STUDIES IN *FUZZINESS*
AND *SOFT COMPUTING*

William E. Hart
N. Krasnogor · J. E. Smith
Editors

Recent Advances in Memetic Algorithms

Springer

Studies in Fuzziness and Soft Computing, Volume 166

Editor-in-chief
Prof. Janusz Kacprzyk
Systems Research Institute
Polish Academy of Sciences
ul. Newelska 6
01-447 Warsaw
Poland
E-mail: kacprzyk@ibspan.waw.pl

Further volumes of this series can be found on our homepage: springeronline.com

Vol. 150. L. Bull (Ed.)
Applications of Learning Classifier Systems, 2004
ISBN 3-540-21109-8

Vol. 151. T. Kowalczyk, E. Pleszczyńska, F. Ruland (Eds.)
Grade Models and Methods for Data Analysis, 2004
ISBN 3-540-21120-9

Vol. 152. J. Rajapakse, L. Wang (Eds.)
Neural Information Processing: Research and Development, 2004
ISBN 3-540-21123-3

Vol. 153. J. Fulcher, L.C. Jain (Eds.)
Applied Intelligent Systems, 2004
ISBN 3-540-21153-5

Vol. 154. B. Liu
Uncertainty Theory, 2004
ISBN 3-540-21333-3

Vol. 155. G. Resconi, J.L. Jain
Intelligent Agents, 2004
ISBN 3-540-22003-8

Vol. 156. R. Tadeusiewicz, M.R. Ogiela
Medical Image Understanding Technology, 2004
ISBN 3-540-21985-4

Vol. 157. R.A. Aliev, F. Fazlollahi, R.R. Aliev
Soft Computing and its Applications in Business and Economics, 2004
ISBN 3-540-22138-7

Vol. 158. K.K. Dompere
Cost-Benefit Analysis and the Theory of Fuzzy Decisions - Identification and Measurement Theory, 2004
ISBN 3-540-22154-9

Vol. 159. E. Damiani, L.C. Jain, M. Madravia
Soft Computing in Software Engineering, 2004
ISBN 3-540-22030-5

Vol. 160. K.K. Dompere
Cost-Benefit Analysis and the Theory of Fuzzy Decisions - Fuzzy Value Theory, 2004
ISBN 3-540-22161-1

Vol. 161. N. Nedjah, L. de Macedo Mourelle (Eds.)
Evolvable Machines, 2005
ISBN 3-540-22905-1

Vol. 162. N. Ichalkaranje, R. Khosla, L.C. Jain
Design of Intelligent Multi-Agent Systems, 2005
ISBN 3-540-22913-2

Vol. 163. A. Ghosh, L.C. Jain (Eds.)
Evolutionary Computation in Data Mining, 2005
ISBN 3-540-22370-3

Vol. 164. M. Nikravesh, L.A. Zadeh, J. Kacprzyk (Eds.)
Soft Computing for Information Prodessing and Analysis, 2005
ISBN 3-540-22930-2

Vol. 165. A.F. Rocha, E. Massad, A. Pereira Jr.
The Brain: From Fuzzy Arithmetic to Quantum Computing, 2005
ISBN 3-540-21858-0

William E. Hart
N. Krasnogor
J. E. Smith (Eds.)

Recent Advances in Memetic Algorithms

Springer

William E. Hart
Sandia National Laboratories
Algorithms and
Discrete Mathematics Department
87185-1110 Albuquerque, NM
USA
E-mail: wehart@sandia.gov

Dr. J. E. Smith
Faculty of Computing, Engineering
and Mathematical Sciences
University of the West of England
Coldharbour Lane
BS16 1QY Bristol
United Kingdom

N. Krasnogor
Automatic Scheduling, Optimisation
and Planning Group
School of Computer Science and IT
University of Nottingham
Jubilee Campus
Wollaton Road
NG8 1BB Nottingham
United Kingdom
E-mail: nxk@cs.nott.ac.uk

ISSN 1434-9922
ISBN 3-540-22904-3 Springer Berlin Heidelberg New York

Library of Congress Control Number: 2004111139

Springer is a part of Springer Science+Business Media
springeronline.com

Printed in Germany

Typesetting: data delivered by editors
Cover design: E. Kirchner, Springer-Verlag, Heidelberg
Printed on acid free paper 62/3020/M - 5 4 3 2 1 0

Preface

Memetic algorithms are evolutionary algorithms that apply a local search process to refine solutions to hard problems. As such, Memetic algorithms are a particular class of global-local search hybrids. In these algorithms the global character of the search is given by the evolutionary nature of the approach while the local search aspect is usually performed by means of constructive methods, intelligent local search heuristics or other search techniques. Memetic algorithms have been successfully applied to hundreds of real-world problems and are the subject of intense scientific research both in academia and industry. The implementation of ever more sophisticated MAs has been made possible thanks to advances in computing capabilities, moreover, their use has spread to domains that range from the construction of optimal university exam timetables, to the prediction of protein structures and the optimal design of space-craft trajectories. The importance of Memetic Algorithms in both real-world applications and academic research has lead to the establishment of the series of international workshops on Memetic algorithms (WOMA)[1]. WOMA has served as a forum for the exchange of ideas and knowledge on Memetic Algorithms and by the time of the writing of this preface the fifth workshop on that series will take place in Birmingham, UK. As the co-founders of the WOMA series we felt that it was necessary to fill the gap that was present in the literature on metaheuristic optimisation in general and evolutionary optimisation in particular given by the lack of a book dedicated exclusively to Memetic Algorithms. The book that is in your hands represents our first attempt to fill that gap.

Recent Advances in Memetic Evolutionary Algorithms is the first book where Memetic Algorithms are the central topical matter. This book presents the reader with a rich gallery of works where the state of the art on Memetic Algorithms is presented. Each chapter was written by world experts on the subject. Readers will have the unique opportunity to have a coherent, inte-

[1] See www.cs.nott.ac.uk/~nxk

grated view on both good practice examples and new trends in optimisation technology based on these algorithms.

Researchers and postgraduate students in academia and research centers who are interested in search and optimisation technologies, metaheuristics, artificial intelligence, soft computing, combinatorial optimization, continuous optimization, global and local search , planning and decision making problems and strategies, operations research will find this book of enormous value. Similarly, undergraduate students who want to complement existing textbooks on artificial intelligence and modern heuristics will find a rich source of information on this powerful technique. More generally speaking, this book could be used to complement modules on evolutionary algorithms and metaheuristic optimisation. Practitioners in industry, engineering and science who need to know the state of the art on optimization techniques and managers and decision-makers who need powerful tools and techniques to make informed decisions within a variety of domains like scheduling, timetabling, VLSI design, fleet and vehicle routing, personnel rostering, drugs and molecular design and optimization, bioinformatics, telecommunication networks optimization, logistics, operations research, etc. will find this book a rich source of examples and good practices.

The book is structured in four parts. The first part, *Introduction to Memetic Algorithms*, contains an introductory chapter where MAs are briefly introduced and the most important algorithmic desing issues that are specific to these algorithms discussed. The chapter also sets the scene for the rest of the book by defining the terminology that is to be encountered in later chapters. This chapter also contains a discussion section where different hybridisation schemes are discussed.

The second part of the book, *Applications*, contains seven application oriented chapters. The first chapter in this section, by Katayama and Narihisa, applies a MA to the maximum diversity problem. The authors' MA is a combination of an evolutionary algorithm with both crossover and mutation , a repair mechanism and local search. In this paper the solutions to very large instances for this problem are reported for the first time. The chapter by Pelta and Krasnogor, presents the results of using an innovative fuzzy logic based local search framework (called FANS) in conjunction with Multimeme algorithms with the aim of predicting the structure of proteins in a simplified lattice model. This application chapter introduces two innovations to the Memetic Algorithm: the use of FANS as memes and the use of multiple memes (hence the name Multimeme) to make the search more robust. The following chapter by Prins and Bouchenoua extends the vehicle routing problem and the capacitated arc routing problem by a generalization called "Node, Edge and Arc Routing Problem". The authors present state of the art Memetic algorithm for this more general routing problem. The fourth chapter in the section explores a real world problem faced at a car manufacturing industry (BMW). Kñodler et.al. employ MAs for the solution of the optimal calibration of automotive internal combustion engines. Next, Smith demonstrate how the

co-evolution of MAs can be successfully applied to a molecular design problem. He shows how the scalability of MAs is improved by the co-evolution of the rules with which to perform local search. In the following chapter Yao and coworkers employ sophisticated hybrid evolutionary algorithm to solve a hard problem in Telecommunications. The Terminal Assignment Problem in communication networks has very many practical applications. New algorithms are proposed and benchmarked. In the last chapter of the applications section Areibi describes a family of MAs that can be used to optimise the design of VLSI circuits. Taken together, all these chapters represent a wide range of applications and showcase the impact that Memetic algorithms have had in a variety of engineering domains.

