50	0.46

Table 4. Correlation between the measured epistasis correlation and the measured difficulty of functions for the 6 sets of parameters (and for the 14 functions over 10 trials).

in the initial search space. Then, using this new search space within the GA, we hope to make the function X easier and to improve the GA performances. Of course, this may not work for all functions: the problem of finding a good representation is most probably as hard as the initial optimization problem itself (Liepins and Vose 1990). What we believe is that maybe, in some cases, a better representation that increases the epistasis correlation can be easily found.

Let us consider a transformation R of the initial search space $S = \{0, 1\}^l$ into a new search space $S' = \{0, 1\}^l$. Any binary string s of S is changed to another string s' of S' with R . R must satisfy several constraints. For instance, one property to be kept absolutely is that the optima in S are changed into the optima in S' . Thus, searching for an optimum in S' will lead, once this optimum is found, directly to an optimum of S by applying R^{-1} to this optimum. R must thus be reversible/bijective, and furthermore, it must be computable in polynomial time in l in order to be realizable in practice. In order to evaluate a binary string $s' \in S'$, we use the following fitness function: $X'(s') = X(R^{-1}(s')) = X(s)$ where $s = R^{-1}(s')$. Any optimum in one of the search space will also be an optimum in the other search space. Increasing the epistasis correlation thus seems to be a good idea because it may linearize the function X . This leads to the following general algorithm:

1. Generate randomly a sample of strings $S_a \subset S$, and compute the epistasis correlation $e_{corr}(S_a)$,
2. Find a new representation R^* , such that $e_{corr}(R^*(S_a)) > e_{corr}(S_a)$, (where e_{corr} denotes the epistasis correlation),
3. Use the GA to find an optimum s'^* in $S' = R(S)$ (all strings in S' are evaluated by using the new fitness function $X'(s') = X(R^{-1}(s'))$),
4. Translate this optimum in the original search space S with the relation $s^* = R^{-1}(s'^*)$.

4.2 Remapping bits 3 by 3

The problem now is to generate transformations R from $\{0, 1\}^l$ into itself which are bijective and easy to compute (see point 2 of the previous algorithm). One possibility consists in considering the bits of an individual s 3 by 3, and to remap each group of 3 bits according to the same bijective transformation r . The general transformation R is thus obtained by applying several times a simpler transformation r that works only on groups of 3 bits. r can be simply encoded in a lookup table of 8 entries, like for instance:

Input	000	001	010	011	100	101	110	111
↓	↓	↓	↓	↓	↓	↓	↓	↓
Output	101	011	001	100	000	010	111	110

Since r must be bijective, there are only $8!$ such transformations r of 3 bits to 3 bits. r is also easily reversible by considering the lookup table from outputs to inputs. These desired properties are also kept when considering R , since R is a repeated combination of a smaller transformation r . In addition, since here $l = 24, 30, 40$ or 50 , then l is not necessarily a multiple of 3, and thus the 2 (or 1) remaining bits at the end of the string are left unchanged.

4.3 Generating and evaluating transformations

Transformations R of the previous type can be generated randomly simply by performing permutations in the output row of the lookup table of r . The generated transformations can be evaluated according to their associated epistasis correlation. This requires simply to encode with R the strings generated in the sample of strings S_a . One should notice that this initial set is the same for every transformation, and since $X(R(s)) = X(s)$, one does not need to use the evaluation function on the transformed binary strings of $R(S_a)$ in order to evaluate a given transformation R . One can thus generate as many transformations as desired without evaluating new strings with X . This is a very important point which has the advantage in practice of searching for a better representation with no additional cost from the evaluation function point of view.

The best generated transformation R^* is the one which maximizes the epistasis correlation. If the epistasis correlation has not been improved, then $R^* = Identity$.

4.4 Experimental results

In this experiment, we have applied the previous algorithm to every functions of table 1. For each tested function, one can thus record the following information: the initial epistasis correlation and the initial performance of the GA without any transformation (denoted by E_{init} and GA_{init} in the following), and then the possibly improved final epistasis correlation and the final GA performance in the new search space (denoted respectively by E_{final} and GA_{final} in the following).

Function	E_{init}	E_{final}	GA_{init}	GA_{final}
1	0.112	0.638	0.0035	0.0082
2	0.505	0.723	0.0068	0.0046
3	0.160	0.394	5.7565	7.7567
4	0.091	0.190	0.0370	0.0568
5	0.152	0.521	1.4800	1.5076
6	0.097	0.121	0.0100	0.0132
7	0.097	0.119	0.7561	0.7384
8	0.145	0.233	0.0000	0.0000
9	0.959	0.959	4.0872	2.9024
10	0.875	0.875	0.0000	0.0000
11	0.993	0.993	0.0000	0.1000
12	0.439	0.909	0.2000	0.0000
13	0.386	0.919	11.0000	2.2000
14	0.555	0.636	24.0000	20.8000

Table 5. Results obtained when remapping the search space according to the epistasis correlation.

Table 5 gives the obtained results on the 14 functions averaged over 10 runs. The number of strings generated was 2000, and 20 transformations were generated randomly at each run in order to generate R^* . The parameters of the GA were $|pop| = 100$, $G = 100$, $p_{cross} = 0.8$ and $p_{mut} = 0.05$. The epistasis correlation is improved for most of the time except for functions 9, 10 and 11 where $R^* = Identity$. However, the performances of the GA are not improved with these apparently better representations, except for functions 12 and 13. Perhaps, this is due to the fact that functions 12 and 13 are defined on groups of 3 bits only and are thus very simple compared to the others. Maybe, even with a good encoding, the problem remains hard to optimize for the other functions and the improvement in performances cannot be detected.

One could argue that a problem of generalization takes place, like in machine learning: the best transformation generated with S_a improves the epistasis correlation of strings in $R^*(S_a)$, but maybe this improvement does not generalize to other strings of S . We have performed some tests that reject this hypothesis. This has consisted simply in generating another set of string S_b and checking that the transformation R^* learned on S_a also improves the epistasis correlation of transformed strings of S_b .

5 Conclusion and discussion

We have studied in this paper epistasis measures from an experimental point of view in order to check how useful these measures can be in order to predict the GA performances. We have presented two kinds of tests: trying to relate the epistasis measure to the GA overall performances evaluated in comparison with

a random search, and trying to find better binary representations that would increase the epistasis correlation of a sample of strings and possibly improve also the GA performances. Apparently, for the 14 tested functions, epistasis correlation predicts neither the GA performances nor the changes of representation with simple transformation. The only relation which seems to hold is that functions with high epistasis correlation are rather easy for GAs, and this can be related to other recent studies such as (Soule and Foster 1997).

These negative results should be tempered by several points. We have used only first-order epistasis. Maybe, second-order epistasis computed with schemata of order 2 would have given better information about the functions. Quadratic function like f_1 may have an epistasis correlation of order 2 that is closer to 1. Also, maybe our transformations are too simple: they change the search space in a good way on average but the relevant areas of the space may remain as hard as in the initial space. We believe that epistasis measures are interesting compared to others because they require low computational costs only, which is a necessary condition for bridging the gap between the theory and practice of GAs.

The immediate perspectives of this work will be presented in (Rochet et al. 1997). We try to relate also the epistasis correlation to the performances of 1X and UX crossovers. We use also more complex transformations. We have shown recently that epistasis measures can be generalized to real coded representations by using linear approximation of a function in R^l (Rochet et al. 1996). This allows us to perform similar experiments in continuous spaces.

References

- Ackley D.H. (1987), *A connectionist machine for genetic hillclimbing*, Boston, MA: Kluwer Academic.
- Davidor Y. (1991). Epistasis Variance: A Viewpoint on GA- Hardness. In *Foundations of Genetic Algorithms*. Gregory J. E. Rawlins (Ed). Morgan Kaufmann Publishers, San Mateo. pp 23-35.
- Goldberg D.E. (1987), Simple genetic algorithms and the minimal deceptive problem, *Genetic Algorithms and Simulated Annealing*, L. Davis (Ed), Morgan Kaufmann, pp 74-88.
- Goldberg D.E., Korb B. et Deb K. (1989), Messy genetic algorithms: motivations, analysis, and first results. *Complex systems* 4, pp 415-444.
- Grefenstette J.J. (1992), Deception considered harmful, *Proceedings of the second workshop on Foundations of Genetic Algorithms*, 1992, D. Whitley (Ed), Morgan Kaufmann, pp 75-91.
- Grefenstette J. J. (1995). Predictive Models Using Fitness Distributions of Genetic Operators. In *Foundation of Genetic Algorithms* 3. D. Whitley and M. Vose (Eds), Morgan Kaufmann, San Mateo, CA.
- Holland J.H. (1975), *Adaptation in natural and artificial systems*. Ann Arbor: University of Michigan Press.
- Jones T. and Forrest S. (1995), Fitness distance correlation as a measure of problem difficulty for genetic algorithms, *Proceedings of the Sixth Interna-*

- tional Conference on Genetic Algorithms, 1995, L.J. Eshelman (Ed), Morgan Kaufmann, pp 184-192.
- Liepins G. E., Vose M.D. (1990). Representationnal issues in genetic optimization. *Journal of Experimental and Theoretical Artificial Intelligence* 2 Research paper, pp 101-115.
- Manderick B., de Weger M. and Spiessens P. (1991), The genetic algorithm and the structure of the fitness landscape, *Proceedings of the Fourth International Conference on Genetic Algorithms*, 1991, R.K. Belew and L.B. Booker (Eds), Morgan Kaufmann, pp 143-150.
- Manela M. and Campbell J.A. (1992), Harmonic analysis, epistasis and genetic algorithms, *Proceedings of the Second Conference on Parallel Problem Solving from Nature 1992*, R. Manner et B. Manderick (Eds), Elsevier, pp 57-64.
- Reeves C.R., Wright C. C. (1995). Epistasis in Genetic Algorithms: An Experimental Design Perspective. *Proceedings of the sixth International Conference on Genetic Algorithms*. Larry J. Eshelman (Ed), Morgan Kaufmann, San Mateo, CA. pp 217-224.
- Rochet S. (1997). Epistasis in genetic algorithms revisited. To appear in *Information Sciences*.
- Rochet S., Slimane M. and Venturini G. (1996), Epistasis for real encoding in genetic algorithms, *IEEE ANZIS'96*, V. L. Narasimhan and L. C. Jain (eds), Australia, pp. 268-271.
- Rochet S., G. Venturini, M. Slimane (1997), A critical study of epistasis measures, in preparation.
- Soule T. and Foster J.A. (1997), Genetic algorithm hardness measures applied to the maximum clique problem, *Proceedings of the seventh International Conference on Genetic Algorithms*, 1997, T. Baeck (Ed.), Morgan Kaufmann, pp 81-88.
- Venturini G., Rochet S. and Slimane M. (1997), Schemata and deception in binary genetic algorithms: a tutorial, to appear in *Control and Cybernetics*, Special Issue on Evolutionary Algorithms, M. Schoenauer and Z. Michalewicz (Eds).
- Whitley D. (1991), Fundamental principles of deception in genetic search, *Proceedings of the first Workshop on Foundations of Genetic Algorithms*, 1991, G.J.E. Rawlins (Ed), Morgan Kaufmann, pp 221-241.
- Whitley D., Mathias K., Rana S. and Dzubera J. (1995), Building better test functions, *Proceedings of the Sixth International Conference on Genetic Algorithms*, Eshelman L.J. (ed), Morgan Kaufmann publishers, pp. 239-246.
- Wilson S.W. (1991), GA-easy does not imply steepest-ascent optimizable, *Proceedings of the Fourth International Conference on Genetic Algorithms*, 1991, R.K. Belew and L.B. Booker (Eds), Morgan Kaufmann, pp 85-89.

A priori comparison of binary crossover operators: No universal statistical measure, but a set of hints.

Leila Kallel and Marc Schoenauer

CMAP – URA CNRS 756

Ecole Polytechnique

Palaiseau 91128, France

`FirstName.Name@polytechnique.fr`

Abstract. The choice of an operator in evolutionary computation is generally based on comparative runs of the algorithms. However, some statistical a priori measures, based on samples of (parents-offspring), have been proposed to avoid such brute comparisons. This paper surveys some of these works in the context of binary crossover operators. We first extend these measures to overcome some of their limitations. Unfortunately, experimental results on well-known binary problems suggest that any of the measures used here can give false indications. Being all defined as averages, they can miss the important parts: none of the tested measures have been found to be a good indicator alone. However, considering together the mean improvement to a target value and the Fitness Operator Correlation gives the best predictive results. Moreover, detailed insights on the samples, based on some graphical layouts of the best offspring fitnesses, seem to allow more accurate predictions on the respective performances of binary crossover operators.

1 Introduction

The most simple and common way to compare some parameter settings (including the choice of operators) of evolutionary algorithms (EAs) is to run numerous experiments for each one of the settings, and to compare averages of on-line or off-line fitness. This trial-and-error experimental method is unfortunately time consuming. Nevertheless, the lack of efficient and robust measures of difficulty and/or adequation of settings for a given problem has made this experimental comparative method very popular since the early conferences of EAs (see e.g. [14]).

Some studies have proposed heuristical approaches to try to estimate the efficiency of different operators a priori. These methods are usually based on statistical measures computed on random samples of points of the search space. Two kinds of such methods can be distinguished :

The first set of methods do not take into account the specific characteristics of operators, but rather implicitly rely on conceptual notions that are common to all operators. For instance, the strong causality of mutations is assumed in [10], where a distance in the genotype space is used to assess correlation length, and in [8], where the correlation between fitness and distance to optimum is described as a good predictor of EA performances. In the same line, the respectfulness

of operators toward schemata [6, 12] or epistasis [4] is used to characterize the suitability of the representation and its associated operators.

The second kind of methods directly address the issue of operator performance relative to the fitness landscape. In the early days of evolution strategies, Rechenberg [13] derived the famous 1/5 rule by theoretical calculation of the expected progress rate toward the optimum for a zero mean Gaussian mutation, on the sphere and corridor models in $\mathbb{R}^n$, in relation to the probability of a successful mutation.

More recently, Weinberger [15] introduced the notion of fitness operator correlation, for asexual operators, as the correlation between the fitnesses of the parents and their iterated offspring. Manderick [10] adapted this correlation measure to crossover operators. He devoted attention to the correlation between mean parental fitness and mean offspring fitness, and indicated the possible usefulness of this measure to assess the role of an operator on the NK-landscapes and the TSP problems.

However, as noted by Altenberg [1], such fitness operator correlation suffers from theoretical limitations: a maximal correlation is possible even in the absence of any improvement of fitness from parents to offspring. Altenberg further suggested that the performance of a genetic algorithm is related to the probability distribution of offspring fitness indexed by their parents' fitness that result from application of the genetic operator during the GA. Unfortunately, Altenberg noted that empirical estimation of the fitness probability distribution could be problematic if the searched surface has multiple domains of attraction.

Grefenstette [7] used the probability distribution of offspring fitnesses indexed by the mean parents fitnesses to yield a first-order linear approximation of the suitability of a genetic operator. Another approach was suggested by Fogel and Ghoeil [5]: to compare different selection schemes and operators, they use the mean improvement of offspring, and concludes that the uniform intermediate recombinations yields better results (higher mean improvement) than the one-point crossover, for all three selection methods they tested, on some two-dimensional functions in the real-valued framework.

This paper is concerned with statistical a priori measures that can be used to assess crossover efficiency: All the above measures (fitness operator correlation, probability distributions, and mean improvement) have been tested separately. As far as we know, no comparison has been made of their efficiency on the same problem. The goal here is to achieve such a comparison on different binary problems from the GA literature. The different selection schemes will not be discussed here, and for all experiments linear ranked-based selection will be used.

The paper is organized as follows. Section 2 presents in detail and discusses the statistical measures briefly introduced above. A modified version of the fitness operator correlation is introduced to address some of its limitations. Mean improvement method is modified to only account for strictly positive improve-

ments. Section 3 experimentally compares the estimates of those different measures on various test cases in the light of actual results of GA runs, demonstrating the inadequacy of the fitness operator correlation for most problems, and the potentialities of the statistical measures based on best offspring fitnesses. These results yield a better understanding of the domain of validity and the limitations of each statistical measure. Moreover, a detailed look at some samples appeals for on-line measures: Further research directions are sketched in the last section, in light of the obtained results.

2 Comparing Crossover operators

This section details some of statistical measures that have been proposed in previous work, and discusses some of their limitations. This leads to proposing alternative measures.

Consider the problem of optimizing a real-valued fitness function $\mathcal{F}$ defined on a metric search space $\mathcal{E}$ and suppose that there exists one global optimum of $\mathcal{F}$. The common basis of all statistical measures described below is the use of a sample of random individuals in $\mathcal{E}$. The operators to be assessed are then applied on all (or part of) the sample.

2.1 Mean Fitness Operator Correlation

Manderick & al. [10] define the Mean Fitness Operator Correlation (M-FOC) as follows:

From a sample of n couples of individuals (x_i, y_i) , let (f_{pi}) be the mean fitness of couple i , and (f_{ci}) the mean fitness of their two offspring with crossover OP. The *M-FOC coefficient* of the operator OP is defined by:

$$\frac{\frac{1}{n} \sum_{i=1}^n (f_{pi} - \bar{f}_p)(f_{ci} - \bar{f}_c)}{\sigma_{F_p} \sigma_{F_d}}, \quad (1)$$

where $\bar{f}_p, \bar{f}_c, \sigma_{F_p}$ and σ_{F_d} are respectively the means and standard deviations of parents' and offspring mean fitness.

This coefficient measures the extent to which the mean fitness of a couple of parents can give some information about the mean fitness of their offspring. A very high correlation between parents and offspring (M-FOC coefficient close to 1) indicates that offspring mean fitness would be almost proportional to parental mean fitness. No correlation (M-FOC coefficient close to 0) shows that no information about offspring mean fitness can be gained from the parents mean fitness. GAs using high-correlated operators are expected to achieve best performances as demonstrated in [10] on the NK-landscapes and the TSP problems.

Unfortunately, suppose there is a quasi-linear relationship between the mean fitnesses of both sets of parents and offspring. The M-FOC correlation will be maximum (1), whatever the slope of that linear relation. In particular, if that slope is less than 1, no offspring fitness can be better (in mean) than its parents': the correlation can be maximal when no improvement is to be expected! However

unrealistic this particular case can be, it nevertheless shows the limits of M-FOC as a measure of operator’s efficiency.

In that line, as noted by Fogel and Ghozeil [5], M-FOC cannot be used on problems that yield zero mean difference between parents and offspring fitnesses, as in the case of linear fitness functions with real-valued representations. The correlation is always maximal, and cannot reflect the high dependency of their convergence rate on the the standard deviation of the Gaussian mutation [13].

2.2 Best Fitness Operator Correlation

To address the above-mentioned limitations of M-FOC, we propose to study the correlation between the best (rather than mean) parent and offspring fitnesses. The definition of the B-FOC coefficient is the same as in equation 1, except that $\bar{f}_p, \bar{f}_c, \sigma_{F_p}$ and σ_{F_d} are replaced by the means and standard deviations of the best fitnesses of parents and offspring. This coefficient should avoid some of the ambiguities pointed out in the preceding subsection.

2.3 Mean Improvement

Fogel and Ghozeil [5] use 1000 couples of parents to compare different selection schemes and operators. To generate each couple, 100 random individuals are drawn. The couple is then chosen among them either randomly, or as the two best individuals, or as one random and the best. The mean improvement of the best offspring (of each couple) over the best individual of the initial 100 individuals is then computed. Such mean improvements are finally averaged over the 1000 couples. As noted in [5], one limitation of this approach is its computational cost (102.000 fitness evaluations). The following definitions are a tentative to achieve the same goals at lower cost: Consider a sample of N random individuals as $N/100$ disjoint subsets of individuals. Among each of these 100 size subsets, 100 couples of parents are selected randomly (selection scheme is presented in 3.1). Two cases are considered:

Mean Improvement over the best parent

In the same line as Fogel and Ghozeil’s second procedure (computing the improvement over the best parent), we propose to compute the improvement of the best offspring of each couple over its best parent. The mean improvement of N couples of parents is computed as follows:

$$\text{MI-P} = \frac{1}{N} \sum_{i=1}^N \max(0, \mathcal{F}(\text{best offspring}) - \mathcal{F}(\text{best parent}))$$

Mean Improvement to target

Fogel and Ghozeil’s first procedure amounts to computing improvement of the offspring over a target set to the best fitness of the current set of 100 individuals. Rather, we define a target $\mathcal{T}$ as the median of the set $\{f_i\}_{i=1, \frac{N}{100}}$ where f_i is the fitness of the best individual among those of the i th -100 size- subset generated. The mean improvement to target is then computed as follows:

$$\text{MI-T} = \frac{1}{N} \sum_{i=1}^N \begin{cases} \max(0, \mathcal{F}(\text{best offspring}) - \mathcal{T}) & \text{if } \mathcal{F}(\text{best offspring}) > \mathcal{F}(\text{best parent}) \\ 0 & \text{otherwise} \end{cases}$$

2.4 Probability of Improvement

The above definitions of MI-P and MI-T do not consider the number of offspring that actually are source of improvement. In order to separate the mean improvement from the probability of improvement, the latter is defined as the fraction of offspring actually having a better fitness than their parents (PI-P) or the target (PI-T). Two new measures are then defined, termed MI-P+ and MI-T+, by counting in the definitions of MI-P and MI-T only those offspring.

3 Experimental Validation

3.1 Experimental conditions

From now on, the search space will be the binary space $\{0,1\}^n$ ($n = 900$ for all problems, except the Royal Road), and the underlying EA some standard binary GA: population size 100, linear ranked-based selection, crossover rate of 0.6, mutation rate of $1/n$ per bit, elitist generational replacement.

Two operators are compared: The uniform crossover (UC) and the one-point crossover (1C). For each of them, the statistical measures are computed either on samples of 10000 individuals, with one crossover per parent, or on samples of 200 individuals, each couple generating 100 offspring (by repeating 50 crossovers operations). The parents are selected with a linear ranking selection applied to each disjoint subset of 100 individuals. In both cases, the computational cost is 20000 fitness evaluations per operator.

The goal of these experiments is to compare operators under the same conditions of initialization and selection. Therefore neither the selection scheme (as in [5]) nor the initialization procedure (as in [9]) will be modified.

The stability of all the following statistical measures have been checked carefully. The relative difference between measures on UC and 1C, has been found stable over 10 trials in most cases. However, all unstability cases are marked with a “*” in the tables.

3.2 Test cases

Onemax problems

In the well-known *onemax* problem, the fitness function is simply the number of 1's in the bitstring. Hence, the fitness is equal to n minus the distance to the global optimum. In order to avoid structural bias, modified versions of the onemax problem were designed: The fitness function is defined as the Hamming distance to a fixed bitstring. That bitstring is chosen to have a certain number

O of 1's, randomly placed in the bitstring. Such problem is termed the $(O, n-O)$ -*onemax* problem. Different values for O were experimented: (900, 0), (800, 100) and (450, 450).

Gray-coded Baluja functions F1 and F3

Consider the functions of k variables $(x_1, \dots, x_k)$, with $x_i \in [-2.56, 2.56]$ [2]:

$$F_1(\mathbf{x}) = \frac{100}{10^{-5} + \sum_{i=0}^{k-1} |y_i|}, y_0 = x_0 \text{ and } y_i = x_i + y_{i-1} \text{ for } i = 1, \dots, k-1$$

$$F_3(\mathbf{x}) = \frac{100}{10^{-5} + \sum_i |.024 * (i+1) - x_i|}$$

They reach their maximum value of 10^7 at point $(0, \dots, 0)$. The Gray-encoded versions of F_i , with 100 variables encoded on 9 bits each are considered.

The 4 peaks problem

In the FourPeaks problem [3], the fitness is the maximum of the length of the sequence of 0's starting at the first bit position and the length of the sequence of 1's ending at the last position, plus a reward of n if both sequences are larger than a given threshold T . There are two global optima, made of a block of 0's followed by a block of 1's, of lengths T and $n-T$ or $n-T$ and T . However, there are two local optima, the all 1's and all 0's bitstrings, from which it is difficult for a GA to escape. The most difficult instance of that problem (i.e. $T = n/4$) is used here.

The Ugly problem

The Ugly problem [16] is defined from an elementary deceptive 3-bit problem ($F(\mathbf{x}) = 3$ if $\mathbf{x} = 111$; $F(\mathbf{x}) = 2$ for $\mathbf{x}$ in $0\star\star$, and $F(\mathbf{x}) = 0$ otherwise). The –deceptive – full problem is composed of 300 concatenated elementary deceptive problems. The maximum is the all 1's bitstring.

The Royal Road problem

The 64-bits Royal Road problem used here was conceived to study into details the combination of features most adapted to GA search (laying a *Royal Road*). A precise definition as well as an analysis of the unexpected difficulties of this problem can be found in [11].

3.3 Results

For all problems above, the statistical measures defined in section 2 have been computed, and their predictions compared to actual GA runs.

Average on-line results of the runs (over 21 runs) are presented in Figure 1, and the statistical measures for both uniform (UC) and one-point (1C) crossovers are presented in tables 1 and 2. Each table presents FOC and MI-P coefficients with one crossover per parent, as well as MI-T coefficients for both 1 and 50 crossovers per parent. Only one of the three onemax problems is presented since the two others gave almost the same results.

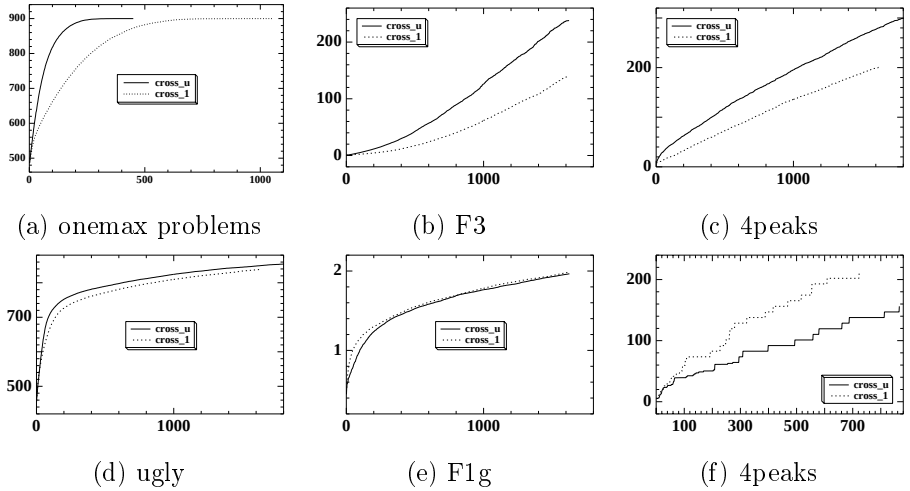


Figure 1: *Average on-line results of 21 GA runs on different problems (fitness $\times$ generation): influence of the crossover operators.*

Large unstability has been found for MI-T with the Royal Road problem. This is due to the highly discontinuous fitness values it yields (observed values are limited to 1, 5.2, 5.4, 21.6). Hence, when computing MI-T on few individuals, this can amount to contradictory results for different samples. More generally, MI-T has been found slightly unstable. This is not surprising when considering that very low probabilities are involved (lines PI-T in the tables). However, when considering the case of 50 crossovers per parent, the unstability does no longer alter the significance of the differences between MI-T coefficients of operators. A last general remark is that all MI coefficients should be considered relatively to the actual fitness range in the sample.

FOC Coefficients The only case where higher M-FOC coefficient and better convergence coincide is obtained with the F1g problem! On the 4peaks, Ugly and F3g problems the reverse situation is observed: Better on-line performances (Figure 1) are obtained for the operator with lower M-FOC values (Tables 1 and 2). On the opposite, runs for the Royal Road problem with either UC or 1C are similar (the 99% T-test fails) until generation 500, though, their respective M-FOC coefficients are quite different (Table 2).

On the same problems, B-FOC give similar indications than M-FOC, and hence suffer the same defects. On the Onemax problems, where the M-FOC always equals 1, the B-FOC coefficient reaches higher values for the 1C, while the runs (Figure 1) using UC are far better than those using 1C.

Hence neither M-FOC nor B-FOC seems an accurate predictor of relative performance of operators, at least under linear ranking selection.

	onemax problems		F3g		4peaks	
	UC	1C	UC	1C	UC	1C
M-FOC	1	1	0.88	0.99	0.7	0.93
B-FOC	0.7	0.8	0.6	0.8	0.6	0.99
MI-P	3.4(0.03)	2.7(0.03)	0.01	0.008	0.4	0.002
MI-P+	7(0.08)	6(0.06)	0.02	0.016	2	2
PI-P	0.23	0.23	0.26	0.25	0.1	0.003
MI-T	0.07(0.012)	0.06(0.016)	0.00026	0.00021	0.1	0.002
MI-T+	5.6(0.3)	5.6(0.5)	0.02	0.02	2.2	1.8
PI-T	0.006	0.005	0.006	0.005	0.005	6e-5
MI-T (50)	3.2(1)	1.2(0.7)	0.01	0.004	0.6	0.02
MI-T+ (50)	8(1)	6(1)	0.028	0.025	2.7	1.7
PI-T (50)	0.02	0.018	0.018	0.016	0.01	0.0001

Table 1: *Mean(standard deviation) of statistical a priori measures for different problems, on a 10000 individuals sample, 2 offspring for each parent with the uniform and one-point crossover operators. Bold fonts indicates measures on a 200 individuals sample, and 100 offspring for each parent. The + concerns strictly positive improvements only.*

	ugly		F1g		royal	
	UC	1C	UC	1C	UC	1C
M-FOC	0.74	0.99	0.2	0.7	0.18	0.92
B-FOC	0.5	0.8	0.1(0.01)	0.6(0.005)	0.18	0.91
MI-P	5.9(0.07)	3.6(0.05)	0.05(7e-4)	0.03(4e-4)	0.19	0.019
MI-P+	12.3(0.1)	7.7(0.1)	0.09(9e-4)	0.05(8e-3)	4.2	4
PI-P	0.27	0.23	0.3	0.28	0.02	0.002
MI-T	0.08(0.01)	0.07(0.008)	7e-4(1e-4)	5e-4(1e-4)	e-3(8e-4)	2e-4(4e-4)
MI-T+	6.6(0.5)	6.8(0.5)	0.05(2e-3)	0.05(5e-3)	1.5(1.2)	0.9(2.3)
PI-T	0.006	0.005	0.007	0.006	0.0004	0.0001
MI-T(50)	5(2)	1.3(0.8)	0.05(0.02)	0.02(7e-3)	0.07(0.07)	0.01(0.02)
MI-T+(50)	10(2)	8(1.5)	0.07(0.01)	0.06(0.01)	2.5(2.8)	0.6(4)
PI-T(50)	0.018	0.016	0.016	0.015	4e-4(1e-4)	4e-4(8e-4)

Table 2: *Mean(standard deviation) of statistical a priori measures for different problems, on a 10000 individuals sample, 2 offspring for each parent with the uniform and one-point crossover operators. Bold fonts indicates measures on a 200 individuals sample, and 100 offspring for each parent.*

Mean Improvement to Parents If we consider the F1g problem, the results of the runs (figure 1), show that the one-point crossover average run is better than the uniform one until generation 200 (confirmed with a 99 % confidence T-test). But mean improvement to parents values (Table 2), suggest the opposite. A close look at the plot of best parents $\times$ offspring (Figure 2), shows that UC

yields better improvement than OP only at low fitness values of (best) parents. The situation is reversed for higher fitnesses. This situation could have been predicted by the Best FOC results: a 0.7 correlation for the 1C means that the best offspring's fitness is almost proportional to its best parent's fitness, while the 0.2 value for the UC means on the otherhand that well-fitted offspring can be obtained whatever the (best) parent's fitness. The misleading results of improvement to parents are so no longer surprising. It seems that MI-P values can be compared reliably when the B-FOC coefficients are similar, as it is the case of the F3g and the three onemax problems.

For the Ugly and Royal Road problems, the situation is similar to that of F1g problem. Average improvements to parents are very different in favor to UC, whereas the runs show identical performances: the T-test with 99 % confidence finds no difference between average runs until generation 30 (Ugly) and 500 (Royal Road). Again, at the opposite of 1C, the UC yields good fitness offspring from low fitness parents, as illustrated by the B-FOC coefficients.

Note however that the reverse is not true: different B-FOC values do not necessarily imply false MI-P previsions, as illustrated by the 4peaks problem.

Considering the probabilities of improvement (PI-P and MI-P+) for the F1g and Ugly problems does not improve the results: the same bias as for MI-P is observed, for the same reasons (different B-FOC coefficients).

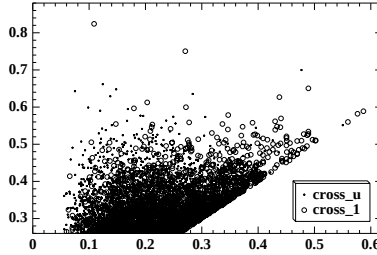


Figure 2 : *Fitness of best offspring $\times$ fitness of best parent, for the offspring that outperformed their parents, on the F1g problem.*

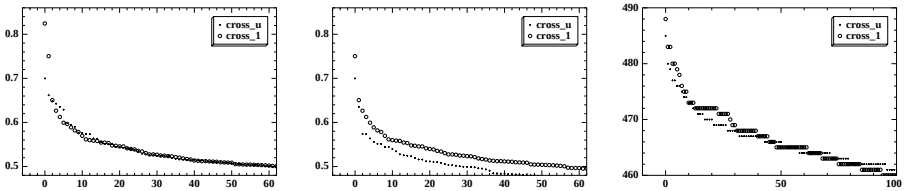
Mean Improvement to Target The case of 1 crossover per parent is found to be rather unstable and often involves very low improvement probabilities, which does not help comparison. Note for example that for the onemax problems (table 1), the observed differences of MI-T, with one crossover per parent, are very small (around 0.01) relatively to the range of fitness values of the problem ([400,500]). Hence only the case of 50 crossovers per parents (bold fonts in tables) will be considered in the following.

When considering the improvement to target, the dependency on the parents' fitness vanishes. So no bias is induced by the dissymmetric repartitions of improvement to parents. Nevertheless, for F1g and Ugly problems, MI-T values (Table 2) are still indicating some superiority of UC, whereas the opposite actually happens during the GA runs for F1g, and Ugly runs are similar until

generation 30 (Figure 1). The main reason for that is probably that averaging the improvements hides both the evolution of their differences and the best fitted offspring obtained: For the F1g problem, when getting to higher parents' fitness, the 1C gets better improvement to target than the UC as can be seen for instance on the plot (best parent $\times$ best offspring) of Figure 2.

Considered together with the probability of improvement, some additional information about the sample repartition can be obtained. In fact, IP-T does take into account the whole set of generated offspring, while MI-T does only take into account the set of best offspring of each parent. For the problematic cases of F1g and Ugly, MI-T+ values are still favoring the wrong operator, but with much smaller differences than MI-T, and IP-T values are close for both operators. MI-T+ still suffers from the bias of averaging, and does nevertheless hide the most important information about offspring, which is their maximal fitnesses, and how they evolve when higher parents are considered.

Further directions Other measures, not presented here, have been computed for both UC and 1C operators; such as mean fitness values of all offspring, or of all offspring that outperformed their parents. None of these have been found of any help to find about operators suitability. Rather, the most important factor is found to be the 'maximal potential' of an operator, i.e. how fit are the best offspring it can give birth to.



(a) F1g :whole sample (b) F1g: Half sample (c) Ugly: whole sample
Figure 3: *Fitnesses of best offspring $\times$ their rank, with one crossover per parent, for uniform and one-point crossovers: The relative position of curves changes when only the top half of the sample is considered for offspring generation.*

To illustrate this, consider the plots of fitness repartition of offspring as a function of their rank according to their fitness. The case of F1g problem presented in Figure 3-a shows the best offspring of the whole sample. One cannot say which curve is better: 1C yields higher fitness offspring than UC, but UC gives a higher number of well-fitted offspring. (Remember that the all the considered measures suggest the superiority of UC, and that the runs are ordered the other way round.) But consider now Figure 3-b, which plots the same repartition of best offspring, but when considering only the best half of the parents in the sample. The curve for 1C gets largely better than UC. Figure 3-c presents the case of the ugly problem: curves of 1C and UC are similar (and so are the runs), while all the previous measures favor the UC.

This plot was found efficient in predicting the runs order for all the considered problems. The plot is almost unchanged when considered with the top half of the sample.

A crucial factor for operator comparisons seems to be their ability to generate high values, even rarely, or even for the best of parents. And this can be quite different from any mean value on the whole sample. The mean as a measure does not give any information about the most likely direction of evolution. This calls for alternative measures, as well as for on-line dynamic measurement.

4 Summary and conclusion

Some statistical measures to allow assessment of a priori suitability of a genetic operator have been presented and experimented to compare 1-point crossover and uniform crossover in the binary framework.

These experiments have first confirmed the inadequacy of fitness operator correlation, which was either unable to detect differences, or even predicting the inverse of actual results of GA. FOC only indicates whether well-fitted offspring are obtained from well-fitted parents or not. However, such information can be useful in addition to other measures.

Average improvement to parents (MI-P) can be very misleading for two reasons:

- 1- The dependency on the parents fitnesses can induce big differences (in MI-P) while the fitness distribution of offspring -independently of their parents- is the same for different operators.

- 2- It suffers from the loss of information induced by any measure based on averaging.

Those remarks address both cases of 1 and 50 crossovers per parents, which generally gave the same tendencies, the case of 50 crossovers, showing much higher differences.

When considering average improvement to target (MI-T), only the case of 50 crossovers per parents gave stable and significant differences relatively to the fitness range of each problem. MI-T does no longer suffers from the first bias above but still meets the second:

MI-T, even with 50 crossovers per parents, fails on both F1g and Ugly problems to predict the best crossover. In our opinion, this is due to the second bias of average measure. In fact, on the F1g problem, the relative position of offspring fitness distribution for UC and 1C operators changes when considering only fit parents. Moreover, the averaging does not capture any information about the fittest offspring of each operator. This information, is found to be crucial to compare crossovers.

So none of the proposed measures was found satisfactory alone, though MI-T appears the most useful – especially when B-FOC values are similar. To design

more efficient measures, we think averaging should be avoided, and best fitness offspring should be emphasized. Whatever the measure, the most important factor seems to be the way it evolves when getting to higher fitness parents: Further work will have to design online measures.