The third part of the book, *Methodological Aspects of Memetic Algorithms*, includes six chapters that explore the principles behind Memetic Algorithms (in some cases for specific applications). Krasnogor, in the first chapter of the third section, explores two techniques that can be use to improve the robustness of MAs, namely, the use of multiple local searchers and the availability of operators which (under certain circumstances) may accept moves that deteriorates the current solution with the aim of escaping deceptive local optima. The next chapter by Merz explores in detail the interplay between MAs, greedy operators, K-opt type of local searchers and fitness landscapes. In "Self-Assembling of Local Searchers in Memetic Algorithms" Krasnogor and Gustafson argue the case of the simultaneous search for solutions and solvers within MAs. They introduce the concept of Self-Assembling of local searchers an exemplify their use in two hard problems. The chapter by Sinha et.al. introduces a theoretical system-level framework for efficiently combining global searchers such as genetic and evolutionary algorithms with domain specific local searchers. In Burke and Landa Silva chapter the design principles that must be considered when engineering MAs for scheduling and timetabling are discussed in detail and an important review of literature is presented. The last chapter of the methodological part of the book by Knowles and Corne present an in-deep study of the role of Memetic Algorithms in Multi-objective optimisation and the way in which MAs must be design in order to produce good quality solutions for hard multi-objective problems. The chapter also points to an extensive literature.

The fourth and last part of the book, *Related Search Technologies*, contains two chapter. The first chapter by Wyatt and Bull employs an MA within a Learning Classifier System framework use to learn the characteristics of continuous-valued problem spaces. In the last chapter of the book, Comellas and Gallegos introduce a new metaheuristic called "Angels & Mortals" and exemplify its use on the $K-$coloring graph problem.

Albuquerque - United States , June 2004, *Bill Hart*
Nottingham - United Kingdom, June 2004, *Nat Krasnogor*
Bristol - United Kingdom, June 2004, *Jim Smith*

Contents

Part I

Introduction to Memetic Algorithms

Memetic Evolutionary Algorithms

W.E. Hart[1], N. Krasnogor[2], and J.E. Smith[3]

[1] Sandia National Laboratory
Albuquerque, New Mexico
USA
[2] Automatic Scheduling, Optimisation and Planning Group
School of Computer Science and IT
University of Nottingham, U.K.
http:\\www.cs.nott.ac.uk\~nxk
natalio.krasnogor@nottingham.ac.uk
[3] Faculty of Computing, Engineering and Mathematical Sciences,
University of the West of England,
Bristol BS16 12QY, U.K.
james.smith@uwe.ac.uk
http://www.cems.uwe.ac.uk/ jsmith

1 Summary

Memetic Evolutionary Algorithms (MAs) are a class of stochastic heuristics for global optimization which combine the parallel global search nature of Evolutionary Algorithms with Local Search to improve individual solutions. These techniques are being applied to an increasing range of application domains with successful results, and the aim of this book is both to highlight some of these applications, and to shed light on some of the design issues and considerations necessary to a successful implementation. In this chapter we provide a background for the rest of the volume by introducing Evolutionary Algorithms (EAs) and Local Search. We then move on to describe the synergy that arises when these two are combined in Memetic Algorithms, and to discuss some of the most salient design issues for a successful implementation. We conclude by describing various other ways in which EAs and MAs can be hybridized with domain-specific knowledge and other search techniques.

2 Introduction

Memetic Algorithms (MAs) are a class of stochastic global search heuristics in which Evolutionary Algorithms-based approaches are combined with local search techniques to improve the quality of the solutions created by evolution. MAs have proven very successful across a wide range of problem domains such

as combinatorial optimization [27], optimization of non-stationary functions [42], and multi-objective optimization [20] (see [29] for an extensive bibliography).

Methods for hybridizing EAs with local search have been given various names in research papers such as: *hybrid GAs, Baldwinian EAs, Lamarckian EAs, genetic local search algorithms*, and others. Moscato [3] coined the name *memetic algorithm* to cover a wide range of techniques where evolutionary-based search is augmented by the addition of one or more phases of local search.

The natural analogies between human evolution and learning, and EAs and artificial neural networks (ANNs) prompted a great deal of research into the use of MAs to evolve the structure of ANNs. ANNs were trained using back-propagation or similar means during the 1980s and early 1990s. However, research applying MAs to ANNs gave a great deal of insight into the role of learning, Lamarckianism, and the Baldwin effect to guide evolution (e.g. [8, 7, 8, 9, 10, 11, 12] amongst many others). This research reinforced the experience of "real-world" practitioners as to the usefulness of incorporating local search and domain-based heuristics within an EA framework.

Since then a number of PhD theses [14, 25, 15, 27, 16] have developed algorithmic analyses of MAs. These analyses and related empirical results demonstrate the potential impact of MAs, and in practice, many state-of-the-art EAs employ some element of hybridization using local search. Research in MAs is now sufficiently mature and distinct to have its own annual international workshop, and an extensive on-line bibliography of MA research is maintained at [29].

In this chapter we set the scene for the rest of this book by providing brief introductions to Evolutionary Algorithms (EAs) and Local Search (LS). We also discuss some of the issues which arise when hybridizing the two to create MAs. As our aim is to provide an overview, we cannot hope to give a detailed description of either EAs or the many LS methods available. There are wide variety of books discussing these methods that the user can read for further detail (e.g., see [17, 18]). The rest of this chapter is organized as follows:

- In Section 3 we provide a brief overview and historical background to the field of Evolutionary Algorithms, focusing particularly on their use as search and optimization techniques.
- In Section 4 we provide a brief introduction to local search and some related techniques.
- In Section 5 we discuss some of the motives and rationale underpinning the hybridization of EAs with other search technologies and motivate this book's focus on Memetic Algorithms. Our focus is on EA hybrids in which LS acts on the output of evolutionary operators, that is to say in which some form of "lifetime learning" or "plasticity" is incorporated into the "standard" evolutionary cycle..

- In Section 6 we discuss some of the design issues that must be considered when implementing an MA.
- Finally, in Section 7 we discuss the *structure* of evolutionary and memetic algorithms, and consider various places within the evolutionary cycle that other heuristics and or domain specific knowledge may be incorporated.

3 A Brief Introduction to Evolutionary Algorithms

The idea of applying Darwinian principles to automated problem solving dates back to the forties, long before the breakthrough of computers [19]. As early as 1948, Turing proposed "genetical or evolutionary search", and by 1962 Bremermann had actually executed computer experiments on "optimization through evolution and recombination". During the 1960s three different implementations of the basic idea were developed in different places. In the USA, Fogel, Owens, and Walsh introduced **evolutionary programming** [20, 21], while Holland called his method a **genetic algorithm** [22, 23, 24]. Meanwhile, in Germany, Rechenberg and Schwefel invented **evolution strategies** [25, 26]. For about 15 years these areas developed separately; but since the early 1990s they have been viewed as different representatives of a common technology that has come to be known as **evolutionary computing** (EC) [27, 28, 29, 30, 31]. In the early 1990s a fourth methodology following the same general ideas emerged, **genetic programming**, championed by Koza [32, 33, 34]. The contemporary terminology denotes the whole field by evolutionary computing, and the algorithms involved are termed **evolutionary algorithms**; evolutionary programming, evolution strategies, genetic algorithms, and genetic programming are subareas belonging to the corresponding algorithmic variants.

3.1 The Principal Metaphor

The common underlying idea behind different evolutionary algorithms is the same: given a population of individuals, mechanisms adapted from natural selection and genetic variation are used to evolve individuals with high fitness. Given a quality function to be maximized, we can randomly create a set of candidate solutions, i.e., elements of the function's domain, and apply the quality function as an abstract fitness measure – the higher the better. Based on this fitness, some of the better candidates are chosen to seed the next generation by applying recombination and/or mutation to them. Recombination is an operator applied to two or more selected candidates (the so-called parents) and results in one or more new candidates (the children). Mutation is applied to one candidate and results in one new candidate. Executing recombination and mutation leads to a set of new candidates (the offspring) that compete – based on their fitness (and possibly age)– with the old ones for a place in the next generation. This process can be iterated until a candidate

with sufficient quality (a solution) is found or a previously set computational limit is reached.

In this process there are two fundamental forces that form the basis of evolutionary systems:

- Variation operators (recombination and mutation) create the necessary diversity and thereby facilitate novelty.

- Selection filters, and induces constraints on, candidate solutions.

The combined application of variation and selection generally leads to improving fitness values in consecutive populations. It is easy to view such an evolutionary process as optimizing by iteratively generating solutions with increasingly better values. Alternatively, evolution it is often seen as a process of adaptation. From this perspective, the fitness is not seen as an objective function to be optimized, but as an expression of environmental requirements. Matching these requirements more closely implies an increased viability, reflected in a higher number of offspring. The evolutionary process makes the population increasingly better at being adapted to the environment.