References

1. L. Altenberg. The schema theorem and Price's theorem. In L. D. Whitley and M. D. Vose, editors, *Foundations of Genetic Algorithms 3*, pages 23–49, San Mateo, CA, 1995. Morgan Kaufmann.
2. S. Baluja. An empirical comparizon of seven iterative and evolutionary function optimization heuristics. Technical Report CMU-CS-95-193, Carnegie Mellon University, 1995.
3. S. Baluja and R. Caruana. Removing the genetics from the standard genetic algorithms. In A. Prieditis and S. Russel, editors, *Proceedings of ICML 95*, pages 38–46. Morgan Kaufmann, 1995.
4. Y. Davidor. Epistasis variance: A viewpoint on representations, GA hardness, and deception. *Complex systems*, 4:369–383, 1990.
5. D. B. Fogel and A. Ghozeil. Using fitness distributions to design more efficient evolutionary computations. In T. Fukuda, editor, *Proceedings of the Third IEEE International Conference on Evolutionary Computation*, pages 11–19. IEEE, 1996.
6. D. E. Goldberg and M. Rudnick. Genetic algorithms and the variance of fitness. *Complex Systems*, 5:266–278, 1991.
7. J. J. Grefenstette. Predictive models using fitness distributions of genetic operators. In L. D. Whitley and M. D. Vose, editors, *Foundations of Genetic Algorithms 3*, pages 139–161. Morgan Kaufmann, 1995.
8. T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In L. J. Eshelman, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 184–192. Morgan Kaufmann, 1995.
9. L. Kallel and M. Schoenauer. Alternative random initialization in genetic algorithms. In Th. Bäck, editor, *Proceedings of the 7th International Conference on Genetic Algorithms*. Morgan Kaufmann, 1997. To appear.
10. B. Manderick, M. de Weger, and P. Spiessens. The genetic algorithm and the structure of the fitness landscape. In R. K. Belew and L. B. Booker, editors, *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 143–150. Morgan Kaufmann, 1991.
11. M. Mitchell and J.H. Holland. When will a genetic algorithm outperform hill-climbing ? In S. Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms*, page 647, 1993.
12. N. J. Radcliffe and P. D. Surry. Fitness variance of formae and performance prediction. In L. D. Whitley and M. D. Vose, editors, *Foundations of Genetic Algorithms 3*, pages 51–72. Morgan Kaufmann, 1995.
13. I. Rechenberg. *Evolutionstrategie: Optimierung Technischer Systeme nach Prinzipien des Biologischen Evolution*. Fromman-Holzboog Verlag, Stuttgart, 1973.
14. J. D. Schaffer, R. A. Caruana, L. Eshelman, and R. Das. A study of control parameters affecting on-line performance of genetic algorithms for function optimization. In J. D. Schaffer, editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 51–60. Morgan Kaufmann, 1989.

15. E.D. Weinberger. Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological Cybernetics*, 63:325–336, 1990.
16. D. Whitley. Fundamental principles of deception in genetic search. In G. J. E. Rawlins, editor, *Foundations of Genetic Algorithms*. Morgan Kaufmann, 1991.

The Dynamical Nightwatch's Problem Solved by the Autonomous Micro-Robot Khepera

A. Löffler*, J. Klahold, and U. Rückert

Paderborn University, Heinz Nixdorf Institute, System and Circuit Technology,
Fürstenallee 11, D-33102 Paderborn, Germany

E-mail: loeffler,klahold,rueckert@hni.uni-paderborn.de
<http://hni.uni-paderborn.de/fachgruppen/rueckert>

Abstract. In this paper, we present the implementation, both in a simulator and in a real-robot version, of an efficient solution to the so-called dynamical nightwatch's problem on the micro-robot Khepera. The problem consists mainly in exploring a previously unknown environment while detecting, registering and recognizing light sources which may dynamically be turned on and off. At the end of each round a report is requested from the robot. Therein we made use of an agent-based approach and applied a self-organizing feature map in order to refine some of the behaviour generating control-modules.

1 Introduction

1.1 Motivation

Our basic motivation to consider the implementation of behaviour generating control structures in mobile autonomous systems is represented by the following question: "What is the most complex task a robot is able to carry out, thereby coping with its limited resources, e.g. the available sensor sources, finite energy supply and limited processor power? The encountered consequences may be a restricted perception of its environment, runtime constraints and the problem of severe real-time demands." To be able to successfully envisage this kind of problem, we strongly believe in adopting the following two-step program: firstly, to implement software solutions making recourse to concepts of neural information processing; secondly, to replace single software modules by resource-efficient microelectronic components. In this framework, the contents of the present paper corresponds to the first point, i.e. the implementation of a software solution to a hard robotics' problem. Hereby, the micro-robot Khepera (see Fig.1) serves as an exemplary model for a mobile autonomous system.

Processing the sensor data of a mobile robot appropriately seems still to be a difficult and largely unsolved problem [1]. Hence, it remains an in general challenging task to consistently implement a set of behaviour generating control

* Supported by DFG-Graduiertenkolleg "Parallele Rechnernetzwerke in der Produktionstechnik", GRK 124/2-96.

modules based on the different sensor sources of the system in question. Considering the micro-robot Khepera in its basic configuration, there are three main sources of sensor data which may be used to implement such modules:

<i>source of sensor data</i>	<i>task of the respective control-module</i>
(1) the infra-red proximity sensors	→ exploration of the environment
(2) the ambient light sensors	→ detection and registration of light sources
(3) the wheel-based step counters	→ basic positioning system.

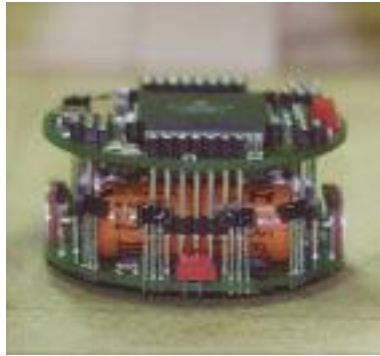


Fig. 1. Khepera is a micro-robot which has a diameter of 55mm, a height of 30mm and weighs 70g. The processor system contains a Motorola 68331 with 256Kbyte RAM and 256Kbyte ROM. Khepera perceives its environment using 8 infrared distance- and light-sensors; moreover incremental encoders placed on each motor axis are available for step counting.

In the next paragraph, we define an appropriate problem, i.e. one which includes all three of the above mentioned tasks.

1.2 The Dynamical Nightwatch's Problem

Imagine the town's nightwatch strolling over central square at late evening, looking for light sources, e.g. street lamps, lighted windows, etc., thereby avoiding obstacles, following walls (because he is a bit short-sighted) and turning round at dead ends. Having found the first light source, he fixes his report-board at the wall next to it to write down everything he will have encountered when he will arrive again at this location. In fact, he notes whether he has found a new lantern or recognized an old one and its current state (on/off) since the last stop at the board. Probably he would try to guess his position on the square by counting his steps and using already registered light sources as local landmarks to verify his counting.

The micro-robot Khepera should be able to perform an analogous task when transferred to a kind of suitable toy-world.

1.3 Design Principles

Animals may be considered as biological analogies to mobile autonomous robots. Other than the latter, they constantly had (and still have) to prove their fitness by surviving a natural selection process. Thus, it seems profitable to exploit their evolutionary optimized strategies for robot design, in this context especially those which deal with information processing. In order to do this, we adopted the following design principles inspired by concepts of neural information processing thereby making recourse to the so called agent paradigm [2–4] and some earlier works in cybernetics [5] :

- simple modules cope with simple tasks (e.g. obstacle avoidance, edge following, etc.)
- the control-modules are organized in parallel, i.e. no explicit hierarchical structure is implemented, and their results are simply superpositioned. In case of this not being possible, a data/event-driven decision is made (e.g. in dead ends, the turning-round-module takes control).
- no global goals are explicitly given to the robot
- arising complex behaviour is thus a result of the interaction of the basic modules
- adaptive algorithms, e.g. neural networks, are used to incorporate acquired informations about the environment.

2 Exploration

Enabling Khepera to explore its environment, we had to ensure free motion, i.e. avoiding obstacles, leaving substructures of the environment and not getting stuck in dead ends or similar structures. This was attained by implementing the following three basic control-modules:

2.1 Obstacle Avoidance

The obstacle avoidance was implemented as in Braitenberg’s vehicle IIIb [5] using the very simple neural concept of cross-inhibition. This means that if one of Khepera’s infra-red sensors detects an obstacle, the opposite wheel will be slowed down proportionally to the sensor input. By this way the obstacle is avoided (see Fig.2).

2.2 Edge Following

Khepera tries to keep the distance to the wall, i.e. the value of the outmost sensor, constant (see Fig.3). Note that no explicit distance to the encountered wall

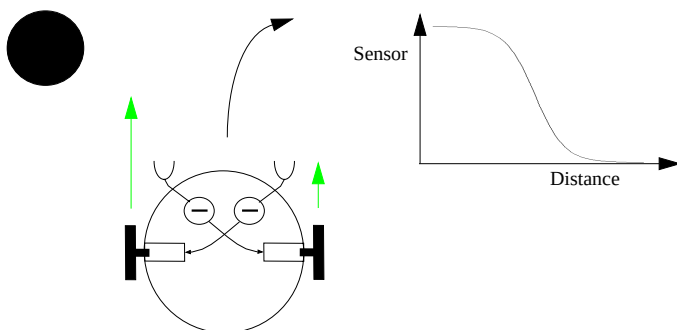


Fig. 2. A sketch of the robot avoiding a cylindrical obstacle (left) and an infra-red sensor vs. distance-to-light source characteristic are shown (right).

is be kept which allows a more flexible behaviour of the robot than otherwise be possible, especially in narrow corridors and similar structures. When a wall is detected for the first time (and avoided by means of the obstacle avoidance module), the outmost sensorvalue is registered. Afterwards, the robot will turn away from the wall, if this sensorvalue increases, and respectively will turn towards it, if it decreases. This algorithm is particularly simple, easy to implement and robust. Edge following allows to leave substructures of the environment quicker than it would be possible otherwise.

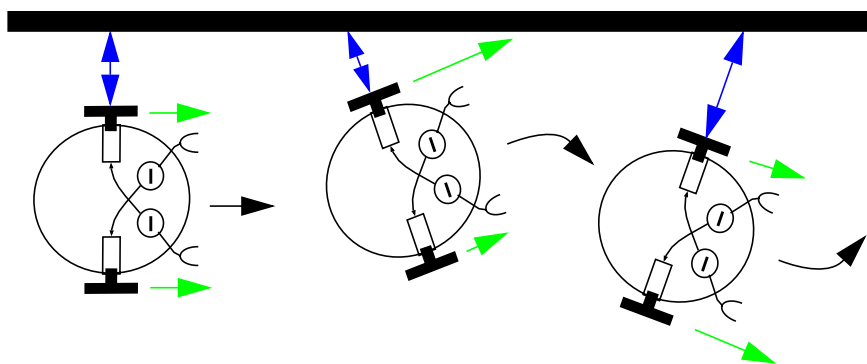


Fig. 3. Visualisation of the simple edge-following algorithm where three different situations might occur: a wall is detected (register sensorvalue_memory), turn away from wall (if sensorvalue > sensorvalue_memory), turn towards wall (if sensorvalue < sensorvalue_memory).

2.3 Turning

If a situation occurs in which Khepera is not able to leave a small area (symbolized by the circle) after some tries, it simply turns round to where it came from (see Fig.4). This module ensures that the robot does not get stuck in dead end like structures.

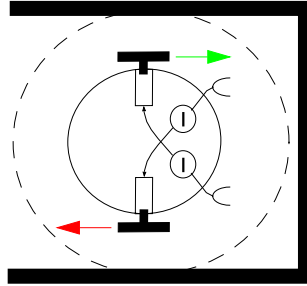


Fig. 4. The turning procedure.

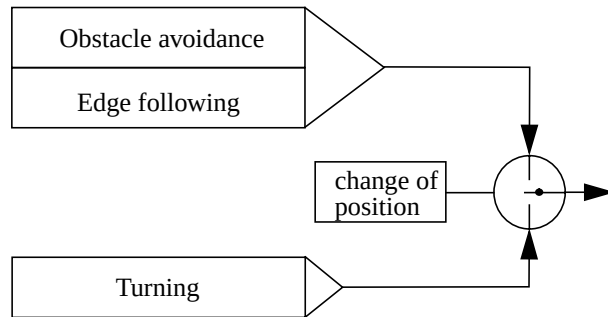


Fig. 5. The interplay of the three control modules ensuring constant exploration.

The interplay of the three modules is shown in Fig.5. A module, recognising the change of the position of Khepera, decides whether obstacle avoidance and edge following or the turning procedure get active. This means that if the robot is not able to change its position sufficiently during some 50 steps, it will turn around thereby suppressing the two other exploration modules; otherwise a superposition of the normal obstacle avoidance/edge following algorithms is used.

3 Positioning

An at least locally reliable positioning system had to be necessarily at basis of the light-source mapping process. For this purpose, we used the step counting functionality of Khepera. The position vector of Khepera consists of an x -value, y -value and the direction-indicating angel α . The new position (x, y, α) is calculated from the old one (x_0, y_0, α_0) using the values n_r and n_l given by the incremental encoders placed on each motor axis. The table below shows how the new position is calculated:

Table 1. Calculation of the change of position

				
$ n_l = n_r $	n_r S	n_r S	$n_r - n_l$ T	$n_r - n_l$ T
$ n_l < n_r $	n_l S $n_r - n_l$ C	n_l S $n_r - n_l$ C	$-2 * n_l$ T $n_r + n_l$ C	$-2 * n_l$ T $n_r + n_l$ B
$ n_l > n_r $	n_r S $n_r - n_l$ C	n_r S $n_r - n_l$ C	$2 * n_r$ T $-(n_r + n_l)$ C	$2 * n_r$ T $-(n_r + n_l)$ C

Straight:

$$x = x_0 + n \cdot \Delta l \cdot \cos \alpha$$

$$y = y_0 + n \cdot \Delta l \cdot \sin \alpha$$

Turn:

$$\alpha = \alpha_0 + \frac{n}{2 \cdot r} \cdot \Delta l$$

Curve:

$$\alpha = \alpha_0 + \frac{n}{2 \cdot r} \cdot \Delta l$$

$$x = x_0 + r \cdot (\cos \alpha - \cos \alpha_0)$$

$$y = y_0 + r \cdot (\sin \alpha - \sin \alpha_0)$$

4 Sensor Calibration

Adapting an one-dimensional self-organizing feature map [7, 8] by training with a real robot data set, we managed to calibrate the ambient light-sensors (see Fig.6), i.e. to transform the coarse grained information of Khepera's angle-to-light-source into a fine grained one; this also helped to overcome sensor-response variations due to fabrication tolerances. Calibration of the ambient light sensors means to classify the sensor vectors by an one-dimensional Kohonen feature map. Each neuron is active for a certain class of sensor vectors giving a direct information about Khepera's angle-to-light source.

The charts (Fig.7) show the sensor data, respectively used to learn and to test the neural network. Before usage, the data were standardized by a scaling process, which increased the accuracy considerably.

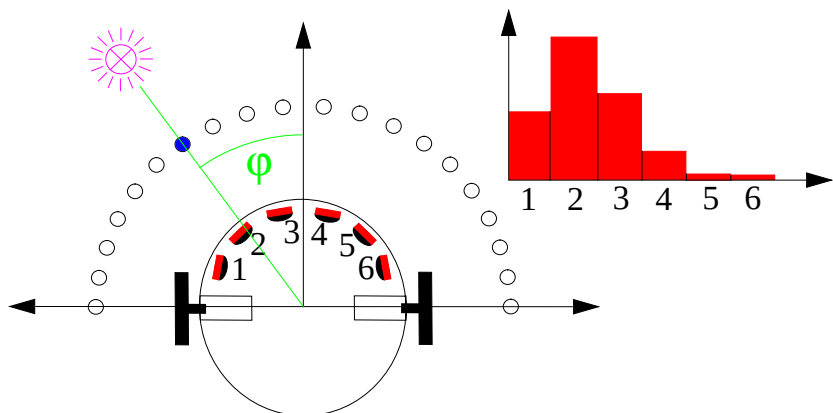


Fig. 6. The picture on the left shows that every neuron belongs to a specified angel. Each neuron is trained to react on a special characteristic of sensor values (e.g. the neuron represented by the filled circle reacts on the sensor vector given in the bar chart on the right).

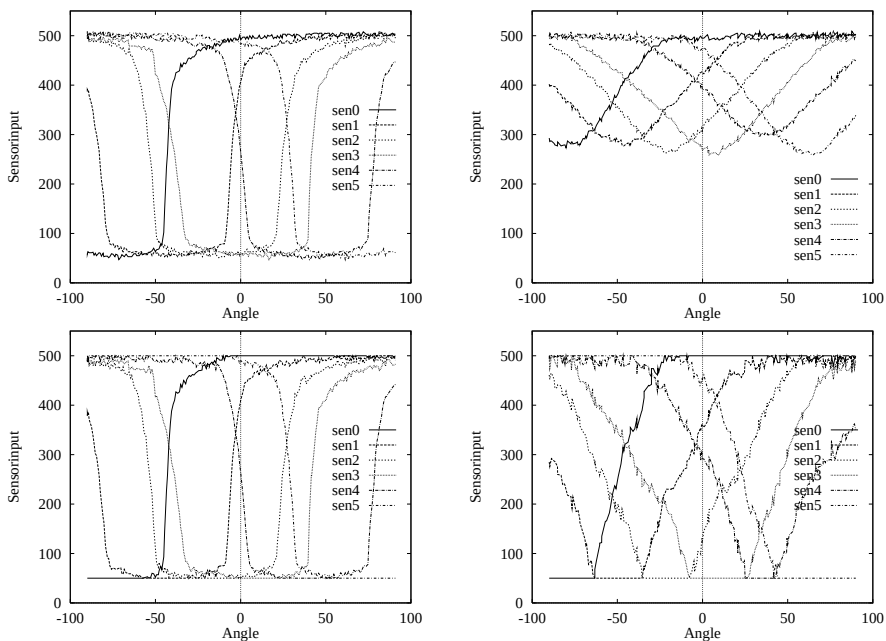


Fig. 7. The sensor data sets used to train and test the neural net; upper left: learning data, upper right: test data, lower left: scaled learning data, lower right: scaled test data.

The network consists of 60 neurons. It was trained in 20 cycles with an incremental change of the neighbourhood-width from 10 to 1 and a learning rate from 0.6 to 0.1. The result was an average error of 1.52 and a maximum one of 6.66 degrees at the recall process. The test at double distance caused an average error of 6.52 and a maximum one of 21.89 degrees. The results are graphically depicted in Fig.8.

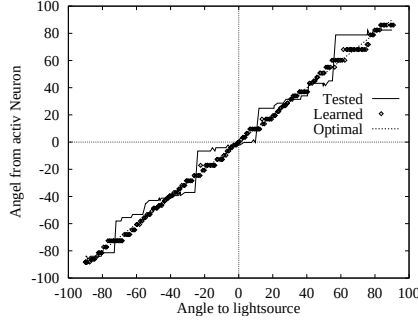


Fig. 8. The result of the adaptation process is shown in this chart. The dotted line serves as a referential, the bold dots correspond to the data being learned and the strait line belongs to data registered at the double distance with respect to the learning set.

5 Mapping Procedure

The procedure of the position-determination of detected light-sources was implemented as follows (see Fig.9). If the robot detects a light-source (1), Khepera approaches it (2) until one sensor value exceeds a certain threshold (optimal range for the neural network is reached) (3). Then Khepera registers its position vector (x_1, y_1, α_1) , turns by 90 degrees and moves in a strait line (4) to a second turning point (5). Afterwards, the robot reapproaches the light-source (6) until the optimal range is reached again (7). With the new position vector (x_2, y_2, α_2) and the old one, the location of the target can be calculated.

In this section, the conditions for changing the state of a light source are explained (see Fig.10). If Khepera penetrates the inner circle (in the Manhattan distance) of a previously discovered light source without detecting it, the state of the light source will be changed from on to off. If a “new” light source is registered within the outer circle of a previously discovered one, they are recognized as being identical. Eventually, the internal position of the robot is readjusted by a comparison of the previously and the currently registered location of the light source.

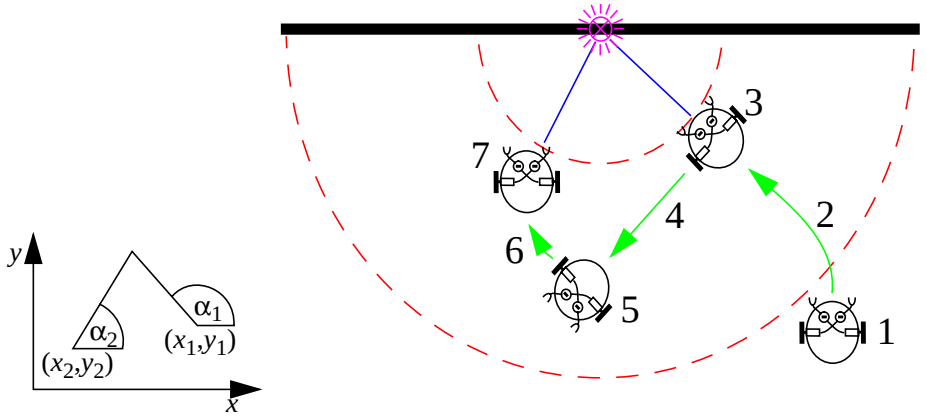


Fig. 9. A schematic view of the successive steps of the mapping procedure (right) and a sketch of the triangulation process using the registered data (left), hence determining the detected light source's location.

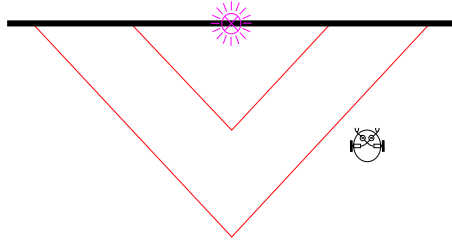


Fig. 10. Proportional view of Khepera and the two circles (in the Manhattan distance) relevant for changing the state of a previously discovered light source.

6 Report

We developed a simple code to enable the robot to report its findings during the last round using the two LEDs of Khepera.

The signals have the following meaning (legend: ● on, ○ off):

- ● New light source found
- ○ Previously discovered light source recognized
- ● Previously discovered light source turned off
- ● Previously discovered light source turned on again

7 Results

This paragraph is dedicated to the obtained results for both the simulator solution to the dynamical nightwatch's problem and the real-robot implementation (see Fig.11, Fig.12) using the obtained paths of the robot in the respective environments as an evaluation benchmark. We state a very satisfactory behaviour of the simulator solution whereas the real-robot version unveils to be feasible only locally. Note that the compiled C-code of the whole program has 87Kbyte, hence we are using only about one third of the RAM capacities of Khepera's processor.

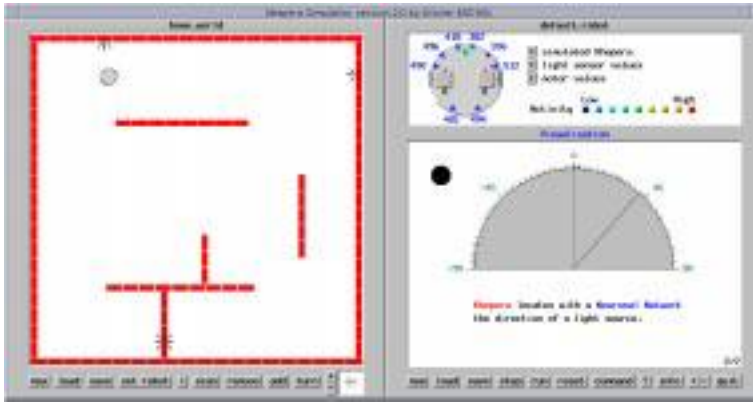


Fig. 11. In the visualisation window of the Khepera simulator [6], the robot's current angle-to-light-source is depicted.

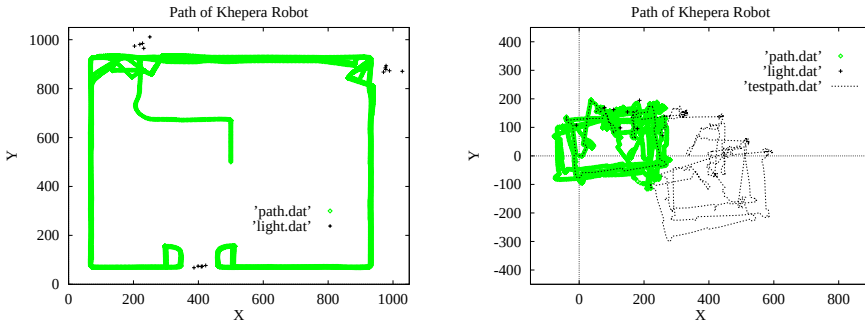


Fig. 12. The left gnuplot shows the path of the simulated Khepera. Note that the small location errors are solely caused by the remaining inaccuracies of the neural network. The right gnuplot shows the path of a real Khepera wherein the bold line represents the path with and the dotted line one without readjustment of the position using the already discovered light source as a landmark.

8 Conclusion and Future Work

Inspired by concepts of neural information processing and the agent paradigm, we implemented a solution to the dynamical nightwatch's problem, both on the simulator and on the real Khepera. In particular, we used an one-dimensional self-organizing feature map for sensor calibration attaining maximum accuracy in average. The simulator solution proofed to be globally reliable, whereas the real-robot implementation had some short-comings due to step-counting errors inducing inaccuracies into the position determination procedure. This was partly cured by taking into account the already registered light sources as local landmarks in order to readjust the current position. The results showed to be satisfactory locally, i.e. when either the robot's environment was restricted to a certain area or the runtime was limited.

Nevertheless, some further research on the topic of navigation is to be done. We hope to enhance the robot's performance considerably by incorporating associative memory and neural classifiers into the control structure. Moreover, it is also envisaged to replace some of the software modules by resource-efficient microelectronic devices as described for example in [9, 10].

References

1. S. Geva, J. Sitte and H. Sira-Ramirez, When are tasks "difficult" for learning controllers?, 1994 World Congress on Computational Intelligence WCCI, July 1994, Orlando, Florida, USA. Proceedings of the IEEE International Conference on Neural Networks, Volume IV, pp.2419-2423
2. P. Maes, Modeling Adaptive Autonomous Agents, Artificial Life Journal, edited by C. Langton, Vol 1, No. 1 & 2, MIT-Press, 1994
3. R. A. Brooks, Intelligence without Representation, Artificial Intelligence, 47, pp. 139-159, 1987
4. R. A. Brooks, Intelligence without Reason, Computers and Thought lecture, Proceedings of IJCAI 91, Sydney, Australia, 1991
5. V. Braitenberg, Vehicles: Experiments in Synthetic Psychology, MIT Press/Bradford Books, 1984
6. O. Michel, Khepera Simulator, version 2.0, 1996,
<http://diwww.epfl.ch/lami/team/michel/khep-sim/>
7. K. Malmstrom, L. Munday, J. Sitte, A Simple Robust Robotic Vision System using Kohonen Feature Mapping, Proceedings of the 2nd IEEE Australia and New Zealand Conference on Intelligent Information Systems, pp. 135-139, 1994
8. T. Kohonen, Self-Organization and Associative Memory, Springer-Verlag, 2nd edition, 1988
9. A. Heitmann, J. Malin, C. Pintaske, U. Rückert, Digital VLSI Implementation of a Neural Associative Memory, 6th International Conference on Microelectronics for Neural Networks, Evolutionary & Fuzzy Systems, 24.-26. September, Dresden, Germany, 1997
10. S. Rüping, M. Porrmann, U. Rückert, SOM Hardware-Accelerator, WSOM'97: Workshop on Self-Organizing Maps, June 4-6, pp. 136-141, Espoo, Finland, 1997

CoDi-1Bit : A Simplified Cellular Automata Based Neuron Model

Felix Gers¹ Hugo de Garis¹ Michael Korkin²

¹ATR, Human Information Processing Laboratories
2-2 Hikari-dai, Seika-cho, Soraku-gun, Kyoto 619-02, Japan.
tel. +81-774-95-1079 , fax. +81-774-95-1008 , flx@hip.atr.co.jp &
degaris@hip.atr.co.jp, <http://www.hip.atr.co.jp/~flx>, &
<http://www.hip.atr.co.jp/~degaris>

² Genobyte Inc. 1319 Spruce St., Suite 210, Boulder, CO, 80302, USA.
tel. + 1 303 545 6790, fax. + 1 303 545 9667, korkin@genobyte.com,
<http://www.genobyte.com>

Abstract. This paper presents some simplifications to our recently introduced “CoDi-model”, which we use to evolve Cellular Automata based neural network modules for ATR’s artificial brain project “CAM-Brain” [11]. The great advantage of CAs as a modeling medium, is their parallelism, which permits neural system simulation hardware based on CoDi to be scaled up without loss of speed. Simulation speed is crucial for systems using “evolutionary engineering” technologies, such as ATR’s CAM-Brain Project, which aims to build/grow/evolve a billion neuron artificial brain. The improvements in the CoDi model simplify it sufficiently, so that it can be implemented in state of the art FPGAs (e.g. Xilinx’s XC6264 chips). ATR is building an FPGA based Cellular Automata Machine “CAM-Brain Machine (CBM)” [13], which includes circuits for neural module evolution and will simulate CoDi about 500 times faster than MIT’s Cellular Automata Machine CAM-8 currently used at ATR.

Keywords

Cellular Automata, Evolutionary Engineering, Evolvable Hardware, Neural Networks, Genetic Algorithms, Genetic encoding, Artificial Brains, Cellular Automata Machine (CAM-8), CAM-Brain Machine (CBM).

1 Introduction

ATR’s CAM-Brain Project aims to build/grow/evolve brain-like-systems, into which human knowledge can be inserted as initial conditions. The brain-like structures are based on cellular automata (CA) which grow inside cellular automata machines (CAMs). Evolutionary engineering techniques, combined with huge computing power, may then be able to evolve such brain-like systems with a level of functionality similar to that of biological brains. The expectation is that

it will be much easier to investigate the behavior of engineering-based brains than biological brains, and that the engineering solutions will reveal insights into the working mechanisms of the evolved systems, which might then be rediscovered in biological brains.

In the CAM-Brain Project, the CAs act as the medium in which to conduct the growth, and in a second step, the neural signaling, of brain-like structures.

To apply evolutionary engineering techniques to the creation of huge neural systems, e.g. an artificial insect brain, we need a simulation tool that has to satisfy two conditions. First of all, it must be possible to simulate most of the important characteristics of biological neural networks. Secondly, the evolution times must be manageably short. The bottle-neck in evolutionary algorithms is the fitness measurement time, i.e. the time needed to simulate the systems. Hence, in order to develop what would presently be considered large-scale systems, on the order of a billion neurons, we believe that scalability by massive parallelism is essential.

With CAs as a simulating tool, the simultaneous demands of generality and scalability can be met. Artificial neurons and neural networks can be modeled at any level of detail, because CA cells can symbolize a molecule in a biological cell membrane, a whole neuron, or any structure in between. Since CAs are only locally connected, they are ideal for implementation on purely parallel hardware. CAs are both structurally flexible and inherently parallel.

However, for parallelization to be advantageous, it is essential that the rules for the underlying CA be simple enough to be stored locally. Indeed, Our CoDi CA based model [11] first grows and then executes neural structures by means of a set of rules small enough to be embodied in large-scale parallel hardware by means of reconfigurable VLSI circuits – Field Programmable Gate Arrays (FPGA's). For simulation purposes, we have executed the model on MIT's Cellular Automata Machine CAM-8 with a 16 bit look-up-table.

This paper summarizes the main ideas and features of the CoDi model, and introduces some simplifications to it that lead to the "CoDi-1Bit" model and its implementation in ATR's FPGA based CAM-Brain Machine (CBM) [13]. The CBM, which is a Xilinx XC6264 chip based hardware device which can evolve 3D CoDi based neural net modules including a complete Genetic Algorithm.

The remainder of this paper consists of the following. Section 2 describes our CoDi-1Bit cellular automaton model's growth and neural signalling phases. Section 3 presents results from some experiments in which simulations of small-scale structures were effected on the CAM8 machine. Section 4 describes some features of the CAM-Brain Machine (CBM), the FPGA machine which at time of writing is under construction. Section 5 discusses the results achieved, and previews the main orientations of our future research.

2 General Description of the CoDi CA

When designing our CA-based neural networks using the new "CoDi-1Bit" model, our objective was to implement them directly in evolvable hardware, e.g. FP-

GAs. Therefore, we tried to keep the CAs as simple as possible, by having a small number of bits to specify the state, keeping the CA rules few in number, and having few cellular neighbours. At this stage, no learning algorithm has been included into the CA, as we rely on evolutionary learning based on a Genetic Algorithm (GA).

We use a von Neumann neighbourhood [2], i.e. a cell in a 3D space looks at its 6 nearest neighbors and at its own state. It should be possible to extend our model to more than 3 dimensions, since the general approach taken is independent of the dimensionality of the CA-space.

We use two different CAs for the growth phase and the signaling phase in our neural networks. The cell interactions are similar in both. The main difference between the two phases, and hence the two CAs, is that the type of a cell can only change during the growth phase, e.g. a blank cell can change to an axon or a dendrite cell. (See later sections for details.) Cell types do not change during the signaling phase, although cells do change their “activity values” and exchange neural signals. (See later sections for details).

Since cell-to-cell interactions are similar in the growth and signalling phases, they can be incorporated into the same CA with a dynamic “phase switch”. This might be useful when a future learning algorithm needs to grow new connections, depending on the activity in already existing axonal or dendritic connections. In the following three subsections (2.1- 2.3) we describe how the CoDi CA can be implemented as a partitioned CA, the CoDi cell-to-cell interaction model itself and the genetic coding, i.e. the initialization, of the CoDi CA.

2.1 Implementation of CoDi as a Partitioned CA

The states of our CAs have two parts, which are treated in different ways. The first part of the cell-state contains the cell’s type and activity level and the second part serves as an interface to the cell’s neighbourhood by containing the input signals from the neighbors. Characteristic of our CA is that only part of the state of a cell is passed to its neighbours, namely the signal and then only to those neighbours specified in the fixed part of the cell state. This CA is called “partitioned” [11], because the state is partitioned into two parts, the first being fixed and the second is variable for each cell (see figure 1).

The advantage of this partitioning-technique is that the amount of information that defines the new state of a CA cell is kept to a minimum, due to its avoidance of redundant information exchange.

For a CA implementation on a look-up-table (LUT) machine such as MIT’s CAM-8 or in FPGAs, it is crucial to minimize the amount of cell-to-cell communication, to keep the number of states in the CA manageable.

This is a major advantage for the implementation of CAs in parallel hardware. Large LUTs or complex programs cannot be stored locally, and hence are non-parallel. Using the partitioning-technique, improves the practicability of a neural network simulated directly in parallel hardware.

The next few subsections detail the new CoDi model and its implementation as a partitioned-CA.

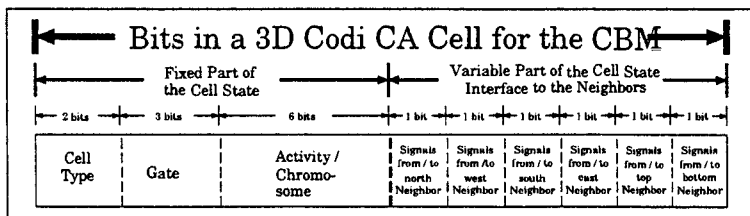


Fig. 1. State representation in the CoDi-1Bit model. During the growth phase 6 of the bits are used to store the chromosome’s growth instructions. The same 6 bits are later used to store the activity of a neuron cell during the signaling phase.

2.2 CoDi Cell Interaction

The CoDi model is named after the essential features of its axonal and dendritic cell behavior, i.e. to “collect” or to “distribute” the cellular values from/to the CA cells’ neighbors. The CoDi model works with four basic types of CA cells, neuron body, axon, dendrite and blank.

- Blank cells represent empty space. They do not participate in any cell interaction during the signaling of the neural network.
- Neuron bodies consist of one CA cell. The neuron body cells collect neural signals from the surrounding dendritic cells and apply an internally defined function to the collected data. In the CoDi model the neurons sum the incoming signal values and fire after a threshold is reached. This behavior of the neuron bodies can be modified easily to suit a given problem. The output of the neuron bodies is passed on to its surrounding axon cells.
- Axonal cells distribute data originating from the neuron body.
- Dendritic cells collect data and eventually pass it to the neuron body.

These types of cell-to-cell interaction cover all kinds of cell encounters. Axonal and dendritic cells do not need to know anything about the cell type of their neighbors. They behave in the same way towards any type of neighbor. Therefore it is not necessary to distinguish between the different cases of cell-to-cell interactions, e.g. axon-neuron, axon-dendrite, dendrite-dendrite etc, so that designing separate rules for each case is unnecessary. The two basic interactions cover every case, and they can be expressed simply, using a small number of rules.

To make axonal and dendritic signal trails (which can be dense in the CA-space) without blockers or sheath cells as in earlier models [4], the four basic cell types need further specification. To the 2 bits which specify the basic cell type, we added 3 more bits to set the “gate” of a 3D CA cell. (See figure 1.) A neuron cell uses this gate to store its orientation, i.e. the direction in which the axon is pointing.

In an axon cell, the gate points to the neighbor from which the neural signals are received. An axon cell accepts input only from this neighbor, but makes its own output available to all its neighbors. In this way axon cells distribute information. The source of information is always a neuron cell. Dendritic cells collect information by accepting information from any neighbor. They give their output, (e.g. a Boolean OR operation on the binary inputs) only to the neighbor specified by their own gate. In this way, dendritic cells collect and “sum” neural signals, until the final sum of collected neural signals reaches the neuron cell.

We will see that each axonal and dendritic cell “belongs” to exactly one neuron cell. This configuration of the CA-space is guaranteed by the preceding growth phase (section 2.3). Figure 2 shows the signaling phase with the CoDi-model. Each neuron is given two dendritic trees and two axonal trees. One of the axonal trees distributes inhibitory signals of the neuron, the other distributes excitatory signals. A neuron has two dendritic trees to maximize the amount of information that can be passed to the neuron body in each time step.

We use binary neural signals which can fill a trail contiguously. The direction of motion is given by the underlying signal trails.

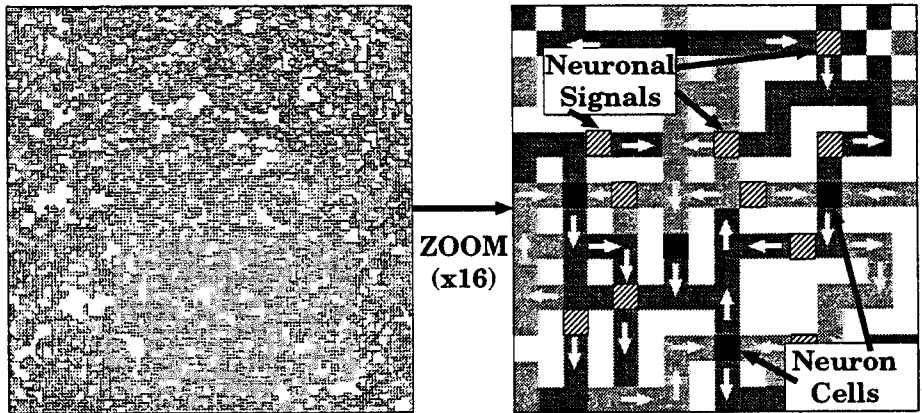


Fig. 2. (Left) A CA-space with $256 \cdot 256$ cells in the signaling phase with the CoDi-model. (Right) A zoom (x16) with five neuron cells (black) with two dendrites and two axons each. The arrows inside the axonal (dark grey) signal trails and dendritic (light grey) signal trails indicate the direction of information flow during the signaling phase. In the growth phase the signals in the dendritic trails travel the other way, because they are emitted by the neuron. To improve the visualization of neural trails, the underlying chromosome is gridded with a checker-board pattern of “grow-straight” instructions. This causes the neural trails to grow on a 2-cell grid.

Synapses The CoDi model does not use explicit synapses, because dendrite cells that are in contact with an axonal trail (i.e. have an axon cell as neighbor) collect the neural signals directly from the axonal trail. This results from the behavior of axon cells, which distribute to every neighbor, and from the behavior of the dendrite cells, which collect from any neighbor.