The general scheme of an evolutionary algorithm is shown in Figure 1 in a pseudocode fashion. It is important to note that many components of evolutionary algorithms are stochastic. During selection, fitter individuals have a higher chance to be selected than less fit ones, but typically even the weak individuals have a chance to become a parent or to survive. For recombination of individuals the choice of which pieces will be recombined is random. Similarly for mutation, the pieces that will be mutated within a candidate solution, and the new pieces replacing them, are chosen randomly.

```
:
Begin
  INITIALIZE population with random candidate solutions;
  EVALUATE each candidate;
  Repeat Until ( TERMINATION CONDITION is satisfied ) Do
    1 SELECT parents;
    2 RECOMBINE pairs of parents;
    3 MUTATE the resulting offspring;
    4 EVALUATE new candidates;
    5 SELECT individuals for the next generation;
  endDo
End.
```

Fig. 1. The general scheme of an evolutionary algorithm in pseudocode.

3.2 Components of an EA

Representation

Solutions to the problem being solved are usually referred to as **phenotypes**. Phenotypes are indirectly manipulated by the EA variation and selection operators by virtue of being encoded in **genotypes**. Genotypes within the EA population are the objects upon which the operators act.

The **representation** employed by an EA can thus be represented by a ternary relation $R = (\mathcal{P}, \mathcal{G}, \mathcal{M})$ that specifies the relationship between the space of phenotypes,$\mathcal{P}$, and the space of genotypes $\mathcal{G}$. The mapping $\mathcal{M}$ is a function with domain in $\mathcal{G}$ and range in $\mathcal{P}$ that provides the "interpretation" of the representation. For example, a phenotypic space of real values, $\mathcal{P} \equiv \Re$, can be easily encoded by a binary genotypic search space, $\mathcal{G} \equiv \{0,1\}^+$, using as a mapping $\mathcal{M}$ a gray coding. That is, the gray coding defines *how* binary strings are to be mapped to, or interpreted into, real values.

It is important to understand that the phenotype space can be very different from the genotype space, and thus the EA designer must ensure that the (optimal) solution to the problem at hand – a phenotype – can be represented in the given genotype space.

The common EC terminology uses many synonyms for naming the elements of these two spaces. On the side of the original problem context, **candidate solution**, phenotype, and **individual** are used to denote points of the space of possible solutions. This space itself is commonly called the **phenotype space**. On the side of the EA, genotype, **chromosome**, and again individual can be used for points in the space where the evolutionary search actually takes place. This space is often termed the **genotype space**. There are also many synonymous terms for the elements of individuals. A placeholder is commonly called a variable, a **locus** (plural: loci), a position, or – in a biology-oriented terminology – a **gene**. An object on such a place can be called a value or an **allele**.

Evaluation Function

The **evaluation function** represents the quality of an individual. It forms the basis for selection, and thereby it facilitates improvements. More accurately, it defines what improvement means. From the problem-solving perspective, it provides the measure with which alternative solutions can be compared. The evaluation function is commonly called the **fitness function** in EC. Problems typically solved by EAs are optimization problems, which are specified with an **objective function**. For minimization problems, an evaluation function is commonly formed by negating the objective function.

Population

A **population** is a set of possible solutions. Specifically, a population is a multiset of genotypes.[4] In some sophisticated EAs, a population has an additional spatial structure, with a distance measure or a neighborhood relation. In such cases the additional structure has also to be defined to fully specify a population. **Initialization** is kept simple in most EA applications: The first population is seeded by (uniformly) randomly generated individuals. However as we shall see in the next section, and succeeding chapters, there may be practical advantages to non-random initialization.

The **diversity** of a population is a measure of the number of *different* solutions present. Common diversity measures are the number of different fitness values present, the number of different phenotypes present, the number of different genotypes, and statistical measures such as entropy. Note that only one fitness value does not necessarily imply only one phenotype is present, and in turn only one phenotype does not necessarily imply only one genotype. The reverse is, however, not true: one genotype implies only one phenotype and fitness value.

As opposed to variation operators that act on the one or two parent individuals, the selection operators (parent selection and survivor selection) work at population level. In general, they take the whole current population into account. For instance, the best individual *of the given population* is chosen to seed the next generation, or the worst individual *of the given population* is chosen to be replaced by a new one. In almost all EA applications the population size is constant and does not change during the evolutionary search.

Parent Selection Mechanism

The role of **parent selection** or **mating selection** is to distinguish among individuals based on their quality to allow the better individuals to become parents of the next generation. An individual is a **parent** if it has been selected to undergo variation in order to create offspring. Together with the survivor selection mechanism, parent selection is responsible for pushing quality improvements. In EC, parent selection is typically probabilistic. Thus, high-quality individuals get a higher chance to become parents than those with low quality. Nevertheless, low-quality individuals are often given a small positive chance, which helps the evolutionary search avoid getting stuck in a local optimum.

Survivor Selection Mechanism

The role of **survivor selection** or **environmental selection** is to distinguish among individuals, based on their quality, to identify those that will

[4] A multiset is a set where multiple copies of an element are possible.

be used in the next generation. The survivor selection mechanism is called after the offspring of the selected parents are created. As mentioned above, in EC the population size is almost always constant, thus a choice has to be made on which individuals will be allowed in the next generation. For this reason survivor selection is also often called **replacement** or replacement strategy. This selection is usually based on their fitness values, favoring those with higher quality, although the concept of age is also frequently used. As opposed to parent selection, which is typically stochastic, survivor selection is often deterministic, for instance, ranking the unified multiset of parents and offspring and selecting the top segment (fitness biased), or selecting only from the offspring (age biased).

Variation Operators - Mutation

The role of **variation operators** is to create new individuals from old ones. In the corresponding phenotype space this amounts to generating new candidate solutions. Variation operators in EC are divided into two types based on the number of objects that they take as inputs.

Mutation, a **unary** variation operator, is applied to one genotype and delivers a (slightly) modified mutant: a **child** or **offspring** genotype. A mutation operator is always stochastic: its output – the child – depends on the outcomes of a series of random choices. It should be noted that an arbitrary unary operator is not necessarily seen as mutation. A problem-specific heuristic operator acting on one individual could be termed as mutation for being unary. However, in general mutation denotes a random, unbiased change. Thus heuristic unary operators can be distinguished from mutation in most cases.

It is important to note that variation operators are representation dependent. That is, for different representations different variation operators have to be defined. For example, if genotypes are bit-strings, then inverting a 0 to a 1 (1 to a 0) can be used as a mutation operator. However, if we represent possible solutions by tree-like structures another mutation operator is required.

Variation Operators - Recombination

Recombination (or **crossover**) is (usually) a **binary** variation operator. As the names indicate, such an operator merges information from two parent genotypes into one or two offspring genotypes. Like mutation, recombination is a stochastic operator: the choice of what parts of each parent are combined, and the way these parts are combined, depend on random events. Recombination operators with a higher arity (using more than two parents) are sometimes possible and easy to implement, but have no biological equivalent. Perhaps this is why they are not commonly used, although several studies indicate that they have positive effects on the evolution [35].

The principle behind recombination is simple – by mating two individuals with different but desirable features, it may be possible to produce an offspring that combines both of those features. This principle has a strong supporting case: it is one which has been successfully applied for millennia by breeders of plants and livestock to produce species that give higher yields or have other desirable features. Evolutionary algorithms create a number of offspring by random recombination, and accept that some will have undesirable combinations of traits, most may be no better, or even worse, than their parents, and hope that some will have improved characteristics. As with mutation, recombination operators in EAs are representation dependant. Whether to apply crossover (mutation) or not is a stochastic decision with a non-zero probability of the operator(s) not being applied.

4 A Brief Introduction to Local Search

Local search is a search method that iteratively examines the set of points in a neighborhood of the current solution and replace the current solution with a better neighbor if one exists. In this section we give a brief introduction to local search in the context of memetic algorithms. For more information there are a number of books on optimization that cover local search in more detail, such as [18]. A local search algorithm can be illustrated by the pseudocode given in Figure 2.

There are three principal components that affect the workings of this local search algorithm.

- The **pivot rule** defines the criteria for accepting an improving point. A **steepest ascent** pivot rule terminates the inner loop only after the entire neighborhood $n(i)$ has been searched, (i.e., $count = |n(i)|$). A **greedy ascent** (or *first ascent*) pivot rule terminates the inner loop as soon as an improvement is found (i.e., $((count = |n(i)|)\ or\ (best \neq i))$). In practice it is sometimes necessary to only consider a randomly drawn sample of size $N \ll |n(i)|$ if the neighborhood is too large to search.
- The **depth** of the local search defines the termination condition for the outer loop. This lies in the continuum between only one improving step being applied ($iterations = 1$) to the search continuing to local optimality where all the neighboors of a solution i have been explored but no one of them found to be better: $((count = |n(i)|)\ and\ (best = i))$. Considerable attention has been paid to studying the effect of changing this parameter within MAs [14, 25], and it can be shown to have an effect on the performance of the local search algorithm, both in terms of time taken, and in the quality of solution found. Furthermore, the impact on computational complexity of various pivot rules have been studied both in the context of local search [36, 37] and within MAs [25].

```
:
Begin
  /* given a starting solution i and a neighborhood function n */
  set best = i;
  set iterations = 0;
  Repeat Until ( depth condition is satisfied ) Do
    set count = 0;
    Repeat Until ( pivot rule is satisfied ) Do
      generate the next neighbor j ∈ n(i);
      set count = count + 1;
      If (f(j) is better than f(best)) Then
        set best = j;
      endIf
    endDo
    set i = best;
    set iterations = iterations + 1;
  endDo
End.
```

Fig. 2. Pseudocode of a local search algorithm.