However axon-axon and dendrite-dendrite interactions of cells that do not belong to the same neuron are not possible. This is because axon cells take their information only from the neighboring axon cell that is closer to the “source” neuron on their axonal trail. Dendritic cells behave in the opposite way, by passing signals to the dendritic neighbor that is closer to the neuron or to the neuron cell itself. This configuration of the CA-space results from the neural growth phase.

These two simple behaviors of axon and dendrite cells (i.e. collect and distribute), which are exact opposites, allow axonal and dendritic trails to be dense in the CA-space, without interacting in an unintended way. The CoDi-model remains functional even if every cell in the CA-space is used.

The strength of a neuron-neuron connection (a synapse) is represented by the number of their neighbouring axon and dendrite cells. The exact structure of the trails and the position of the axon-dendrite neighbor pairs determine the time delay and strength (weight) of a neuron-neuron connection. This principle implies that a single neuron-neuron connection can consist of several synapses with different time delays with independent weights.

2.3 Genetic Encoding, Initialization and Growth of the CoDi CA

Biological Systems have to self-assemble. The genetic encoding of biological brains is indirect, sophisticated and incomplete. There is not enough genetic information in vertebrates to encode the detailed connectivity of their brain circuits.

However, these characteristics of biological brain growth need not apply to electronic brains. Electronic circuits do not have to grow. They can be copied into hardware as initial settings. The genetic encoding can be direct. For example, a neural circuit can be stored as a spatial pattern in bitmap format. The genetic information can be complete, so that a given chromosome always produces exactly the same neural circuit and vice versa.

For the CoDi model, we have kept the biological feature of growing a system. We chose a genetic encoding that is indirect, but very different from biological genetic encoding. We wanted our encoding to be one to one, so that a given chromosome leads to exactly one neural network. In this way we ensure that a chromosome has a unique fitness value and hence do not have to spend time calculating an average fitness for each chromosome.

The advantage of a grown system is that the structural integrity of the basic components (in our case the neurons) is preserved in the presence of mutations (or alternations in the chromosome). All non-blank cells in the CA-space can be assigned to exactly one neuron, (the neuron from which they were grown). This is the case for any chromosome. Though the connectivity of the neural network

may be unsuitable for a given problem after the growth according to a arbitrary chromosome, a consistent structure of the neural network is always guaranteed. The importance of this feature becomes apparent when we compare a) the effect of genetic operators on a neural system that grows and b) neural systems that are mapped directly to their chromosomes. In the first case a mutation leads to a small, local change in the direction of a signal trail. In the case of direct mapping, a mutation and the resulting local change of a cell, which may belong to a signal trail of a complex axonal tree, can lead to disconnection of a large amount of circuitry. This circuitry would then be disconnected from its source neuron, and loose the integration into the basic structure of the neural network model. The situation gets worse, if we look at the way new neural connections come into being. The probability that a new and useful signal trail evolves if all cells in the CA-space are mutated independently (as is the case for direct mapping), are very small. However a mutation in a grown neural network (for example from “turn right” to “split right”) leads to the growth of a new neural signal trail that is consistently integrated into the structure of the source neuron and hence the whole network. This feature of a grown system reduces the search space for possible CA-patterns to the patterns that represent consistent neural structures. The selection of an indirect genetic encoding is a logical consequence of the decision to grow our networks.

The CoDi model uses a distributing chromosome to encode its structure [11]. The chromosome is initially distributed throughout the CA-space, so that every cell in the CA-space contains one instruction of the chromosome, i.e. one growth instruction, so that the chromosome belongs to the network as a whole. This technique has several advantages compared to previous approaches [4].

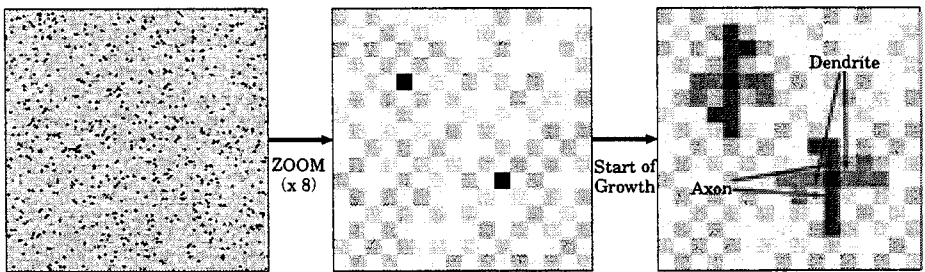


Fig. 3. (Left) An initial pattern of $128 \cdot 128$ 2D CA cells with a neuron-density of 5%. The (black) neuron cells are randomly positioned. (Middle) A zoom (x8) with two neuron cells on a random (2-cell gridded) chromosome. (Right) The beginning of the growth phase, after three CA-steps.

The neuron bodies can be positioned arbitrarily in the CA-space. (See figure 3.) Growth signals are passed to the direct neighbors of the neuron cell according to its chromosome information. The blank neighbors, which receive a neural

growth signal, turn into either an axon cell or a dendrite cell. (See figure 3 (right).) In the earlier CoDi model [11] the growth signals (and signals in general were 2 bits wide), so that they included information containing the cell type of the cell that was to be grown from the signal. To reduce the Codi model to one bit signals we had to find a different way to pass this information. The solution is to work with an oscillating CA, i.e. the growth signals are time multiplexed. In a given time-step only axons pass their growth signals and all blank cells that receive signals interpret them as “grow an axon” signals, whereas in the next time-step dendrites grow. Different ratios of axon to dendrite time-steps are possible, e.g. 3 axonal time-steps on one dendritic time-step.

The cells just grown, then receive the next growth signal from the neuron body cell, which sends them continuously. To decide in which directions axonal or dendritic trails should grow, the grown cells consult their chromosome information which encodes the growth instructions. These growth instructions can have an absolute or a relative directional encoding. An absolute encoding masks the 6 neighbors (e.g. directions) of a 3D cell with 6 bits. The cell contiguous to the face (direction) whose bit is set, will itself become an axon cell or dendrite cell at the next clock tick. For a relative encoding, 5 bits are sufficient, as the direction from which the initial growth signal comes is known to a cell (be stored in its gate field). The growth information can be interpreted relative to this direction, so one bit less information is needed. The absolute encoding has the advantage of being easier to formulate in logic equations and we use it in the FPGA design of the CBM (see section 4). A relative encoding saves one bit in the cell state, which can be crucial for LUT machines. To avoid a 17 bit LUT we used the relative encoding on the CAM-8 (which offers a 16 bit LUT). We will use the relative encoding to illustrate the growth phase with a distributed chromosome.

5 bits of chromosome information in each cell offer $2^5 = 32$ different growth instructions, which are : (“block growth”, 5 simple turns, 10 simple splits, 10 3-direction splits, 5 4-direction splits and “grow in all directions”), see figure 4). Using its chromosome instructions and knowing from which direction the growth signals come, a cell can determine the neighbors to which it transmits the incoming growth signals. (See RHS of figure 4.) If, for example, a growth signal comes from the left neighbor to a cell whose chromosome instruction is “split right”, it will pass the growth signal to its right and bottom neighbors. After a cell is grown, it accepts growth signals only from the direction from which it received its first signal. This “reception direction” information is stored in the “gate” position of each cell’s state.

Figure 5 shows two cut-outs of neural networks at the end of a growth phase. The growth phase stops when growth is saturated, i.e. when no more trails (as specified by the instructions of the network’s chromosome) can be formed.

In most of our figures, we have used gridded neural structures, in order to render the axonal and dendritic trails more visible. To force neuron trails to grow on a 2-cell grid, the underlying chromosome is initialized in a “checker-board” pattern. The “black” squares of the checker-board are filled with “grow-straight”

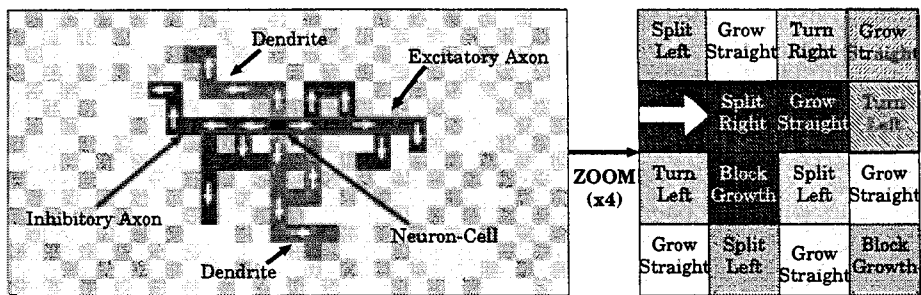


Fig. 4. (Left) A neuron in the CoDi-model with two dendrites and two axons. The arrows inside the axonal and dendritic signal trails indicate the direction of information flow during the growth phase. The neuron grows on a random (2-cell gridded) chromosome in the CA-space. (Right) A signal trail grows according to the underlying chromosome instructions. The trail continues to grow over the two striped cells in the next two time steps. The branch of the trail that leads downward is blocked by the blocker instruction in the chromosome.

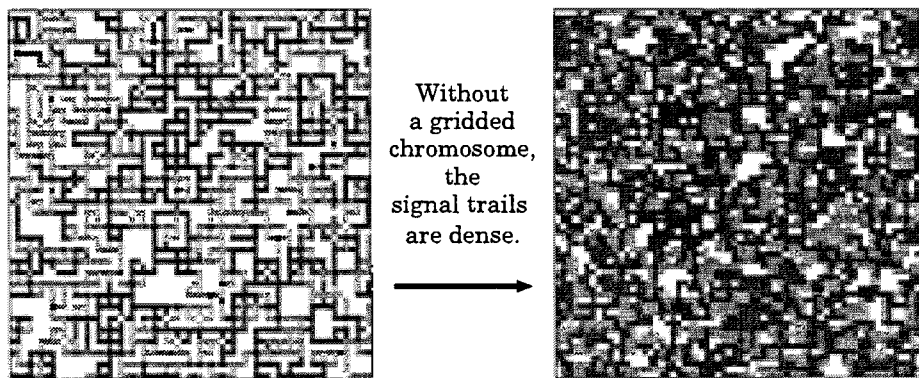


Fig. 5. Two cut-outs of neural networks at the end of the growth phase. (Left) A network grown from a gridded chromosome. (Right) A network grown without gridded chromosome instructions.

instructions and the white squares are filled with arbitrary growth instructions. Since gridding is only a visualization trick, an unconstrained evolutionary algorithm can grow trails which are dense (the RHS of figure 5 shows an example of an ungridded neural network).

Beside the “turns”, “splits” etc. instructions, there are “block-growth” instructions in the chromosome. They cause a cell in an axonal or dendritic trail to block the passage of the incoming growth signals to its neighbors, so the

branch of the trail stops growing. With a high density of blocker instructions in a chromosome, an area in the CA-space can be transformed into a more or less impenetrable “wall”. Figure 6 shows two “walls” with different densities of

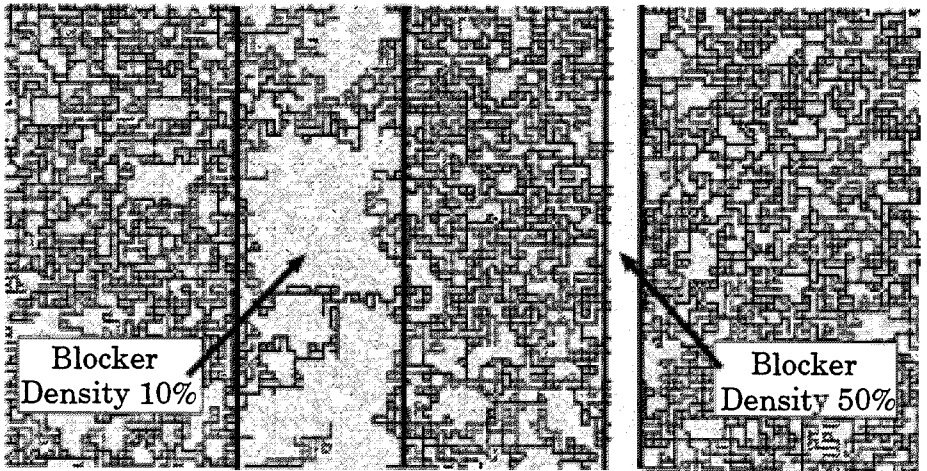


Fig. 6. The use of “walls” to modularize neural circuits in the new CAM-Brain model. The density of blocker instructions on the chromosome determines the probability of trail-growth through a wall and hence the penetrability of the wall.

blocker instructions in the two regions. A combination of walls can partition a neural network into more or less independent modules.

The grown circuitry and the chromosome are connected locally, so that a local learning algorithm can make local changes in the phenotype and the genotype of the network. This enables the use of Lamarckian evolution (with local learning algorithms) in addition to Darwinian evolution.

The distributed chromosome technique of the new CAM-Brain model makes maximum use of the available CA-space and enables the growth of any type of network connectivity. The local connection of the grown circuitry to its chromosome, allows local learning to be combined with the evolution of grown neural networks.

2.4 CoDi on the CAM-Brain Machine (CBM)

Why is the simplicity of our CA-model important? Our long term aim is to build CAs to model brain-like structures. Biological brains, the highly optimized results of millions of years of evolution, are obvious sources of inspiration for our work.

Even the tiny brains of insects consist of about a million neurons with complex connectivity. In order to model a 3D neuron with CAs in a way that preserves the essential functionality of real neurons, we estimate that we need at least a cube of 10^3 CA cells. So the minimum size of CA-space to model an insect brain is 10^9 cells. To update a billion CA cells at acceptable speeds will require the use of parallel hardware. The ideal case for CA updating is to give each CA cell its own processor, so that the whole CA-space is updated, in parallel, in one clock cycle.

To build parallel hardware with 10^9 independent CA processors (for a small mammalian brain it would be 10^{12} CA processors), the components must be very simple in their internal logic, with very little memory. Large look-up-tables (LUTs), as are common in traditional CA implementations, will be impossible. Hence, for a CA to be executable on parallel hardware, the rules must be few enough to be expressible within the limited number of logic gates per CA processor inside the parallel hardware. For this reason we tried to keep our new CAM-brain model as simple as possible.

With the CoDi-1Bit model, illustrated in this paper, it is possible to implement one CA cell with about 150 Xilinx XC6264 "functional units". This inspired us to build our CAM-Brain Machine (CBM)[13].

3 Evolvability Simulation Results

This section presents some results which were undertaken to show that the CoDi-1Bit model is actually evolvable. It also briefly discusses representation issues.

Three experiments were performed to show that it would be worthwhile to invest in the financial cost to build a fully hardware implemented model of the CoDi-1Bit design in the CAM-Brain Machine (CBM). Note that the emphasis here is NOT on the intrinsic interest of the evolved functionality, but rather the evolvability of the model.

3.1 The Comparator

The first software simulation experiment (on a Sun Sparc 10 workstation) aimed to evolve a binary comparator, i.e. two 3 bit numbers were input at the face of a cube of $16*16*16$ CA cells. At the opposite face was an output whose binary value should be 1 if the input signal A was greater or equal to the input signal B, otherwise 0. The signal phase was run over 50 clockcycles, with only the output from the final 20 clocks being used in the fitness definition (to allow time for the signal to circulate through the circuit). If the desired output actually occurred, the fitness value incremented a point. The total points over all 64 cases (8 cases of A, 8 cases of B) were divided by $64*20$. The population size was 10. The chromosome length was about 20K bits. The fitness rose to about 90 percent after about a day's run, and then very slowly increased, but our human patience ran out. We were not prepared to run the experiment for days. We came to the conclusion *that to truly test the evolvability of the CBM, we would need a CBM.*

3.2 The Timer

The next two experiments were "single case" types, so that the fitness measurement time would be within human patience limits. The second experiment was to evolve a timer, i.e. the output signal should be 0 for the first 40 clocks (of 100 total), the next 30 clocks should have a 1 putput, and the remaining 30 clocks should be 0. The inputs were the same 6 inputs as before, but all set at 1. The same output point was used. The $16 \times 16 \times 16$ cube was wrapped around on the side edges, but not at the top and bottom. The fitness definition was chosen so that if a desired 1 occurred, the fitness value incremented 5 points, otherwise, incremented zero points. If a desired 0 occurred, the fitness incremented by 1 point. The total over the 100 clocks was divided by $((40+30)*1)+(30*5) = 220$. After about 100 generations, the desired output evolved perfectly, i.e. fitness was 100 percent. This first success showed that the CoDi-1Bit model could do at least this much.

3.3 The Full Sinus Wave

The third and final experiment undertaken was to evolve a sinusoid output. The inputs were the same as in the second experiment. There was a 2 bit output with the following representation. If the first bit B1 was a 1 and the second bit B2 was a 0, then a fictitious counter was incremented by 1. If B1=0 and B2=1, then the counter was decremented by 1. Other cases left the counter unchanged. An arbitrary sinusoid amplitude and wavelength target shape was chosen, that the counter is supposed to follow as closely as possible, with the counter value being the vertical "y" axis, and the clock count being the horizontal "x" axis. The circuit should evolve so that the outputs B1 and B2 change over time, such that the counter gives the appropriate "y" values over time.

The fitness value was defined to be the inverse of the sum of the squares of the differences between the desired y values and the actual counter values. An incremental or stepwise evolutionary approach was taken that de Garis developed in 1990 [de Garis 1990], namely that the evolution used several intermediate fitness definitions, where the resulting population of GA chromosomes evolved with fitness definition FD1, became the starting generation of chromosomes with fitness definition FD2, etc. Often this approach gives better results than using only one fitness definition. In the case of the sinusoid, at first a quarter wavelength was evolved (where the target sinusoid had an amplitude of 20 and wavelength of 200 and the whole curve shifted up by 30. The fitness measurement started after 20 clocks, at which point, the counter was set initially at the value 30. After about 100 generations, the evolved output followed the desired target within a few percent. The resulting population became the starting population for a second evolution phase where the aim was to evolve a half sinusoid. Again, after a day, and several hundred generations, the actual curve followed the target curve within a few percent. Two further steps were used, to evolve a three-quarter wave and a full wave. The three-quarter wave evolved as well as the previous two, but the full wave did not quite get the actual curve to return to the "base line"

(set at 30). It looked as though the limits of the evolvability of the 4K CA cell module had been reached. An alternative fitness definition which weighted the later clocks was attempted, but had no effect. Fig.7 shows the result of the full sinusoid, both target and actual output.

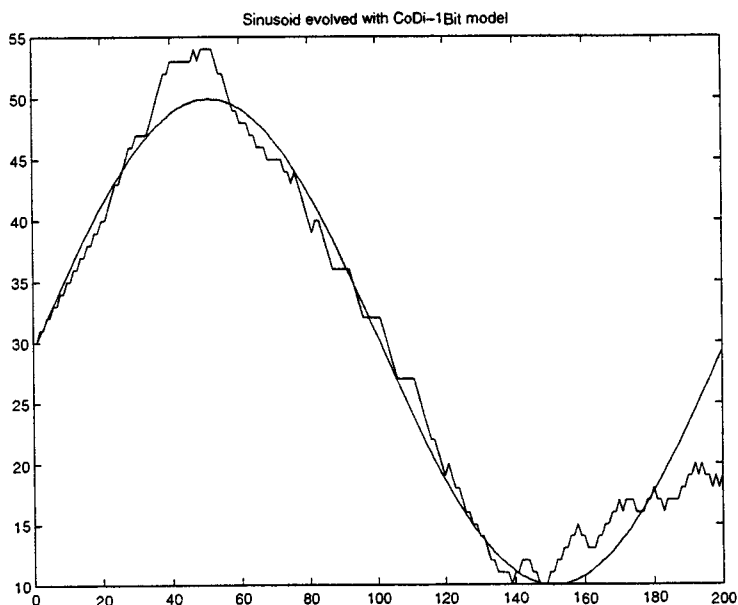


Fig. 7. Sinus wave generator evolved by CoDi model simulation on CAM-8

These few results were thought to be enough to show that the CoDi-1Bit model is evolvable enough to proceed with the construction of the CAM-Brain Machine.

3.4 Representation Issues

Finally, something needs to be said about the representation of the input and output signals. Of course, one is free to interpret these binary inputs and outputs as one chooses, e.g. as binary numbers, as biological "spikes" in a convolution/deconvolution model, as incrementers/decrementers, etc. The issue of representation is an important one which needs to be given a lot more thought in future CBM work.

4 CBM Architecture

The CAM-Brain Machine (CBM) is a special purpose hardware device based on Xilinx's programmable hardware (FPGA) chips (XC6264). The CBM is not just a faster version of the CAM-8 machine. It is based on the CoDi-1Bit neural model and introduces a number of highly specialized features. In this respect, CBM is not intended for general purpose cellular automata applications, such as hydrodynamic modeling, lattice gas simulation, or thermal stress analysis. The CBM consists of the following five major blocks :

- Cellular Automata Module
- Genotype/Phenotype Memory
- Fitness Evaluation Unit
- Genetic Algorithm Unit
- Module Interconnection Memory

Each of these blocks are discussed in detail below.

4.1 Cellular Automata Module

The cellular automata module is the hardware core of the CBM. It is intended to accelerate the speed of brain evolution through a highly parallel execution of cellular state updates. The CA module consists of an array of identical hardware logic circuits or cells arranged in a 3D structure of two modules of $16*16*16$ (4096 cells) or one module of $16*16*32$ (8192 cells). Cells forming the top layer of the module are recurrently connected with the cells in the bottom layer. A similar recurrent connection is made between the cells on the north and south, and the east and west vertical surfaces. Thus a fully recurrent toroidal cube is formed. This feature allows a much higher axonal and dendritic growth capacity by effectively doubling each of the three dimensions of the block.

4.2 Universal Cell

Each hardware logic cell contains circuitry to implement several different CBM functions, neuron, axon, dendrite and blank cell. Depending upon a particular chromosome of the neural module, each of these "universal" cells will perform one of the 4 possible functions in a given module during the signaling phase and may change its type from blank to an axon or a dendrite type during the growth phase. A blank cell can also be initialized to function as a neuron before the beginning of the growth phase.

4.3 CA Module Implementation in Hardware

The CA module is implemented with new Xilinx FPGA devices XC6264. These devices are fully and partially reconfigurable, feature a new co-processor architecture with data and address bus access in addition to user inputs and outputs,

and allow the reading and writing of any of the internal flip-flops through the data bus. An XC6264 chip contains 16384 logic function cells, each cell featuring a flip-flop and some Boolean logic capacity, capable of toggling at a 220 MHz rate. Logic cells are interconnected with neighbors at several hierarchical levels, providing identical propagation delay for any length of connection. This feature is very well suited for a 3D CA space configuration. Additionally, clock routing is optimized for equal propagation time, and power distribution is implemented in a redundant manner. To implement the CA module, a 3D block of identical logic cells is configured inside each XC6264 device, with CoDi specified 1-bit signal buses interconnecting the cells. Each XC6264 device, available in a QFP304 (quad flat pack) package, has up to 180 input/output pins. The number of available pins limits to 180 the maximum number of external connections to be made when interconnecting several XC6264 chips to form the 4096-cell space. Another limitation is imposed by the amount of logic needed to implement all the necessary functions in the CoDi cells. With these limitations in mind, the best arrangement for each XC6264 part is a block of $4*4*8$ cells (total of 128), which results in 160 externally connected cell surfaces, 4 surfaces of $4*8$ and 2 surfaces of $4*4$ cells. To assemble a 4096-cell module, 32 XC6264 chips are needed, configured as $2*4*4$ chips on 4 boards (dual surface mounted). The amount of hardware logic available for each CA cell is $16384/128 = 128$ Xilinx function blocks. This amount is sufficient for implementing all necessary CoDi model functions for growth and signaling, cell interconnection, and other requirements. All cells exchange signals and perform their respective functions synchronously at the rising edge of the master clock running at 30 MHz. The clock rate of 30 MHz (33 ns cycle) permits eleven layers of logic between reregistration to be used for CA cell circuitry design. (Each logic layer's propagation delay is 2.5-3 ns). During every clock cycle (33 ns), all 4096 cells are updated. This yields a speed increase of 614.4 times over the CAM-8 speed for the 1-bit 3D CoDi cells update rate of 200 Mcells/s. This number demonstrates "raw" speed increase, which is somewhat reduced by the module reconfiguration time. (See below).

4.4 Power Consumption and Heat Management

Dynamic power consumption of the module hardware is proportional to the clock rate, the number of flip-flops toggling simultaneously, and the capacitive loading of all the connections inside and outside the FPGAs. Static power consumption is expected to be negligible. Calculations yield a dynamic power consumption of the order of 4 watts for each XC6264 device. With a total of 32 FPGAs, the power consumption of the module is 128 watts (25.6 amperes). It is expected that the rest of the CBM circuitry will additionally require 100 watts. Given the estimated power dissipation, thermal management of the device can be accomplished using standard forced airflow techniques.

4.5 Module Inputs and Outputs

Although the module architecture is fully recurrent, some of the recurrent connections between top and bottom, east and west, etc., are reserved for external inputs and outputs. There are 8 cells on each surface of the module reserved to be used as external inputs to the module (a total of 48) and another 8 cells on each surface to be used as external outputs (a total of 48). Input and output locations are evenly spread over each surface of the module cube. In order to establish connections to external signals, an input cell must be evolved as either a dendrite or a neuron pointed inside. An output cell must be evolved as an axon or a neuron, pointed outside.

4.6 Genotype and Phenotype Memory

There are two modes of CBM operation, namely evolution mode and run mode. In the evolution mode, memory space is used to store the chromosome bitstrings of the evolving population of modules (module genotype). For a module of 4096 cells there are 20480 bits of memory needed. For each module the genotype memory also stores information concerning a maximum of 128 neurons locations and orientations inside the module. This includes, X, Y, Z coordinates (12 bits), gating code (3 bits), input functions (excitatory/inhibitory) (5 bits), giving a total of 20 bits per neuron. Thus, the total chromosome memory requirement for one module is $20480 + 20 \times 128 = 2880$ bytes.

In run mode, memory is used as phenotype memory for the evolved modules. The phenotype data describes grown axonal and dendritic trees and their respective neurons for each module. For phenotype storage there are 6 bits required per cell, i.e. for gating (3 bits), cell type (2 bits), and signal value (1 bit). Neuron cells additionally require 4 bits for the accumulator value stored. Phenotype data is loaded into the CA module to configure it according to the evolved function. Every 1024 cell board on a 4 board module will have 8 Mbytes of its own genotype/phenotype memory, with four 32-bit data buses for parallel access. 32 Mbytes of memory can store over 10 thousand modules at a time. This amount is sufficient to evolve 10 thousand modules at high speed, or to run a simulated brain with one million neurons. A large memory will be based in the main memory of the host computer (Pentium-Pro) connected to the CBM through a PCI bus, capable of transferring data at 132 Mbytes/s. Genotype/phenotype memory is connected to the hardware CA module and is used for rapid reconfiguration of the neural module by loading new chromosome data (or phenotype data) into the hardware registers of each cell through the XC6264 data bus access. Thus the fast hardware for the CA module is time- multiplexed between multiple neural modules in a large brain.

4.7 Hardware Reconfiguration Speed

The module reconfiguration speed is limited by the speed of the XC6264 data bus access and the width of the memory data bus. Each XC6264 device is capable

of accessing 32 internal flip-flops at a 25 MHz rate, which results in 150 million 5-bit cell reconfigurations per second. Each of the 2-chip sections on each of the four boards has its own memory bus. During parallel access to 16 FPGA's a rate of 2.4 billion cell reconfigurations per second is achieved. Thus, a 4096-cell FPGA hardware module can be reconfigured in 1.7 microseconds. There is a two-stage pipelined configuration register in every cell which permits the loading of the configuration data of a new module (phenotype and genotype) while running a previous module. At the end of the signaling or growth phase, the new configuration is instantly available. This feature guarantees continuous running of the CA module at full speed with practically no interruptions for reloading.

4.8 Overall CBM Speed Estimate

The goal of the CBM architecture is to exploit the full theoretical speed of parallel cell update, equal to 122.88 billion cells per second, or 614.4 times faster than the CAM-8. In order to achieve this, each module must be run for at least the time needed to load a module's configuration data (1.7 microseconds) which is equivalent to 51 update cycles. Also, a dual buffered set of signal FIFOs (see sections 3.3 and 3.5) will be provided to avoid interruptions of input/output signal data reloading. However, with some unavoidable overhead, the overall speed is expected to be around 110 billion cell updates per second.

4.9 Fitness Evaluation Unit

When a useful module is being evolved, each instance of a module must be evaluated in terms of its fitness for a targeted task. During the signaling phase, each module generates a sequence (an array) of 48 output signals, or vectors, which is compared with a target array in order to guide the evolutionary process. This comparison gives a measure of performance, or fitness, of the module. Fitness evaluation is supported by a hardware unit which consists of an input vector array stack, a target vector array stack, and a fitness comparator. Both stacks are 6-byte wide SRAMs (FIFOs) storing up to 2048 input and target vectors each to support the signaling phase of up to 2048 cycles. During each clock cycle an input vector is read from its stack and fed into the module's inputs. At the same time, a target vector is read from its stack to be compared with the current module output vector by the fitness evaluation unit. The fitness comparator computes a Hemming distance between each output vector and a corresponding target vector, and accumulates the result for the whole duration of the signaling phase. At the end of the signaling phase, a final measure of the module's fitness is instantly available. Multiple target and input arrays are stored in the host computer memory.

4.10 Genetic Algorithm Unit

To evolve a module, a population of 100 modules is evaluated by computing every module's fitness measure, as described above. The ten best modules are

then selected for further reproduction. A hardware support will be provided to store and update a "current best ten" list. This list is an array of the current ten best module numbers. Each number is an address of the module chromosome in the chromosome memory. After each generation of modules, the ten best are mated and mutated to produce 100 offspring modules to become the next generation. Mating and mutation is performed by the host computer software on the chromosome memory. Because this process can be performed in parallel with the module evaluation, and only takes place once in a generation, it is expected that there will be no significant slowdown in the evolutionary process.

4.11 Module Interconnection Memory

In order to support the run mode of operation, which requires a large number of evolved modules to function as an artificial brain, a module interconnection memory is provided. It consists of an output vector array stack, similar to the input array and target stack, and a larger memory for inter modular pathway storage. When each module of a large brain is configured in the CA hardware core (by loading its phenotype), an input stack is loaded with an array of input vectors. These vectors are previously stored output vectors recorded during the signaling phase of other modules, connected to this module. A large module interconnection memory will store the connection map and the current state of signals "traveling" between modules. There will be software and hardware support provided for combining output arrays from up to 8 modules, to be used as inputs for one module. In addition, an externally connected set of inputs and outputs is provided in hardware, which will allow the reception of signals from external sensors and the sending of signals to external effectors for robotic control. When running a simple million-neuron brain which contains ten thousand 100-neuron modules, the CBM is capable of updating each module for 100 cycles at the rate of about 27 times per second, allowing real-time control of robotic devices.

5 Conclusions

This paper presented a simplification of the CoDi model (called "CoDi-1Bit") for the evolution of neural structures, based on cellular automata. We reduced the signals to one bit with an oscillating cellular automata. This simplification made it possible to implement CoDi-1Bit with about 150 "functional units" of a Xilinx XC6264 chip per CA cell. This led to the development of the CAM-Brain Machine (CBM), whose architecture is also described. Hence, with the new CoDi model, it will be possible to evolve neural structures directly in hardware.

References

1. Toffoli, T. & Margolous, N. '*Cellular Automata Machines*', MIT Press, Cambridge, MA , 1987.
2. von Neumann, J. '*Theory of Self-Reproducing Automata*', ed. Burks A.W. University of Illinois Press, Urbana , 1966
3. Codd, E.F. '*Cellular Automata*', Academic Press, NY 1968.
4. de Garis, H. '*CAM-BRAIN: The Evolutionary Engineering of a Billion Neuron Artificial Brain by 2001 which Grows/Evolves at Electronic Speed Inside a Cellular Automata Machine (CAM)*', in '*Towards Evolvable Hardware*', Springer, Berlin, Heidelberg, NY , 1996.
5. Lloyd, S. '*A Potentially Realizable Quantum Computer*', Science **261**, 1569-1571 1993.
6. Koza, J.R. '*Genetic Programming: On the Programming of Computers by the Means of Natural Selection*', Cambridge, MA, MIT Press , 1992
7. Koza, J.R. & Bennet, F.H. & Andre, D. & Keane, M.M. '*Toward Evolution of Electronic Animals Using Genetic Programming*', ALife V Conference Proceedings, MIT Press , 1996.
8. Sipper, M. '*Co-evolving Non-Uniform Cellular Automata to Perform Computations*', Physica D **92**, 193-208 1996.
9. Carter, F. L. '*Molecular Electronic Devices*', North-Holland, Amsterdam, NY, Oxford, Tokyo , 1986.
10. Gers, F. A. & de Garis, H. '*Porting a Cellular Automata Based Artificial Brain to MIT's Cellular Automata Machine 'CAM-8'*', SEAL'96 Conference Proceedings S7-3 , 1996.
11. Gers, F. A. & de Garis, H. '*CAM-Brain: A New Model for ATR's Cellular Automata based Artificial Brain Project*', ICES'96 Conference Proceedings S7-5 , 1996.
12. Margolus, N. '*Crystalline Computation*', (Preprint).
13. Korkin M. & de Garis H. & Gers F.A. & Hemmi H. '*CBM (CAM-Brain Machine) : A Hardware Tool which Evolves a Neural Net Module in a Fraction of a Second and Runs a Million Neuron Artificial Brain in Real Time*', Genetic Programming Conference, July 1997, Stanford, USA , 1997.

Million Module Neural Systems Evolution

The Next Step in ATR's Billion Neuron Artificial Brain ("CAM-Brain") Project

Hugo de GARIS(1) Lishan KANG (2) Qiming HE (2),
Zhengjun PAN (2) Masahiro OOTANI (3)
Edmund RONALD (4)

(1) ATR Human Information Processing Research Laboratories,
2-2 Hikaridai, Seika-cho, Soraku-gun,
Kansai Science City, Kyoto-fu, 619-02, JAPAN.
degaris@hip.atr.co.jp <http://www.hip.atr.co.jp/~degaris>

(2) Key State Laboratory of Software Engineering,
Wuhan University, Wuhan 430072, PRC.
kang, heqm, zjpan@rjgc.whu.edu.cn <http://www.rjgc.whu.edu.cn>

(3) Electrical and Electronic Engineering Department,
Toyohashi University of Technology,
1-1 Hibarigaoka, Tenpaku-cho, Toyohashi-shi, Aichi-ken, 441, JAPAN.
ohtani@icg.eee.tut.ac.jp <http://icg.eee.tut.ac.jp/~ohtani>

(4) Centre de Mathematiques Appliquees
Ecole Polytechnique, 91 128 Palaiseau, France.
eronald@cmapx.polytechnique.fr

Abstract. This position paper discusses the evolution of multi-module neural net systems, where the number of neural net modules is up to ten million (i.e. an "artificial brain"). ATR's "CAM-Brain" Project [de Garis 1993, 1996] has progressed to the point where it is technically possible (using a new FPGA (Field Programmable Gate Array) based evolvable hardware (EHW or E-Hard) system to be completed by the spring of 1998 [Korkin & de Garis 1997]) to begin to evolve and build an artificial brain containing 10,000 neural net modules. This development raises the prospect that within a few years these numbers will rapidly increase. This paper introduces some issues that such massive system-building will generate. The immediate question is "What should we evolve?" This paper presents some suggested evolvable system targets containing N neural net modules, where N = 100; 1000; 10,000; 100,000; 1,000,000; 10,000,000 with an emphasis on the N = 100 case, for purposes of illustration. The issues involved are not only of a conceptual and evolutionary engineering nature, but (when N is large) economic, managerial and even political as well.

Keywords : Million Neural Net Module Artificial Brains, Evolvable Hardware (EHW, E-Hard), Evolutionary Engineering, Artificial Brain Architectures, CAM-Brain Machine (CBM), Kitten Robot (Robokoneko), Cellular Automata, Neural Networks, Genetic Algorithm, "J-Brain Project".

1 Introduction

1.1 Background

This is a position paper, which states the visions of ATR's Brain Builder Group (BBG) for the next few years. The present aim of our project is to build massively parallel evolvable hardware which will achieve the scale necessary for real-time intelligent perception and control - such hardware we call "artificial brains".

These artificial brains will be too large and too complex to program explicitly - hence we aim to evolve them. The evolutionary approach is inspired by our past experiments with an artificial creature (Lizzy) which we hand-assembled from neural-net modules. However, Lizzy's modules were evolved with the aid of hand-crafted fitness functions (see below). We argue that the crafting of fitness functions and the assembly of evolved modules will be done by a new breed of software professionals called Evolutionary Engineers (EEs).

As a logical model for scalable parallel hardware, we have adopted a 3-dimensional Cellular Automaton paradigm, which is at present moving from simulation to physical prototypes. In 1998, the BBG will possess new hardware which will make the building of 10,000 neural net module artificial brains practical. Hence the time is ripe to start thinking about what might be interesting to build with 100,000; 1,000,000 etc modules.

1.2 ATR's "CAM-Brain Project"

This initiative [de Garis 1993, 1994, 1995, 1996; Gers & de Garis 1996, Korkin & de Garis, 1997], is a project funded by Japan's ATR, running from 1993 to 2001, that aims to build a billion neuron artificial brain by 2001, using cellular automata based neural net modules. Owing to the simplicity of the cellular model, the state of each cell can be stored in a few bytes of RAM (2 bytes in the CoDi 1-bit model [Gers, de Garis and Corkin 97]. Modern workstations easily accept 1 GigaByte of RAM, easily allowing the simulation of networks with ten million artificial neurons, albeit at software speeds. The next step in the CAM-Brain Project is to find a way to evolve neural network modules quickly.

To solve this problem, the Brain Builder Group at ATR is building an electronic device called a "CAM-Brain Machine (CBM)" which grows/evolves the neural circuits (neurons, axons, dendrites and synapses) at electronic speeds inside Xilinx's new Field Programmable Gate Array (FPGA) XC6264 chips [Xilinx 1996]. The CBM implementation was subcontracted in December 96, and a proof-of-concept prototype has already been delivered, with 2 gate-arrays and PCI bus host connectivity; The full version should be delivered in the first quarter of 98. It will be able to evolve several hundred generations of a Genetic Algorithm (GA) population of a 100 neural circuits in a fraction of a second, i.e. a theoretical upper limit of hundreds of thousands of evolved modules a day. Each neural net module consists of some 6000 cellular automata cells in a 3D space, containing roughly 100 artificial neurons. For details see [Korkin & de Garis 1997].

Of course such astronomical module evolution speeds will not be obtainable in practice, because the real bottleneck will shift from the former slowness of software evolution speeds to the human thinking time required to conceive the inter-module circuitry, and the fitness definitions of the various modules contained in the circuit. (At this stage of the research, the individual modules are evolved, whereas the connections between modules and the overall multi-modular circuitry is user (i.e. humanly) specified. Simultaneous multi-modular evolution is a topic for future research. Interestingly, the CBM will allow many thousands of modules to evolve simultaneously in the work station RAM, so this option can be explored in 1998.