- The **neighborhood generating function**, $n(i)$, defines a set of points that can be reached by the application of some move operator to the point i. The application of a neighborhood generating function can be represented as a graph $G = (v, e)$, where the set of vertices v are the points in the search space, and the edges relate to applications of the move operator; $e_{ij} \in G \iff j \in n(i)$. The provision of a scalar fitness value f defined over the search space means that we can consider the graphs defined by different move operators as fitness landscapes [14, 17, 15]. Merz and Freisleben [28] present a number of statistical measures that can be used to characterize fitness landscapes, and that have been proposed by various authors as potential measures of problem difficulty. They show that the choice of move operator can have a dramatic effect on the efficiency and effectiveness of the local search, and hence of the resultant MA.

In some cases, domain-specific information may be used to guide the choice of neighborhood structure within local search algorithms. However, it has recently been shown that the optimal choice of operators can be not only instance specific within a class of problems [28, pp. 254–258], but when incorporated in an MA, it can be dependent on the state of the evolutionary search [26]. Changing the neighborhood operator during search (eg. [30]) may provide a means of progression in cases where points were locally optimal for a given neighborhood operator because a point that is locally optimal with respect to one neighborhood structure may not be with respect to another

(unless they are globally optimal). This observation has also been the guiding principle behind *variable neighborhood search* algorithm [49].

The local search method presented in Figure 2 is fairly simplistic, but local search is a central idea in most successful global search methods. The simplest of these is the so-called "multi-start local search", in which the algorithm is run repeatedly from randomly generated solutions. An elaboration on this is Iterated Local search [45], where a new search is begun from a perturbed version of the end-point of the previous one. Iterated local search attempts to traverse a succession of "nearby" local optima, which is often quite effective in practice.

Perhaps more relevant to this book are two well known heuristics based on local search, namely Tabu Search [46] and Simulated Annealing [47]. Giving a full description of these techniques is beyond the scope of this book, but in essence both modify the pivot rule. Tabu Search does so such that points in the neighborhood of the current solution which have been previously considered are not (generally) eligible to be accepted, whereas in Simulated Annealing a move to an inferior neighbor is permitted with some probability dependent on the fitness difference. Both of these have been used with noticeable success, both as heuristics in their own right, and as improvement methods within Memetic algorithms.

5 Hybridizing EAs

As suggested above, there are a number of benefits that can be achieved by combining the global search of EAs with local search or other methods for improving or refining an individual solution. In this section we give an overview of some of the theoretical and practical motivations for such hybridizations, before presenting one possible framework for Memetic Algorithms.

5.1 Motives

There are a number of factors that motivate the hybridization of evolutionary algorithms with other techniques.

- Many complex problems can be decomposed into a number of parts, for some of which exact methods (or very good heuristics) may already be available. In these cases it makes sense to use a combination of the most appropriate methods for different subproblems. In some cases this may take the form of using the EA either as a post or pre-processor for other algorithms, or incorporating instance specific knowledge into "greedy" variation operators as will be discussed in Section 7. However it is also frequently possibly to use this knowledge to define local search operators (or existing solution improvement techniques) within an evolutionary algorithm.

- Successful and efficient all-purpose "black-box" problem solvers do not exist. The rapidly growing body of empirical evidence and some theoretical results, such as the No Free Lunch (NFL) theorem [109][5] strongly support this view. From an Evolutionary Computing perspective, this implies that EAs are not the holy grail for global search. Experience suggests that in fact the competence of an EA in any given domain depends on the amount of problem-specific knowledge incorporated within it. In practice we frequently apply an evolutionary algorithm to a problem where there is a considerable amount of hard-won user experience and knowledge available. In such cases performance benefits can often arise from utilizing this information in the form of specialist operators (eg. variation and local search) and/or good solution initializations. In these cases it is commonly experienced that the combination of an evolutionary and a heuristic method – a **hybrid EA** – that somehow encapsulates domain specific information performs better than either of its "parent" algorithms alone.

- Although EAs are very good at rapidly identifying good areas of the search space (**exploration**), they are often less good at refining near-optimal solutions (**exploitation**). For example, when a GA is applied to the "One-Max" problem, near-optimal solutions are quickly found but convergence to the optimal solution is slow because the choice of which genes are mutated is random.[6] Thus EA hybrids can search more efficiently by incorporating a more systematic search in the vicinity of "good" solutions. For example, a bit-flipping hill-climber could be quickly applied within each generation for One-Max to ensure fast convergence.

- In practice, many problems have a set of constraints associated with them, and local search or other heuristics can be used as a means of "repairing" infeasible solutions generated by standard variation operators. This is often far simpler and more effective than attempting to find a specialized representation and set of variation operators which ensure the feasibility of all offspring.

- Dawkin's idea of **"memes"**[11] is often used as a motivation for hybridization. Memes can be viewed as units of "cultural transmission" in the same way that genes are the units of biological transmission. They are selected for replication according to their perceived utility or popularity, and then copied and transmitted via inter-agent communication.

[5] The NFL and its implications are still a matter of current debate, for the present we interpret it as stating that all stochastic algorithms have the same performance when averaged over all discrete problems.

[6] The One-Max problem is a binary maximization problem, where the fitness is simply the count of the number of genes set to "1".

> Examples of memes are tunes, ideas, catch-phrases, clothes fashions, ways of making pots or of building arches. Just as genes propagate themselves in the gene pool by leaping from body to body via sperm or eggs, so memes propagate themselves in the meme pool by leaping from brain to brain via a process which, in the broad sense, can be called imitation [11, p. 192].

Since the idea of memes was first proposed by Dawkins, it has been extended by other authors (eg., [6, 13, 15, 2]). From the point of view of the study of adaptive systems as optimization techniques, memetic theory (see for example papers in [54]) provides with a rich set of tools and metaphors to work with. In the context of memetic theory an EA keeps a population of agents composed by both genotypes and memes. As in standard EA, genotypes represent solutions to a particular problem while memes represent "strategies" on how to improve those solutions. It is the memes abilities to transform a candidate solution into (hopefully) a better one that is of direct interest in the context of optimisation. The idea of memes as representing alternative improvement strategies agents can harness (implemented, for example, as distinct local searchers) is what motivated us to propose in [22] the co-evolution of memes and genes and to develop later in [25] the concept of multimeme, self-generating memetic algorithms[56],[57] and co-evolving memetic algorithms [58].

5.2 Memetic Algorithms

The most common use of hybridization within EAs, and that which fits best with Dawkin's concept of the meme, is via the application of one or more phases of improvement to individual members of the population within each generation of an EA. In the simplest design, local search is applied to individuals created by mutation or recombination. A more general form can be described by the pseudocode given in Figure 3 (see also Figure 4), although practitioners typically choose to only apply local search once to the offspring, and sometimes to avoid the use of mutation entirely when using local search.

6 Design Issues for Memetic Algorithms

So far we have discussed the rationale for the use of problem-specific knowledge or heuristics within EAs, and some possible ways in which this can be done. However, as ever we must accept the caveat that like any other technique, MAs are not some "magic solution" to optimization problems, and care must be taken in their implementation. In the sections below we briefly discuss some of the issues that have arisen from experience and theoretical reasoning.

```
:
Begin
  INITIALIZE population;
  EVALUATE each candidate;
  Repeat Until ( TERMINATION CONDITION is satisfied ) Do
    SELECT parents;
    RECOMBINE to produce offspring;
    EVALUATE offspring;
    IMPROVE offspring via Local Search;
    MUTATE offspring;
    EVALUATE offspring;
    IMPROVE offspring via Local Search;
    SELECT individuals for next generation;
  endDo
End.
```

Fig. 3. Pseudocode for a simple memetic algorithm

6.1 Lamarckianism and the Baldwin Effect

The local search methods described above assume that the current incumbent solution is always replaced by the fitter neighbor when found. Within a memetic algorithm, we can consider the local search stage to occur as an improvement, or developmental learning phase within each generation. As such, we can consider whether the changes (*acquired traits*) made to an individual should be kept, or whether the resulting improved fitness should be awarded to the original (pre-local search) member of the population.

The issue of whether acquired traits could be inherited by an individual's offspring was a major issue in nineteenth century, and Lamarck was a strong proponent of this inheritance mechanism. However, the **Baldwin effect** [59] suggests a mechanism whereby evolutionary progress can be guided towards favorable adaptation without this type of inheritance. Although modern theories of genetics strongly favor the latter viewpoint, the design of MAs can employ either Lamarckian or Baldwinian inheritance schemes. MAs are referred to as Lamarckian if the result of the local search stage replaces the individual in the population, and Baldwinian if the original member is kept, but has as its fitness the value belonging to the outcome of the local search process. In a classic early study, Hinton and Nowlan [8] showed that the Baldwin effect could be used to improve the evolution of artificial neural networks, and a number of researchers have studied the relative benefits of Baldwinian versus Lamarckian algorithms [8, 9, 10, 11, 12]. In practice, most recent work has tended to use either a pure Lamarckian approach, or a probabilistic combination of the two approaches, such that the improved fitness is always used, and the improved individual replaces the original with a given probability.