The CBM includes programmable hardware into which human programmers can compile the fitness definition for each module. Thus the fitness of each evolving circuit will be measurable at electronic speeds. The CBM can also function as a successor to the machine currently used by ATR's Brain Builder Group to update the cellular automata (CA) cells which form the basis of the evolvable neural medium [de Garis 1994]. At present, ATR uses MIT's "CAM-8" machine, which can update 200 million CA cells a second. However the chips and methodology of this machine date from 1988. The CBM will be able to update about 100 BILLION cells a second, i.e. about 500 times faster, as Moore's law would predict. Once the evolution of all the neural modules is complete (using the CBM), their evolved CA circuits will be downloaded into a gigabyte of workstation memory to be updated at run time by the CBM. Current estimates are that the CBM will be able to update each CA cell in the RAM at a rate of about 2500 times per second, which we believe sufficient for real time robot control experiments.

In parallel with the testing of the capabilities of the CBM, we intend to design a kitten robot called "Robokoneko" (which is Japanese for "robot kitten") which will be life size, with four legs, a tail, two small TV cameras for eyes, two microphones for ears, an on-board battery and a two-way radio antenna to communicate sensor data to, and movement instructions from, the offline CBM and workstation. "Robokoneko" should weigh less than 2 kilos, so that it can romp around the lab at (hopefully) real life kitten speeds, displaying a repertoire of kitten like behaviors, which should be possible because its brain will consist of 10,000 evolved neural net *modules*.

The CBM will make the creation of million-module artificial brains realistic within the next few years, provided of course, that there are enough human "evolutionary engineers" (EEs) to think up all the fitness definitions and inter module circuit designs in a reasonable time, a requirement which will necessitate the creation of human "fitness (definition) teams" - e.g. the "vision" team, the "legs" team, the "behavioral control" team, etc. Therefore, the time is ripe to start thinking about what such million-module systems might do. This paper provides some suggestions concerning what kinds of artificial brains (with increasing complexities and capabilities) might be interesting to evolve over the next few years. Of course, once the CBM hardware is ready, ideas on what to evolve will develop quickly, especially when other research centers buy the hardware from ATR. The existence of the hardware will stimulate development of the appropriate theories and methodologies. It often happens in the history of science that a new technology or new tool transforms a specialty, e.g. consider the impact that the

invention of the telescope had on astronomy, or the impact of the electron microscope on cell biology etc.

The remainder of this paper consists of the following. Section 2 presents in some detail a 100 module "artificial brain" architecture, which gives an idea of the kind of thing envisioned by the authors. Section 3 presents some suggestions on what might be interesting to evolve for larger and very much larger systems. Finally, in section 4, some comments on the longer term future of brain building are given.

2 A 100 Module System

The aim of this section is to give a fairly concrete example of a multi-module system. The ideas used in this example may prove useful when the time comes to build 1000; 10,000; etc module systems.

de Garis's PhD thesis and other papers [de Garis 1991a,b, 1992] details the simulation of an animat lizard called LIZZY. This 3D quadruped is shown in Fig. 1. It possessed 3 basic behaviors, namely - eat, don't be eaten, and mate. These behaviors were generated and controlled by evolved neural net modules assembled into humanly specified control circuits. (Perhaps at a later stage of research it might be possible to evolve the inter-module connections). Actually, with the workstations of that period, only the detectors and motion controllers were evolved (for reasons of simulation speed - every time a new neural net module was added to the software, the speed of LIZZY's motion on the computer screen decreased). (For a movie of Lizzy's motions, see de Garis's web site, and for full details of Lizzy, see de Garis's PhD thesis, Ch. 7, also on his web site).

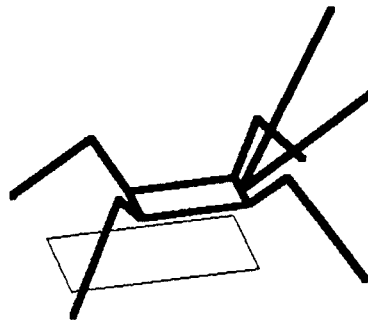


Fig. 1 LIZZY : An Artificial Creature

This section presents a more complete architecture of LIZZY, including the evolved control modules, to illustrate how it is possible to build artificial nervous systems (artificial brains) using evolved neural net modules. This approach will form the basis for future attempts to build million module artificial brains. These modules can be classified into three broad categories, namely - detector, decision, and motion. In broad terms, the detectors feed their output to the decision

modules (which can be combined to form "production rules"), and the outputs from the production rule circuits determine which behaviors to switch on.

LIZZY's antenna are capable of detecting the frequencies of sinusoid signals emanating from an arbitrary fixed point (using evolved frequency detectors). LIZZY is capable of detecting three kinds of creature in its environment - a prey (which, in the simulation model, emits a high frequency), a mate (which emits a middle frequency), and a predator (which emits a low frequency). Each antenna at its tip contains an average signal strength detector. At the base of the two antenna is a signal strength difference detector, which allows Lizzy to orientate towards a prey or a mate, or away from a predator. In the model, the amplitude of the sinusoid signals emanating from the fixed point drops off linearly with distance.

Lizzy's behavioral repertoire is as follows. If a prey or a mate is nearby (as indicated by the frequency detector), Lizzy orientates toward the signal source, walks until it is close, then stops and eats or mates (i.e. the front or back legs move up and down). If the frequency detector shows a predator, Lizzy orientates away from the source and walks away until the signal strength is weak.

There are 5 evolved motions, i.e. one module per motion.

- a) Walk straight ahead
- b) Turn left
- c) Turn right
- d) Peck at food (raise/lower front legs)
- e) Mate with partner (raise/lower back legs)

These motions are switched on and off by enabling signals coming from production rules of the form -

IF (A&B&C) => motion 2, where A,B,C are conditions.

These production rules and motions are created by evolving neural net modules. In greater detail, a Lizzy production rule might look like the following -

IF[(prey)&(SS(L)>SS(R))&(SS<K1)&(SSD>K2)] => Turn-L

i.e. if the source emits a high frequency (i.e. prey = a small animal = food) & the signal strength at the left antenna (SS(L) = signal strength detector (left)) is greater (i.e. closer to the source) than the SS(R) & the signal strength < constant K1 (i.e. weak signal = animal is far away) & signal strength detector difference SSD = ISS(L) - SS(R) > constant K2 THEN turn left.

In other words, if its a prey, and the left antenna is closer to the prey and the prey (signal) is too far way to eat (so that Lizzy has to move) and Lizzy is not oriented toward the prey, then turn left. This is just common sense.

Other rules are -

IF[(prey)&(SS(R)>SS(L))&(SS<K1)&(SSD>K2)] => Turn-R

which is the right equivalent of the above rule.

IF[(prey)&(SS<K1)&(SSD<K2)] => Walk

(i.e. if prey and far away and aligned with prey => walk)

IF[(prey)&(SS>K1)] => Eat (peck at prey)

(i.e. if prey and very close, i.e. the signal strength is high => eat)

Four similar rules exist for mating -

IF[(mate)&(L>R)&(SS<K1)&(SSD>K2)] => Turn-L

IF[(mate)&(R>L)&(SS<K1)&(SSD>K2)] => Turn-R

IF[(mate)&(SS<K1)&(SSD<K2)] => Walk

IF[(mate)&(SS<K1)] => Mate (with partner)

and similarly for a predator -

IF[(predator)&(L>R)&(SS>K3)] => Turn-R

IF[(predator)&(R>L)&(SS>K3)] => Turn-L

IF[(predator)&(SS>K3)&(SSD<K2)] => Walk (Run)

When a motion is switched on, it continues for time "T" (CA clock cycles), to avoid rapid (useless) switching between behaviors (a notion called "behavioral commitment"), so a timer module is needed.

It is usually easier to evolve a single-function module than a multi-function module. However, sometimes a multi-function module is needed, as will be shown below. The neural net modules evolved in [de Garis 1990, 1991a,b, 1992] were all fully connected (binary fraction bit strings representing the connection weights were concatenated to form the GA chromosomes).

For more details on neural net module ("GenNet") evolution, see [de Garis 1990] or consult de Garis's PhD thesis downloadable from his web site). However, when using cellular automata based neural nets (as is the case for the CAM-Brain Project), the connectivity is less, but still dense enough for good evolvability [de Garis 1996]. Extensive connectivity leads to more complex dynamics, hence greater functionality.

Module Descriptions and Fitness Definitions

Having given an idea of the type of control rules which govern Lizzy's behaviors, we now show how the modules used in such circuits can be evolved.

Frequency Detector

Use as input, 3 sinusoids of different frequencies (low, middle, high), e.g. wavelength = 20, 30, 40 clock cycles, and many amplitudes. Evolve 3 separate modules, one for each frequency detection. When the non desired frequency is input, a low output is desired (where output signal values range from 0.0 to 1.0). This output should be high if the input is the desired frequency.

(For details on the actual evolution of Lizzy's frequency detection modules, see de Garis's PhD thesis on his web site). The 3 frequencies are input sequentially (with a variety of amplitudes), and the numerical output values stored over a reasonable number of clock cycles. (The CA are synchronous). A fitness definition meeting these specifications could be the following. (Outputs range from 0.0 to 1.0).

$$\text{FITNESS} = \text{reciprocal}((\text{sum over } i \text{ clock cycles for non desired frequencies } (\text{OUT}_i - 0.0) * (\text{OUT}_i - 0.0)) + \text{weighting factor } (\gamma > 1) * (\text{sum over } i \text{ for desired frequency } (\text{OUT}_i - 0.8) * (\text{OUT}_i - 0.8)))$$

Note that it is useful to extract the outputs after a certain settling delay of the signals.

Signal Strength Detector (SS)

The output should be proportional to the average signal strength coming in (i.e. if the input signal is of the form $A \sin(\text{OMEGA} * Ti)$, then the SS output should be roughly $A^2/2$). So input a series of sinusoids with many amplitudes and many frequencies, with the following fitness definition. (Again, see de Garis's PhD thesis for details).

$$\text{FITNESS} = \text{reciprocal}(\text{sum over frequencies } (\text{sum over amplitudes } "j" (\text{sum over clock cycles } "i" (\text{OUT}_i - A_j * A_j / 2) * (\text{OUT}_j - A_j * A_j / 2))))).$$

Signal Strength Difference Detector (SSD)

An SSD is used to orientate Lizzy relative to the signal source, due to the signal strength differences between signals received at the left and right antenna. An SSD takes two SS outputs as inputs and outputs their difference. To evolve it, input combinations of real numbers, i.e. the tensor product of [0.1, 0.2, 0.3, ..., 0.8, 0.9] with itself.

$$\text{FITNESS} = \text{reciprocal} (\text{sum over clock cycles } "i" (\text{sum over } j = 0.1 \text{ to } 0.9 (\text{sum over } k = 0.1 \text{ to } 0.9 ((\text{OUT}_{ijk} - \text{modulus}(j - k)) * (\text{OUT}_{ijk} - \text{modulus}(j - k))))))$$

Other Modules

Other modules evolved include comparators, and gates, or gates and timers, maximum position detectors, saturators (amplify a signal to maximum).

Putting It All Together

Now that a number of individual modules have been defined, both in terms of function and fitness, they can be put together into humanly specified functional intermodular circuits. Fig. 2 shows how the following production rule can be implemented using various modules.

IF [(prey)&(SS(L)>SS(R))&(SS<K1)&(SSD>K2)] =>Turn-L

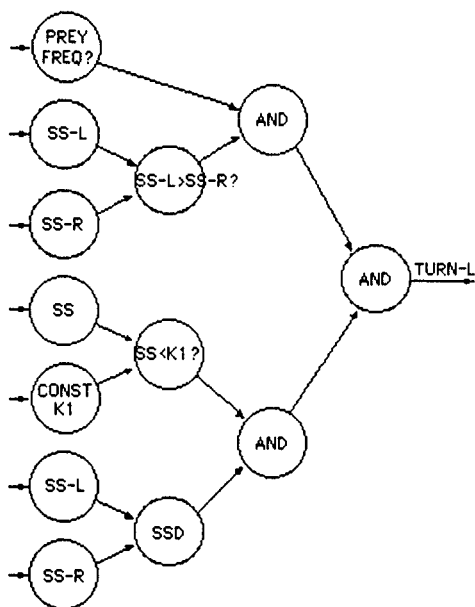


Fig. 2 A Production Rule Circuit

Fig. 3 shows how Lizzy's 5 behaviors can be switched on and off. Only one motion control can be sent to the legs at a time, e.g. if the WALK signal is strongest, then the module MAX-POSN module for WALK will go high and trigger WALK's Timer module (T), making WALK last for T cycles. When WALK's Timer is high, it sends saturated signals to the other MAX-POSN modules, switching them off, until WALK's Timer goes low, T clock cycles later. Then another (or the same) motion is high and a similar process occurs. In the action bus, output signals sum at intersections, which allows for a smooth transition between leg motion types.

The above discussion of LIZZY is only an example of the type of thinking involved in making multi-module neural architectures. We believe that new professions will be created, namely the "Evolutionary Engineer (EE)", whose job will be to invent neural module functions and fitness definitions, and "Brain

Architect (BA)", whose job will be to design artificial brains. In a large scale brain building project, it is likely that top level designers will be the BAs, and they will pass down their high level specifications to lower level EEs who will evolve the actual modules with CBMs. With the creation of the world's first CBM by the end of the century, the need for BAs and EEs will soon arise.

3 Larger Systems

This section is more speculative. It presents some early ideas on what kinds of "large N" systems might be interesting to evolve/build. We begin with $N = 1000$, and increase each time by an order of magnitude, up to 10,000,000. It also discusses some of the personnel, management and political issues involved, as the scales of the projects increase.

The $N = 1000$ Case

The above LIZZY architecture gives an idea of how artificial nervous systems (or, if there are enough modules - artificial brains) can be assembled from evolved neural net modules. By simply adding to LIZZY's behavioral repertoire, one can quickly increase the number of modules to 1000. LIZZY could be made to behave like a toy kitten, so that it could jump, chase its tail, emit simple cries, run at different speeds, etc. A hard working evolutionary engineer (with a CBM) could probably build a 1000 module creature alone (in simulation), but if the modules control a physical robot, maybe a team of two people (i.e. with one of them being a roboticist) could do this work in a year.

The $N = 10,000$ Case

With ten thousand modules, one can begin to experiment with vision and hearing. Simple artificial retinas could be built with some post retinal processing. Maybe some memory could be added. This seeing and hearing creature could avoid objects, approach or flee from slow or fast moving objects respectively, pick up things, etc. At this number of modules, two people would be stretched to do the whole project in a few years. Probably a small team of several evolutionary engineers, a roboticist, and a general programmer would be needed (plus a CBM of course) - say, four people at the least. In fact, the main task of the authors for 1997 will be to create a 10,000 module architecture to control the kitten robot "Robokoneko", as mentioned earlier in this paper. As of February 1997, this work has not yet started, so the many challenges of generating a 10,000 module system design are not yet familiar to us, although this will quickly change once the CBM is ready (the kitten robot as well) by the end of 1997. Our papers in 1998 (the year in which the CBM, Robokoneko and a 10,000 module architectural plan are integrated) will probably have a very different flavor compared to this one.

The N = 100,000 Case

With one hundred thousand modules, more serious versions of creatures with memory, vision, motion generation and detection, hearing, simple comprehension, and multi-sensor interaction can be built. At this number of modules, one needs to begin thinking seriously about the management and personnel planning aspects of such a project. Probably about a dozen or more people would be needed to finish such a project within a few years, a figure within the reach of many universities and smaller companies. Probably most examples of artificial brains will be of this size, given the realities of university and medium sized companies research budgets.

The N = 1,000,000 Case

For a million module system, the management and personnel demands become large. For example, if one makes the assumption that on average, it takes a (human) evolutionary engineer (EE) one hour to conceive and compile the fitness definition of a module (and link the module to a global inter module architecture), then how many EEs would be needed for a 2.5 year, million module, artificial brain research project? Assuming an 8 hour day, a 40 hour week, a 50 week year, i.e. a 2000 hour year, the project would need 500 EE-years, spread over 2.5 years, hence 200 EEs would be needed. This number could be afforded by a large company, so one can expect companies to start building large artificial brains before the year 2000.

Of course, the above figure is based only on the fitness definition creation times. A brain builder project would need to consider many other factors (e.g. sensor development, robotics problems, coordination of human groups, neural signal pathway specification and routing, etc.), which would add to the project time, personnel numbers and costs. Nevertheless, with a suite of CBMs (i.e. each with fraction-of-a-second module evolution times), the complete construction of a million module artificial brain becomes quite realistic within 5 years, say by the end of 2001. It will be interesting to read this paper 5 years from the time of writing (February 1997) to see how far off the mark we were, if at all.

Suggested examples of million module systems might be, artificial kitten pets for children and the aged, robot "guide dogs" to help blind people cross the road, household cleaner robots, etc. These systems would include quite elaborate artificial retinas, and post retinal processing, memory processing, sound generation, even early speech.

The N = 10,000,000 Case

One immediately jumps to the 2000 personnel range. This is a major national or international project, and too big for all except the biggest companies. de Garis dreams that Japan will start a "J-Brain (J = Japanese) Project" (2001-2005), to build a ten million module (billion neuron) artificial brain, once the technologies and methodologies have matured, based on experience gained from

the building of smaller brains. Large national projects of this kind would compare with America's NASA project to put a man on the moon.

Japan's "J-Brain Project" would attempt to build the world's most intelligent artificial brain. In the process, the project would create a whole new industry, where the brain-like computer market would eventually be worth a trillion dollars a year, and would bring tremendous international prestige to Japan, which so far has had an international "uncreative copycat" image. Every household would want a home cleaner robot that could do the vacuuming, the shopping, emptying the garbage, washing the car, etc.

In fact, at the time of writing (February 1997), one of Japan's major scientific research funders, the STA (Science and Technology Agency) has budgeted 20 TRILLION (20,000,000,000,000) yen over a 20 year period, starting in the fall of 1997, to finance three areas of brain research, namely basic neuro-science, neuro-medical-science, and brain engineering. At roughly 20 million yen per researcher per year (salary, equipment, tax etc), that's 50,000 researchers a year. de Garis hopes to persuade the STA to finance the "J-Brain Project" (a mere 2000 researchers)!

Japan is not the only country interested in brain building. Officials of America's scientific research funders NSF and DARPA have both asked de Garis to talk to them about CAM-Brain, which he did in April 1997. The Chinese are also interested - three of the authors and brain stormers of this paper are Chinese. ATR and Wuhan University collaborate closely. Wuhan University hopes to become the major center for brain building research for the whole of China.

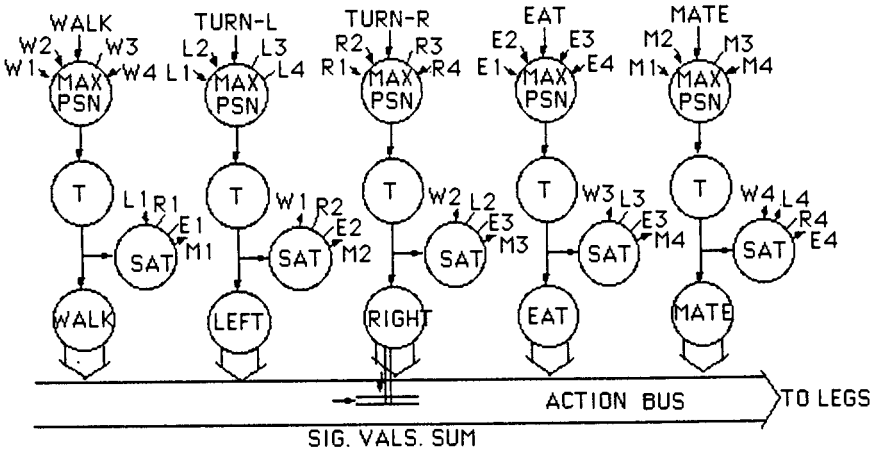


Fig. 3 Motion Selection Circuit

4 Comments

With million module systems and larger, people can begin to test serious models of biological brain function. As electronic technology improves (in our exciting electronic era of what de Garis calls "massive Moore doublings", i.e. where the size of the progressive increments in electronic speeds and densities is becoming enormous, with a 4 GIGAbit experimental memory device announced in early 1997), it will be possible to evolve more biologically realistic neural circuits, so that brain building and brain science can come closer together and benefit from each others advances.