6.2 Preservation of Diversity

The problem of premature convergence, whereby the population converges around some suboptimal point, can be particularly problematic for MAs. If the local search is applied until each point has been moved to a local optimum, then this can lead to a loss of diversity within the population unless new local minima are constantly identified. Alternatively, even if local search is terminated before local optimality, an induced search space with wide basins of attractions could also result in premature convergence to the suboptimal solution at the center of a wide basin of attraction. A number of approaches have been developed to combat this problem:

- when initializing the population with known good individuals, only using a relatively small proportion of them,
- applying local search to a small fraction of the population (which helps ensure that the rest of the population is diverse),
- using recombination operators that are designed to preserve diversity,
- using multiple local searchers, where each one induces a different search space with distinct local optima (eg. [26, 12]);
- modifying the selection operator to prevent duplicates (e.g. as in CHC [31]), and
- using a fuzzy criteria, that explicitly controls diversity, as the pivot rule in the local search stage (eg. [12], 5).
- modifying the selection operator, or local search acceptance criteria, to use a Boltzmann method so as to preserve diversity (eg. III).

This last method bears natural analogies to simulated annealing [62, 47], where worsening moves can be accepted with nonzero probability to aid escape from local optima. A promising method that tackles the diversity issue explicitly is proposed in [24], where during the local search phase a less-fit neighbor may be accepted with a probability that increases exponentially as the range of fitness values in the population decreases:

$$P(accept) = \begin{cases} 1 & \text{if } \Delta E > 0, \\ e^{k* \frac{\Delta E}{F_{max} - F_{avg}}}, & \text{otherwise,} \end{cases}$$

where k is a normalization constant and we assume a maximization problem, $\Delta E = F_{\text{neighbour}} - F_{\text{original}}$.

6.3 Choice of Operators

Probably the most important factor in the design of a MA is the choice of improving heuristic or local search move operator, that is to say, the way that the set of neighboring points to be examined when looking for an improved solution is generated.

There has been a large body of theoretical and empirical analysis of the utility of various statistical measures of landscapes for predicting problem difficulty. The interested reader can find a good summary in [64]. Merz and Freisleben [28] consider a number of these measures in the context of memetic algorithms, and show that the choice of move operator can have a dramatic effect on the efficiency and effectiveness of the local search, and hence of the resultant MA.

One recent result of particular interest to the practitioner is Krasnogor's formal proof that, in order to reduce the worst-case run times, it is necessary to choose a local search method whose move operator is not the same as those of the recombination and mutation operators [25]. This formalizes the intuitive point that within a MA recombination, and particularly mutation, have valuable roles in generating points that lie in different basins of attraction with respect to the local search operator. This diversification is best done either by an aggressive mutation rate, or preferably by the use of a variation operators that have different neighborhood structures.

In general then, it is worth giving careful consideration to the choice of move operators used when designing a MA: for example, using 2-opt for a TSP problem might yield better improvement if not used in conjunction with the "inversion" mutation operator which picks a subtour at random and reverses it. The reason for that is that a genotypic inversion induces (a subspace of) the phenotypic effect of the 2-exchange move operator which is at the heart of 2-opt local searcher.

In some cases, domain-specific information may be used to guide the choice of neighborhood structure within the local search algorithms. However, as we noted earlier, the optimal choice of operators can be not only instance specific within a class of problems but also dependant on the state of the evolutionary search.

One simple way to surmount these problems is the use of *multiple* local search operators in tandem. An example of this can be seen in [30], where a range of problem specific move operators, such as local stretches, rotations and reflections, each tailored to different stages of the protein folding process, are used for a protein structure prediction problem within the context of what is called a *multimemetic algorithm* [26].

The use of a set of possible local search strategies is analogous to Dawkin's memes. The extension of this approach to allow the adaptation of the local search "memes" in the form of a coevolving population, and the implications for search is currently under way in different research groups [22, 65, 22, 37, 68, 58, 69, 56, 57].

6.4 Use of Knowledge

A final point that might be taken into consideration when designing a MA concerns the use and reuse of knowledge gained during the optimization process. One possible hybridization that explicitly uses knowledge about points

already searched to guide optimization is with **tabu search** [46]. In this algorithm a "tabu" list of visited points is maintained, which the algorithm is forbidden to return to. Such methods appear to offer promise for maintaining diversity. Similarly, it is easy to imagine extensions to the Boltzmann acceptance/selection schemes that utilize information about the spread of genotypes in the current population, or even past populations, when deciding whether to accept new solutions.

6.5 Specific Considerations for Continuous Domains

The design of MAs for continuous domains is complicated by several factors. Effective search requires the use of different *search scales* for global and local search. It is not always possible to determine whether a solution is locally optimal. Relatively long local searches may be needed to ensure convergence to local optima (especially if gradient information is unavailable). Although many different local search methods have been developed, they are general methods and thus it is not clear whether any given local search method is effective for a particular application.

Because of these considerations, the design of effective MAs for continuous domains can be quite different than for combinatorial problems. For example, in combinatorial domains it is not unusual to apply local search until a locally optimal solution is found. However, it is often unrealistic to assume that local search methods can quickly identify local minima within a continuous domain. This is often the case when applying derivative-free methods (e.g. the Nelder-Mead simplex method), but it may also be true when derivative information is available. Thus it is generally the case that local search is truncated based on a target balance between global and local search. Specifically, two main strategies have been used to achieve such a balance: (1) truncate local searches after a given number of iterations (or fitness evaluations) and (2) apply local search infrequently (e.g. with a fixed probability).

Although these hybridization strategies are quite effective in practice, they can make it difficult to ensure convergence for these MAs. Although general conditions on the mutation and recombination operators can be enforced to ensure global convergence [70], these convergence results provide little insight into the efficacy of local search. Gradient-based methods can be applied to generate stationary-points (using first-order information) or locally-optimal points (using second-order information), assuming that local search is not truncated after a given number of iterations. However, in many applications derivative-free methods are applied for which the search is truncated. To our knowledge, MAs based on evolutionary pattern search is the only class of MAs for which the convergence of tandem derivative-free local searches within the MA can be ensured [71].

7 Other Hybridization Possibilities

Although our working definition of MAs has been restricted to those methods that incorporate some form of improvement mechanism acting on the output of the evolutionary variation operators, there are a number of other ways in which an EA or MA can be used in conjunction with other operators and/or domain-specific knowledge. This is illustrated in Figure 4.

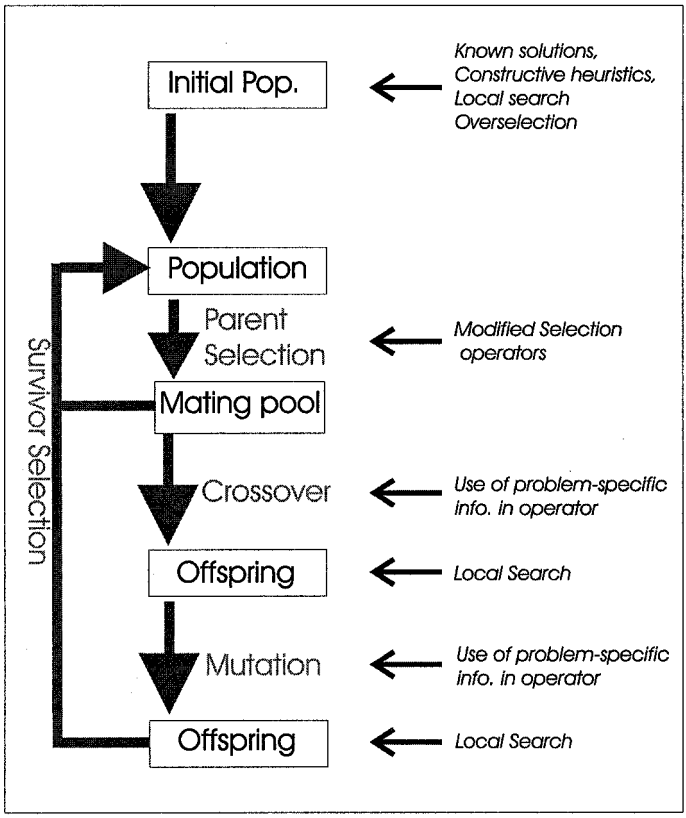


Fig. 4. Possible places to incorporate knowledge or other operators within a single generation.

7.1 Intelligent Initialization

The most obvious way in which existing knowledge about the structure of a problem or potential solutions can be incorporated into an EA is in the initialization phase. In many cases the EA will make rapid initial progress,

which raises questions about the value of expending effort creating a good initial population, however starting the EA by using existing solutions can offer interesting benefits:

1. It is possible to avoid "reinventing the wheel" by using existing solutions. Preventing waste of computational efforts can yield increased efficiency (speed).
2. A nonrandom initial population can direct the search into particular regions of the search space that contain good solutions. Biasing the search can result in increased effectiveness (quality of end solution).
3. All in all, a given total amount of computational effort divided over heuristic initialization and evolutionary search might deliver better results than spending it all on "pure" evolutionary search, or an equivalent multistart heuristic.