However, ten million module artificial brains are only the beginning. Molecular scale electronics (e.g. single electron transistors (SETs), molecular electronic devices (MEDs), quantum dots (QDs), etc) will mean that the heat generation problem (which arises when conventional, irreversible, register-clearing, computing techniques destroy information) at molecular level will have to be overcome. If not, molecular scale circuits will reach temperatures of exploding dynamite. The only way to go will be to reduce the heat (virtually to zero) by using "reversible computation" techniques [Feynman 1996].

Heatless computers will allow electronics to use 3D, size-independent circuits, with 1 bit per atom, so in theory, one could have self assembling asteroid size computers with 10 to power 40 components. These huge numbers absolutely dwarf the human brain's pitiful tens of billions of neurons.

Brain builder technology in the late 21st century will threaten humanity's status as dominant species, with profound global political consequences. (See the "Cosmist" essays on this topic on de Garis's web site, or [de Garis 1996b]).

References

Note : de Garis papers can be found at site :- <http://www.hip.atr.co.jp/~degaris>

[de Garis 1990] "Genetic Programming : Building Artificial Nervous Systems Using Genetically Programmed Neural Network Modules", Hugo de Garis, in Porter B.W. & Mooney R.J. ed., Proc. 7th. Int. Conf. on Machine Learning, pp 132-139, Morgan Kaufmann, 1990.

[de Garis 1990b] "Genetic Programming : Modular Evolution for Darwin Machines", Hugo de Garis, Int. joint Conf. on Neural Networks, January 1990, Washington DC, USA.

[de Garis 1991a] "Lizzy: The Genetic Programming of an Artificial Nervous System", Hugo de Garis, Int. Conf. on Artificial Neural Networks, June, 1991, Espoo, Finland.

[de Garis 1991b] "Genetic Programming, Artificial Nervous Systems, Artificial Embryos, and Embryological Electronics", Hugo de Garis, in "Parallel Problem Solving from Nature", Lecture Notes in Computer Science, 496, Springer Verlag, 1991.

[de Garis 1992] "Artificial Nervous Systems : The Genetic Programming of Production Rule GenNet Circuits", Hugo de Garis, Int. Joint Conf. on Neural Networks, November 1992, Beijing, China.

[de Garis 1993] "Neurite Networks : The Genetic Programming of Cellular Automata based Neural Nets which Grow", Hugo de Garis, Int. Joint Conf. on Neural Networks, October 1993, Nagoya, Japan.

[de Garis 1994] "An Artificial Brain : ATR's CAM-Brain Project Aims to Build/Evolve an Artificial Brain with a Million Neural Net Modules inside a Trillion Cell Cellular Automata Machine", Hugo de Garis, New Generation Computing Journal, Vol. 12, No.2, Ohmsha & Springer Verlag.

[de Garis 1995] "The CAM-Brain Project : The Genetic Programming of a Billion Neuron Artificial Brain by 2001 which Grows/Evolves at Electronic Speeds inside a Cellular Automata Machine", Hugo de Garis, Int. Conf. on Artificial Neural Networks and Genetic Algorithms, April 1995, Ales France.

[de Garis 1996] "CAM-Brain : ATR's Billion Neuron Artificial Brain Project : A Three Year Progress Report", Hugo de Garis, Int. Conf. on Evolutionary Computation, May 1996, Nagoya Japan.

[de Garis 1996b] "Cosmism : Nano-Electronics and 21st Century War", Nanotechnology Magazine, July, 1996, also on de Garis's web site under "Essays".

[Feynman 1996] "The Feynman Lectures on Computation", R.P. Feynman, Addison Wesley, 1996.

[Gers & de Garis 1996] "CAM-Brain : A New Model for ATR's Cellular Automata Based Artificial Brain Project", Felix Gers & Hugo de Garis, Int. Conf. on Evolvable Systems, October 1996, Tsukuba, Japan.

[Korkin & de Garis 1997] "CBM (CAM-Brain Machine) : A Hardware Tool which Evolves a Neural Net Module in a Fraction of a Second and Runs a Million Neuron Artificial Brain in Real Time", Micahel Korkin & Hugo de Garis, Genetic Programming Conference, July, 1997, Stanford, USA.

[Xilinx 1996] "Xilinx Data Manual 1996".

Author Index

- Agapie, A. 183
Allouche, J.-P. 157

Bessière, P. 237

Collard, P. 69
Crisan, C. 263
Cuenca, C. 109

Dedieu, E. 237
Ducoulombier, A. 81

Eggenberger, E. 251
Eiben, A. E. 95
El Kharoubi, E. M. 275
Escazut, C. 69

de Garis, H. 315, 335
Gaspin, C. 145
Gers, F. 315
Glover, F. 3
Gottlieb, J. 55

Hauw, J. K. van der 95
He, Q. 335
Heudin, J.-C. 109

Kallel, L. 287
Kang, L. 335
Klahold, J. 303
Korkin, M. 315

Leblanc, B. 157
Lebeltel, O. 237
Loeffler, A. 303
Lutton, E. 157

Mauri, G. 123
Muehlenbein, H. 263

Naudts, B. 207

Oh, S. 195
Ootani, M. 335

Pan, Z. 335
Peyral, M. 81
Piccolboni, A. 123

Ravisé, C. 81
Rochet, S. 275
Rudolph, G. 223
Rueckert, U. 303

Salomon, R. 251
Schiex, T. 145
Schoenauer, M. 81, 287
Sebag, M. 81
Servet, I. 137
Slimane, M. 275
Stern, D. 137

Tanomaru, J. 167
Travé-Massuyès, L. 137

Venturini, G. 275
Verschoren, A. 207
Voss, N. 55

Yoon, H. 195

Lecture Notes in Computer Science

For information about Vols. 1–1289

please contact your bookseller or Springer-Verlag

Vol. 1290: E. Moggi, G. Rosolini (Eds.), *Category Theory and Computer Science. Proceedings, 1997*. VII, 313 pages. 1997.

Vol. 1291: D.G. Feitelson, L. Rudolph (Eds.), *Job Scheduling Strategies for Parallel Processing. Proceedings, 1997*. VII, 299 pages. 1997.

Vol. 1292: H. Glaser, P. Hartel, H. Kuchen (Eds.), *Programming Languages: Implementations, Logics, and Programs. Proceedings, 1997*. XI, 425 pages. 1997.

Vol. 1293: C. Nicholas, D. Wood (Eds.), *Principles of Document Processing. Proceedings, 1996*. XI, 195 pages. 1997.

Vol. 1294: B.S. Kaliski Jr. (Ed.), *Advances in Cryptology — CRYPTO '97. Proceedings, 1997*. XII, 539 pages. 1997.

Vol. 1295: I. Prívvara, P. Ružička (Eds.), *Mathematical Foundations of Computer Science 1997. Proceedings, 1997*. X, 519 pages. 1997.

Vol. 1296: G. Sommer, K. Daniilidis, J. Pauli (Eds.), *Computer Analysis of Images and Patterns. Proceedings, 1997*. XIII, 737 pages. 1997.

Vol. 1297: N. Lavrač, S. Džeroski (Eds.), *Inductive Logic Programming. Proceedings, 1997*. VIII, 309 pages. 1997. (Subseries LNAI).

Vol. 1298: M. Hanus, J. Heering, K. Meinke (Eds.), *Algebraic and Logic Programming. Proceedings, 1997*. X, 286 pages. 1997.

Vol. 1299: M.T. Pazzienza (Ed.), *Information Extraction. Proceedings, 1997*. IX, 213 pages. 1997. (Subseries LNAI).

Vol. 1300: C. Lengauer, M. Griebel, S. Gorlatch (Eds.), *Euro-Par '97 Parallel Processing. Proceedings, 1997*. XXX, 1379 pages. 1997.

Vol. 1301: M. Jazayeri, H. Schauer (Eds.), *Software Engineering - ESEC/FSE'97. Proceedings, 1997*. XIII, 532 pages. 1997.

Vol. 1302: P. Van Hentenryck (Ed.), *Static Analysis. Proceedings, 1997*. X, 413 pages. 1997.

Vol. 1303: G. Brewka, C. Habel, B. Nebel (Eds.), *KI-97: Advances in Artificial Intelligence. Proceedings, 1997*. XI, 413 pages. 1997. (Subseries LNAI).

Vol. 1304: W. Luk, P.Y.K. Cheung, M. Glesner (Eds.), *Field-Programmable Logic and Applications. Proceedings, 1997*. XI, 503 pages. 1997.

Vol. 1305: D. Corne, J.L. Shapiro (Eds.), *Evolutionary Computing. Proceedings, 1997*. X, 307 pages. 1997.

Vol. 1306: C. Leung (Ed.), *Visual Information Systems. X, 274 pages. 1997.*

Vol. 1307: R. Kompe, *Prosody in Speech Understanding Systems. XIX, 357 pages. 1997. (Subseries LNAI).*

Vol. 1308: A. Hameurlain, A.M. Tjoa (Eds.), *Database and Expert Systems Applications. Proceedings, 1997*. XVII, 688 pages. 1997.

Vol. 1309: R. Steinmetz, L.C. Wolf (Eds.), *Interactive Distributed Multimedia Systems and Telecommunication Services. Proceedings, 1997*. XIII, 466 pages. 1997.

Vol. 1310: A. Del Bimbo (Ed.), *Image Analysis and Processing. Proceedings, 1997*. Volume I. XXII, 722 pages. 1997.

Vol. 1311: A. Del Bimbo (Ed.), *Image Analysis and Processing. Proceedings, 1997*. Volume II. XXII, 794 pages. 1997.

Vol. 1312: A. Geppert, M. Berndtsson (Eds.), *Rules in Database Systems. Proceedings, 1997*. VII, 214 pages. 1997.

Vol. 1313: J. Fitzgerald, C.B. Jones, P. Lucas (Eds.), *FME '97: Industrial Applications and Strengthened Foundations of Formal Methods. Proceedings, 1997*. XIII, 685 pages. 1997.

Vol. 1314: S. Muggleton (Ed.), *Inductive Logic Programming. Proceedings, 1996*. VIII, 397 pages. 1997. (Subseries LNAI).

Vol. 1315: G. Sommer, J.J. Koenderink (Eds.), *Algebraic Frames for the Perception-Action Cycle. Proceedings, 1997*. VIII, 395 pages. 1997.

Vol. 1316: M. Li, A. Maruoka (Eds.), *Algorithmic Learning Theory. Proceedings, 1997*. XI, 461 pages. 1997. (Subseries LNAI).

Vol. 1317: M. Leman (Ed.), *Music, Gestalt, and Computing. IX, 524 pages. 1997. (Subseries LNAI).*

Vol. 1318: R. Hirschfeld (Ed.), *Financial Cryptography. Proceedings, 1997*. XI, 409 pages. 1997.

Vol. 1319: E. Plaza, R. Benjamins (Eds.), *Knowledge Acquisition, Modeling and Management. Proceedings, 1997*. XI, 389 pages. 1997. (Subseries LNAI).

Vol. 1320: M. Mavronicolas, P. Tsigas (Eds.), *Distributed Algorithms. Proceedings, 1997*. X, 333 pages. 1997.

Vol. 1321: M. Lenzerini (Ed.), *AI*IA 97: Advances in Artificial Intelligence. Proceedings, 1997*. XII, 459 pages. 1997. (Subseries LNAI).

Vol. 1322: H. Hußmann, *Formal Foundations for Software Engineering Methods. X, 286 pages. 1997.*

Vol. 1323: E. Costa, A. Cardoso (Eds.), *Progress in Artificial Intelligence. Proceedings, 1997*. XIV, 393 pages. 1997. (Subseries LNAI).

Vol. 1324: C. Peters, C. Thanos (Eds.), *Research and Advanced Technology for Digital Libraries. Proceedings, 1997*. X, 423 pages. 1997.

Vol. 1325: Z.W. Raś, A. Skowron (Eds.), *Foundations of Intelligent Systems. Proceedings, 1997*. XI, 630 pages. 1997. (Subseries LNAI).

- Vol. 1326: C. Nicholas, J. Mayfield (Eds.), *Intelligent Hypertext*. XIV, 182 pages. 1997.
- Vol. 1327: W. Gerstner, A. Germond, M. Hasler, J.-D. Nicoud (Eds.), *Artificial Neural Networks – ICANN '97*. Proceedings, 1997. XIX, 1274 pages. 1997.
- Vol. 1328: C. Retoré (Ed.), *Logical Aspects of Computational Linguistics*. Proceedings, 1996. VIII, 435 pages. 1997. (Subseries LNAI).
- Vol. 1329: S.C. Hirtle, A.U. Frank (Eds.), *Spatial Information Theory*. Proceedings, 1997. XIV, 511 pages. 1997.
- Vol. 1330: G. Smolka (Ed.), *Principles and Practice of Constraint Programming – CP 97*. Proceedings, 1997. XII, 563 pages. 1997.
- Vol. 1331: D. W. Embley, R. C. Goldstein (Eds.), *Conceptual Modeling – ER '97*. Proceedings, 1997. XV, 479 pages. 1997.
- Vol. 1332: M. Bubak, J. Dongarra, J. Waśniewski (Eds.), *Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Proceedings, 1997. XV, 518 pages. 1997.
- Vol. 1333: F. Pichler, R. Moreno-Díaz (Eds.), *Computer Aided Systems Theory – EUROCAST'97*. Proceedings, 1997. XII, 626 pages. 1997.
- Vol. 1334: Y. Han, T. Okamoto, S. Qing (Eds.), *Information and Communications Security*. Proceedings, 1997. X, 484 pages. 1997.
- Vol. 1335: R.H. Möhring (Ed.), *Graph-Theoretic Concepts in Computer Science*. Proceedings, 1997. X, 376 pages. 1997.
- Vol. 1336: C. Polychronopoulos, K. Joe, K. Araki, M. Amamiya (Eds.), *High Performance Computing*. Proceedings, 1997. XII, 416 pages. 1997.
- Vol. 1337: C. Freksa, M. Jantzen, R. Valk (Eds.), *Foundations of Computer Science*. XII, 515 pages. 1997.
- Vol. 1338: F. Plášil, K.G. Jeffery (Eds.), *SOFSEM'97: Theory and Practice of Informatics*. Proceedings, 1997. XIV, 571 pages. 1997.
- Vol. 1339: N.A. Murshed, F. Bortolozzi (Eds.), *Advances in Document Image Analysis*. Proceedings, 1997. IX, 345 pages. 1997.
- Vol. 1340: M. van Kreveld, J. Nievergelt, T. Roos, P. Widmayer (Eds.), *Algorithmic Foundations of Geographic Information Systems*. XIV, 287 pages. 1997.
- Vol. 1341: F. Bry, R. Ramakrishnan, K. Ramamohanarao (Eds.), *Deductive and Object-Oriented Databases*. Proceedings, 1997. XIV, 430 pages. 1997.
- Vol. 1342: A. Sattar (Ed.), *Advanced Topics in Artificial Intelligence*. Proceedings, 1997. XVII, 516 pages. 1997. (Subseries LNAI).
- Vol. 1343: Y. Ishikawa, R.R. Oldehoeft, J.V.W. Reynders, M. Tholburn (Eds.), *Scientific Computing in Object-Oriented Parallel Environments*. Proceedings, 1997. XI, 295 pages. 1997.
- Vol. 1344: C. Ausnit-Hood, K.A. Johnson, R.G. Pettit, IV, S.B. Opdahl (Eds.), *Ada 95 – Quality and Style*. XV, 292 pages. 1997.
- Vol. 1345: R.K. Shyamasundar, K. Ueda (Eds.), *Advances in Computing Science – ASIAN'97*. Proceedings, 1997. XIII, 387 pages. 1997.
- Vol. 1346: S. Ramesh, G. Sivakumar (Eds.), *Foundations of Software Technology and Theoretical Computer Science*. Proceedings, 1997. XI, 343 pages. 1997.
- Vol. 1347: E. Ahronovitz, C. Fiorio (Eds.), *Discrete Geometry for Computer Imagery*. Proceedings, 1997. X, 255 pages. 1997.
- Vol. 1348: S. Steel, R. Alami (Eds.), *Recent Advances in AI Planning*. Proceedings, 1997. IX, 454 pages. 1997. (Subseries LNAI).
- Vol. 1349: M. Johnson (Ed.), *Algebraic Methodology and Software Technology*. Proceedings, 1997. X, 594 pages. 1997.
- Vol. 1350: H.W. Leong, H. Imai, S. Jain (Eds.), *Algorithms and Computation*. Proceedings, 1997. XV, 426 pages. 1997.
- Vol. 1351: R. Chin, T.-C. Pong (Eds.), *Computer Vision – ACCV'98*. Proceedings Vol. I, 1998. XXIV, 761 pages. 1997.
- Vol. 1352: R. Chin, T.-C. Pong (Eds.), *Computer Vision – ACCV'98*. Proceedings Vol. II, 1998. XXIV, 757 pages. 1997.
- Vol. 1353: G. BiBattista (Ed.), *Graph Drawing*. Proceedings, 1997. XII, 448 pages. 1997.
- Vol. 1354: O. Burkart, *Automatic Verification of Sequential Infinite-State Processes*. X, 163 pages. 1997.
- Vol. 1355: M. Darnell (Ed.), *Cryptography and Coding*. Proceedings, 1997. IX, 335 pages. 1997.
- Vol. 1356: A. Danthine, Ch. Diot (Eds.), *From Multimedia Services to Network Services*. Proceedings, 1997. XII, 180 pages. 1997.
- Vol. 1357: J. Bosch, S. Mitchell (Eds.), *Object-Oriented Technology*. Proceedings, 1997. XIV, 555 pages. 1998.
- Vol. 1358: B. Thalheim, L. Libkin (Eds.), *Semantics in Databases*. XI, 265 pages. 1998.
- Vol. 1361: B. Christianson, B. Crispo, M. Lomas, M. Roe (Eds.), *Security Protocols*. Proceedings, 1997. VIII, 217 pages. 1998.
- Vol. 1362: D.K. Panda, C.B. Stunkel (Eds.), *Network-Based Parallel Computing*. Proceedings, 1998. X, 247 pages. 1998.
- Vol. 1363: J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer, D. Snyers (Eds.), *Artificial Evolution*. XI, 349 pages. 1998.
- Vol. 1364: W. Conen, G. Neumann (Eds.), *Coordination Technology for Collaborative Applications*. VIII, 282 pages. 1998.
- Vol. 1365: M.P. Singh, A. Rao, M.J. Wooldridge (Eds.), *Intelligent Agents IV*. Proceedings, 1997. XII, 351 pages. 1998. (Subseries LNAI).
- Vol. 1367: E.W. Mayr, H.J. Prömel, A. Steger (Eds.), *Lectures on Proof Verification and Approximation Algorithms*. XII, 344 pages. 1998.
- Vol. 1368: Y. Masunaga, T. Katayama, M. Tsukamoto (Eds.), *Worldwide Computing and Its Applications – WWCA'98*. Proceedings, 1998. XIV, 473 pages. 1998.
- Vol. 1370: N.A. Streitz, S. Konomi, H.-J. Burkhardt (Eds.), *Cooperative Buildings*. Proceedings, 1998. XI, 267 pages. 1998.
- Vol. 1373: M. Morvan, C. Meinel, D. Krob (Eds.), *STACS 98*. Proceedings, 1998. XV, 630 pages. 1998.