There are a number of possible ways in which the initialization function can be changed from simple random creation, such as:

- **Seeding** the population with one or more previously known good solutions arising from other techniques.
- In **selective initialization** a large number of random solutions are created and then the initial population is selected from these. Bramlette [72] suggests that this should be done as a series of N k-way tournaments rather than by selecting the best N from $k \cdot N$ solutions. Other alternatives include selecting a set based not only on fitness but also on diversity so as to maximize the coverage of the search space.
- Performing a local search starting from each member of initial population, so that the initial population consists of a set of points that are locally optimal with respect to some move operator.
- Using one or more of the above methods to identify one (or possibly more) good solutions, and then cloning them and applying mutation at a high rate (**mass mutation**) to produce a number of individuals in the vicinity of the start point.

These methods have been tried and have exhibited performance gains for certain problems. However, the important issue of providing the EA with sufficient diversity for evolution to occur must also be considered. In [73] Surry and Radcliffe examined the effect of varying the proportion of the initial population of a GA that was derived from known good solutions. Their conclusions were:

- The use of a small proportion of derived solutions in the initial population aided genetic search.
- As the proportion was increased, the *average* performance improved.
- The *best* performance came about from a more random initial population.

In other words, as the proportion of solutions derived from heuristics used increased, so did the mean performance, but the variance in performance

decreased. This meant that there were not the occasional really good runs resulting from the EA searching completely new regions of space and coming up with novel solutions. For a certain type of problems, such as design problems, this is an undesirable property.

7.2 Hybridization During Genotype to Phenotype Mapping

A widely used hybridization of memetic algorithms with other heuristics is during the genotype–phenotype mapping $\mathcal{M}$ prior to evaluation. This approach, where the EA is used to provide the inputs controlling the application of another heuristic, is frequently used and similar approaches have been used to great effect for timetabling and scheduling problems [74], and in the "sector first–order second" approach to the vehicle routing problem [75].

There is a common thread to all of these approaches, which is to make use of existing heuristics and domain information wherever possible. The role of the EA is often that of enabling a less biased application of the heuristics, or of problem decomposition, so as to permit the use of sophisticated, but badly scaling heuristics when the overall problem size would preclude their use.

7.3 Hybridization Within Variation Operators: Intelligent Crossover and Mutation

A number of authors have proposed so-called "intelligent" variation operators, which incorporate problem- or instance-specific knowledge. To give a simple example, if a binary-coded GA is used to select features for use in another classification algorithm, one might attempt to bias the search towards more compact features sets via the use of a greater probability for mutating from the allele value "use" to "don't use" rather than vice versa. A related approach can be seen in [76], where genes encode for microprocessor instructions, which group naturally into sets with similar effects. The mutation operator was then biased to incorporate this expert knowledge, so that mutations were more likely to occur between instructions in the same set than between sets.

A slightly different example of the use of problem-specific (rather than instance-specific) knowledge can be seen in the modified one-point crossover operator used for protein structure prediction in [77]. Here the authors realized that the heritable features being combined by recombination were folds, or fragments of three-dimensional structure. A property of the problem is that during folding protein structures can be free to rotate about peptide bonds. The modified operator made good use of this knowledge by explicitly testing all the possible different orientations of the two fragments, (accomplished by trying all the possible allele values in the gene at the crossover point) in order to find the most energetically favorable. If no feasible conformation was found, then a different crossover point was selected and the process repeated. This could be seen as a simple example of the incorporation of a local search phase into the recombination operator, but in practice the nature of the models

used is such that generally these approaches only need to consider partial solutions when deciding whether an offspring is feasible. Note that this should be distinguished from the simpler "crossover hill-climber" proposed in [15], in which all of the l-1 possible offspring arising from one-point crossover are constructed and the best chosen.

Operators can be modified in a complex manner to incorporate highly specific heuristics, which makes use of instance-specific knowledge. A good example of this is Merz and Friesleben's distance-preserving crossover (DPX) operator for the TSP [78]. This operator has two motivating principles: making use of instance specific knowledge, whilst at the same time preserving diversity within the population to prevent premature convergence. Diversity is maintained by ensuring that the offspring inherits all of the edges common to both parents, but none of the edges that are present in only one parent. The "intelligent" part of the operator comes from the use of a nearest-neighbor heuristic to join together the subtours inherited from the parents, thus explicitly exploiting instance-specific edge length information. It is easy to see how this type of scheme could be adapted to other problems, via the use of suitable heuristics for completing the partial solutions after inheritance of the common factors from both parents.

It should be noted that under our working definition of MAs, the use of such "intelligent" operator within an EA does not generally on its own constitute a MA, since they use instance-specific knowledge to guide the construction of *partial* solutions. This can be contrasted with the use of local search acting on offspring, where a neighborhood of *complete* solutions is examined and an improved solution accepted.

8 Conclusions

In this chapter we gave a gentle introduction to Memetic Evolutionary Algorithms and role they play as complements to pure Evolutionary Algorithms and pure Local Search. We briefly discussed the historical context of MAs, and we gave the motivation behind the use and research on this important brand of global-local search hybrids. We also mentioned some of the design principles a practitioner needs to take into consideration when designing Memetic Algorithms for new domains.

References

1. Merz, P.: Memetic Algorithms for Combinatorial Optimization Problems: Fitness Landscapes and Efective Search Strategies. PhD thesis, Department of Electrical Engineering and Computer Science, University of Siegen, Germany (2000)

2. Vavak, F., Fogarty, T., Jukes, K.: A genetic algorithm with variable range of local search for tracking changing environments. In Voigt, H.M., Ebeling, W., Rechenberg, I., Schwefel, H.P., eds.: Proceedings of the 4th Conference on Parallel Problem Solving from Nature. Number 1141 in Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, New York (1996) 376–385
3. Knowles, J., Corne, D.: A comparative assessment of memetic, evolutionary and constructive algorithms for the multi-objective d-msat problem. In: Gecco-2001 Workshop Program. (2001) 162–167
4. Moscato, P.: Memetic algorithms' home page, visited july 2003: http://www.densis.fee.unicamp.br/~moscato/memetic_home.html (2003)
5. Moscato, P.: On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical Report Caltech Concurrent Computation Program Report 826, Caltech, Caltech, Pasadena, California (1989)
6. Hinton, G., Nowlan, S.: How learning can guide evolution. Complex Systems **1** (1987) 495–502
7. Bull, L., Fogarty, T.: An evolutionary strategy and genetic algorithm hybrid: An initial implementation and first results. In Fogarty, T., ed.: Evolutionary Computation: Proceedings of the 1994 AISB Workshop on Evolutionary Computing, Springer, Berlin, Heidelberg, New York (1994) 95–102
8. Houck, C., Joines, J., Kay, M., Wilson, J.: Empirical investigation of the benefits of partial Lamarckianism. Evolutionary Computation **5** (1997) 31–60
9. Mayley, G.: Landscapes, learning costs and genetic assimilation. Evolutionary Computation **4** (1996) 213–234
10. Turney, P.: How to shift bias: lessons from the Baldwin effect. Evolutionary Computation **4** (1996) 271–295
11. Whitley, L., Gordon, S., Mathias, K.: Lamarkian evolution, the Baldwin effect, and function optimisation. In Davidor, Y., Schwefel, H.P., Männer, R., eds.: Proceedings of the 3rd Conference on Parallel Problem Solving from Nature. Number 866 in Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, New York (1994) 6–15
12. Whitley, L., Gruau, F.: Adding learning to the cellular development of neural networks: evolution and the Baldwin effect. Evolutionary Computation **1** (1993) 213–233
13. Hart, W.: Adaptive Global Optimization with Local Search. PhD thesis, University of California, San Diego (1994)
14. Krasnogor, N.: Studies in the Theory and Design Space of Memetic Algorithms. PhD thesis, University of the West of England (2002)
15. Land, M.: Evolutionary Algorithms with Local Search for Combinatorial Optimization. PhD thesis, University of California, San Diego (1998)
16. Moscato, P.: Problemas de Otimizacão NP, Aproximabilidade e Computacão Evolutiva:Da Prática à Teoria. PhD thesis, Universidade Estadual de Campinas,Brasil (2001)
17. Eiben, A., Smith, J.: Introduction to Evolutionary Computing. Springer, Berlin, Heidelberg, New York (2003)
18. Aarts, E., Lenstra, J., eds.: Local Search in Combinatorial Optimization. Discrete Mathematics and Optimization. Wiley, Chichester, UK (1997)
19. Fogel, D., ed.: Evolutionary Computation: the Fossil Record. IEEE Press, Piscataway, NJ (1998)

20. Fogel, L., Owens, A., Walsh, M.: Artificial intelligence through a simulation of evolution. In Callahan, A., Maxfield, M., Fogel, L., eds.: Biophysics and Cybernetic Systems. Spartan, Washington DC (1965) 131–156
21. Fogel, L., Owens, A., Walsh, M.: Artificial Intelligence through Simulated Evolution. Wiley, Chichester, UK (1966)
22. De Jong, K.: An Analysis of the Behaviour of a Class of Genetic Adaptive Systems. PhD thesis, University of Michigan (1975)
23. Holland, J.: Genetic algorithms and the optimal allocation of trials. SIAM J. of Computing **2** (1973) 88–105
24. Holland, J.: Adaption in Natural and Artificial Systems. MIT Press, Cambridge, MA (1992) 1st edition: 1975, The University of Michigan Press, Ann Arbor.
25. Rechenberg, I.: Evolutionstrategie: Optimierung Technisher Systeme nach Prinzipien des Biologischen Evolution. Fromman-Hozlboog Verlag, Stuttgart (1973)
26. Schwefel, H.P.: Evolution and Optimum Seeking. Wiley, New York (1995)
27. Bäck, T.: Evolutionary Algorithms in Theory and Practice. Oxford University Press, Oxford, UK (1996)
28. Bäck, T., Fogel, D., Michalewicz, Z., eds.: Evolutionary Computation 1: Basic Algorithms and Operators. Institute of Physics Publishing, Bristol (2000)
29. Bäck, T., Fogel, D., Michalewicz, Z., eds.: Evolutionary Computation 2: Advanced Algorithms and Operators. Institute of Physics Publishing, Bristol (2000)
30. Eiben, A., Michalewicz, Z., eds.: Evolutionary Computation. IOS Press (1998)
31. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. 3rd edn. Springer, Berlin, Heidelberg, New York (1996)
32. Banzhaf, W., Nordin, P., Keller, R., Francone, F.: Genetic Programming: An Introduction. Morgan Kaufmann, San Francisco (1998)
33. Koza, J.: Genetic Programming. MIT Press, Cambridge, MA (1992)
34. Koza, J.: Genetic Programming II. MIT Press, Cambridge, MA (1994)
35. Eiben, A.: Multiparent recombination. [28] chapter 33.7 289–307
36. Johnson, D., Papadimitriou, C., Yannakakis, M.: How easy is local search. Journal of Computer And System Sciences **37** (1988) 79–100
37. Yannakakis, M.: Computational complexity. In Aarts, E., Lenstra, J., eds.: Local Search in Combinatorial Optimization, John Wiley & Sons Ltd. (1997) 19–55
38. Weinberger, E.D.: Correlated and Uncorrelated Fitness Landscapes and How to Tell the Difference. Biological Cybernetics **63** (1990) 325–336
39. Stadler, P.F.: Towards a Theory of Landscapes. In Lopéz-Peña, R., Capovilla, R., García-Pelayo, R., Waelbroeck, H., Zertuche, F., eds.: Complex Systems and Binary Networks. Volume 461 of Lecture Notes in Physics., Berlin, New York, Springer Verlag (1995) 77–163 SFI preprint 95-03-030.
40. Jones, T.: Evolutionary Algorithms, Fitness Landscapes and Search. PhD thesis, The University of New Mexico, Albuquerque, NM (1995)
41. Merz, P., Freisleben, B.: Fitness landscapes and memetic algorithm design. In Corne, D., Dorigo, M., Glover, F., eds.: New Ideas in Optimization. McGraw Hill, London (1999) 245–260
42. Krasnogor, N., Smith, J.: Emergence of profitable search strategies based on a simple inheritance mechanism. In Spector, L., Goodman, E., Wu, A., Langdon, W., Voigt, H.M., Gen, M., Sen, S., Dorigo, M., Pezeshk, S., Garzon, M., Burke,

E., eds.: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001), Morgan Kaufmann, San Francisco (2001) 432–439
43. Krasnogor, N., Blackburne, B., Burke, E., Hirst, J.: Multimeme algorithms for protein structure prediction. [12] 769–778
44. Hansen, P., Mladenovič, N.: An introduction to variable neighborhood search. In Voß, S., Martello, S., Osman, I., Roucairol, C., eds.: Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization. Proceedings of MIC 97 Conference. Kluwer Academic Publishers, Dordrecht, The Netherlands (1998)
45. Lourenco, H.R., Martin, O., Stutzle, T.: Iterated local search. In Glover, F., Kochenberger, G., eds.: Handbook of Metaheuristics. Kluwer Academic Publishers, Norwell, MA (2002) 321–353
46. Glover, F.: Tabu search: 1. ORSA Journal on Computing **1** (1989) 190–206
47. Kirkpatrick, S., Gelatt, C., Vecchi, M.: Optimization by simulated anealing. Science **220** (1983) 671–680
48. Wolpert, D., Macready, W.: No Free Lunch theorems for optimisation. IEEE Transactions on Evolutionary Computation **1** (1997) 67–82
49. Dawkins, R.: The Selfish Gene. Oxford University Press, Oxford, UK (1976)
50. Cavalli-Sforza, L., Feldman, M.: Cultural Transmission and Evolution: A Quatitative Approach. Princeton University Press, Princeton, NJ. (1981)
51. Durham, W.: Coevolution: Genes, Culture and Human Diversity. Stanford University Press (1991)
52. Gabora, L.: Meme and variations: A computational model of cultural evolution. In L.Nadel, Stein, D., eds.: 1993 Lectures in Complex Systems. Addison Wesley (1993) 471–494
53. Blackmore, S.: The Meme Machine. Oxford University Press, Oxford, UK (1999)
54. of Memetics. Advisory Board:S.Blackmore, J., G.Cziko, R.Dawkins, D.Dennett, L.Gabora, D.Hull., eds.: Journal of Memetics: Evolutionary Models of Information Transmission. (http://jom-emit.cfpm.org/)
55. Krasnogor, N.: Co-evolution of genes and memes in memetic algorithms. In Wu, A., ed.: Proceedings of the 1999 Genetic And Evolutionary Computation Conference Workshop Program. (1999)
56. Krasnogor, N.: Self-generating metaheuristics in bioinformatics: The protein structure comparison case. Genetic Programming and Evolvable Machines. Kluwer academic Publishers **5** (2004) 181–201
57. Krasnogor, N., Gustafson, S.: A study on the use of "self-generation" in memetic algorithms. Natural Computing **3** (2004) 53–76
58. Smith, J.: Co-evolving memetic algorithms: A learning approach to robust scalable optimisation. [1] 498–505
59. Baldwin, J.: A new factor in evolution. American Naturalist **30** (1896)
60. Krasnogor, N., Pelta, D.: Fuzzy memes in multimeme algorithms: a fuzzy-evolutionary hybrid. In Verdegay, J., ed.: Fuzzy Sets based Heuristics for Optimization, Springer (2002)
61. Eshelman, L.: The CHC adaptive search algorithm: how to have safe search when engaging in non-traditional genetic recombination. In Rawlins, G., ed.: Foundations of Genetic Algorithms, Morgan Kaufmann, San Francisco (1990) 263–283
62. Aarts, E., Korst, J.: Simulated Annealing and Boltzmann Machines. Wiley, Chichester, UK (1989)

63. Krasnogor, N., Smith, J.: A memetic algorithm with self-adaptive local search: TSP as a case study. In Whitley, D., Goldberg, D., Cantu-Paz, E., Spector, L., Parmee, I., Beyer, H.G., eds.: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000), Morgan Kaufmann, San Francisco (2000) 987–994
64. Kallel, L., Naudts, B., Reeves, C.: Properties of fitness functions and search landscapes. In Kallel, L., Naudts, B., Rogers, A., eds.: Theoretical Aspects of Evolutionary Computing. Springer, Berlin, Heidelberg, New York (2001) 175–206
65. Bull, L., Holland, O., Blackmore, S.: On meme–gene coevolution. Artificial Life **6** (2000) 227–235
66. Krasnogor, N., Gustafson, S.: Toward truly "memetic" memetic algorithms: discussion and proofs of concept. In Corne, D., Fogel, G., Hart, W., Knowles, J., Krasnogor, N., Roy, R., Smith, J., Tiwari, A., eds.: Advances in Nature-Inspired Computation: The PPSN VII Workshops, Reading, UK, PEDAL (Parallel, Emergent & Distributed Architectures Lab), University of Reading (2002) 9–10
67. Smith, J.: Co-evolution of memetic algorithms: Initial investigations. [12] 537–548
68. Smith, J.: The co-evolution of memetic algorithms for protein structure prediction. In Corne, D., Fogel, G., Hart, W., Knowles, J., Krasnogor, N., Roy, ., Smith, J., Tiwari, A., eds.: Advances in Nature-Inspired Computation: The PPSN VII Workshops, Reading, UK, PEDAL (Parallel, Emergent & Distributed Architectures Lab), University of Reading (2002) 14–15
69. Smith, J.: Protein structure prediction with co-evolving memetic algorithms. [1] 2346–2353
70. Rudolph, G.: Convergence of evolutionary algorithms in general search spaces. [82] 50–54
71. Hart, W., DeLaurentis, J., Ferguson, L.: On the convergence of an implicitly self-adaptive evolutionary algorithm on one-dimensional unimodal problems. IEEE Trans Evolutionary Computation (to appear) (2003)
72. Bramlette, M.: Initialization, mutation and selection methods in genetic algorithms for function optimization. In Belew, R., Booker, L., eds.: Proceedings of the 4th International Conference on Genetic Algorithms, Morgan Kaufmann, San Francisco (1991) 100–107
73. Surry, P., Radcliffe, N.: Innoculation to initialise evolutionary search. In T.C.Fogarty, ed.: Evolutionary Computing: Proceedings of the 1996 AISB Workshop, Springer, Berlin, Heidelberg, New York (1996) 269–285
74. Hart, E., Ross, P., Nelson, J.: Solving a real-world problem using an evolving heuristically driven schedule builder. Evolutionary Computation **6** (1998) 61–81
75. Thangiah, S., Vinayagamoorty, R., Gubbi, A.: Vehicle routing and time deadlines using genetic and local algorithms. [81] 506–515
76. Smith, J., Bartley, M., Fogarty, T.: Microprocessor design verification by two-phase evolution of variable length tests. In: Proceedings of the 1997 IEEE Conference on Evolutionary Computation, IEEE Press, Piscataway, NJ (1997) 453–458
77. Unger, R., Moult, J.: A genetic algorithm for 3D protein folding simulations. [81] 581–588
78. Friesleben, B., Merz, P.: A genetic local search algorithm for solving the symmetric and assymetric travelling salesman problem. [82] 616–621

79. Guervos, J.M., Adamidis, P., Beyer, H.G., Fernandez-Villacanas, J.L., Schwefel, H.P., eds.: Proceedings of the 7th Conference on Parallel Problem Solving from Nature. In Guervos, J.M., Adamidis, P., Beyer, H.G., Fernandez-Villacanas, J.L., Schwefel, H.P., eds.: Proceedings of the 7th Conference on Parallel Problem Solving from Nature. Number 2439 in Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, New York (2002)
80. 2003 Congress on Evolutionary Computation (CEC 2003). In: 2003 Congress on Evolutionary Computation (CEC 2003), IEEE Press, Piscataway, NJ (2003)
81. Forrest, S., ed.: Proceedings of the 5th International Conference on Genetic Algorithms. In Forrest, S., ed.: Proceedings of the 5th International Conference on Genetic Algorithms, Morgan Kaufmann, San Francisco (1993)
82. Proceedings of the 1996 IEEE Conference on Evolutionary Computation. In: Proceedings of the 1996 IEEE Conference on Evolutionary Computation, IEEE Press, Piscataway, NJ (1996)

Part II

Applications of Memetic Algorithms

An Evolutionary Approach for the Maximum Diversity Problem

Kengo Katayama[1] and Hiroyuki Narihisa[2]

[1] Okayama University of Science, 1 - 1 Ridai-cho, Okayama, 700-0005 Japan
katayama@ice.ous.ac.jp
[2] Okayama University of Science, 1 - 1 Ridai-cho, Okayama, 700-0005 Japan
narihisa@ice.ous.ac.jp

Summary. The objective of the maximum diversity problem (MDP) is to select a set of m-elements from larger set of n-elements such that the selected elements maximize a given diversity measure. The paper presents an evolutionary algorithm incorporating local search — memetic algorithm (MA) — for the MDP which consists of a greedy method, simple evolutionary operators, a repair method, and a k-flip local search based on *variable depth search*. In the MA, the k-flip local search starts with a feasible solution and obtains a local optimum in the feasible search space. Since infeasible solutions may be created by the simple crossover and mutation operators even if they start with feasible ones found by the local search, the repair method is applied to such infeasible solutions after the crossover and the mutation in order to guarantee feasibility of solutions to the problem. To show the effectiveness of the MA with the k-flip local search, we compare with a MA with 2-flip local search for large-scale problem instances (of up to $n = 2500$) which are larger than those investigated by other researchers. The results show that the k-flip local search based MA is effective particularly for larger instances. We report the best solution found by the MA as this is the first time such large instances are tackled.

1 Introduction

We consider the following maximum diversity problem (MDP). Given a symmetric $n \times n$ matrix d_{ij} ($d_{ij} = d_{ji}$ and $d_{ii} = 0$) and a predetermined number of size m ($n > m > 1$), the objective of the MDP is to select a subset of m-elements from n-elements such that the selected elements maximize a *diversity* measure. The MDP is represented as the following quadratic zero-one integer program:

$$\begin{aligned} \text{maximize } & f(x) = \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_i x_j, \\ \text{subject to } & \sum_{i=1}^{n} x_i = m, \\ & x_i \in \{0,1\},\ \forall i = 1,\ldots,n. \end{aligned} \tag{1}$$

The first model of the MDP has been formulated by Kuo, Glover, and Dhir [15] in which the concept of diversity is quantifiable and measurable. The concept of diversity is described as follows: consider a set of elements $S = \{s_i : i \in N\}$ with the index set $N = \{1, 2, \ldots, n\}$ and their common r attributes that each element possesses, denoted by s_{ik}, $k \in R = \{1, 2, \ldots, r\}$. To measure the diversity of a selected set of elements, a specified distance d_{ij} between each pair of elements s_i and s_j is required. One of the most commonly used distances may be the Euclidean distance, $d_{ij} = [\sum_{k=1}^{r}(s_{ik} - s_{jk})^2]^{1/2}$. In the MDP, it is assumed that the matrix d can be given by such a distance.

Kuo et al. proved the problem to be NP-hard, both with and without restricting the d_{ij} coefficients to non-negative values. Moreover, they transformed the maximum diversity model into two equivalent linear integer programming models and maximin diversity model in order to solve the problem by integer programming approaches. The MDP shown above is a general diversity maximization model that arises in data mining [14] and is substantially equivalent to the model of Kuo et al.

The MDP has a large number of applications. For example, such applications are immigration and admissions policies, committee formation, curriculum design, market planning and portfolio selection [5, 15]. Moreover, there are VLSI design and exam timetabling problems [23]. Others are environmental balance, medical treatment, genetic engineering, molecular structure design, agricultural breeding stocks, right sizing the firm, and composing jury panels [14].

The form of the MDP is quite similar to that of the unconstrained binary quadratic programming problem (BQP) in that they are both problems of maximizing a quadratic objective by suitable choice of binary (zero-one) variables. The BQP can be expressed as follows:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_i x_j,\ x_i \in \{0,1\},\ \forall i = 1,\ldots,n. \tag{2}$$

Thus, the MDP can be interpreted as a constrained version of the BQP.

Applications of the BQP are known to be abundant as well as the MDP. They appear in machine scheduling, traffic message management, CAD, capital budgeting and financial analysis, and molecular conformation [3]. Furthermore, it has been known that several classical combinatorial optimization problems can be formulated as a BQP, such as maximum cut problem, maximum clique problem, maximum vertex packing problem, minimum vertex cover problem, and maximum (weight) independent set problem [21, 22].

Since the problems MDP and BQP are NP-hard, exact methods would become prohibitively expensive to apply for large scale problem instances, whereas the heuristic or metaheuristic approaches may find high quality solutions of near-optimum with reasonable times. For the BQP, several heuristic and metaheuristic approaches have been developed; for example, greedy methods [7, 19], local searches [11, 19], and the metaheuristics, tabu search [3, 6], simulated annealing [3, 9], iterated local search [12], and evolutionary methods such as scatter search [1] and genetic algorithms incorporating local search [10, 17, 18, 20].

On the other hand, studies on such approaches for the MDP seem much more limited. Ghosh [8] showed a greedy randomized adaptive search procedure (GRASP) for the MDP. The GRASP metaheuristic was tested on small problem instances with $n \leq 40$. Glover et al. [5] proposed two constructive heuristics and two destructive heuristics for the MDP. They tested them for several instances of up to $n = 30$. Kochenberger et al. [14] dealt with large instances of the general MDP from $n = 100$ to $n = 1000$, which are randomly generated, and tested a tabu search metaheuristic taken into account searching *infeasible* space. Their tabu search to the MDP is based on the algorithm that has been developed for the BQP. The tabu search includes the strategic oscillation with constructive and destructive heuristics. The details of the tabu search can be found in [6].

This paper presents an evolutionary approach to the MDP. To the best of our knowledge, such an evolutionary approach is the first attempt to the MDP. Our approach consists of a greedy method to create initial solutions, simple evolutionary operators such as uniform crossover and bit-flip mutation, a repair method to turn an infeasible solution created by crossover or mutation into a feasible one, and a sophisticated k-flip local search based on variable depth search [13, 16], to be an effective memetic algorithm (MA) for the MDP. To show the effectiveness of the MA, computational experiments are conducted on large problem instances of up to $n = 2,500$ compared to the previous studies for the MDP. The results demonstrate that the k-flip local search based MA is more effective than a MA based on 2-flip local search in terms of final solutions, particularly for large-scale problem instances.

The paper is organized as follows. In the next section, we show the k-flip local search incorporated in the memetic algorithm for the MDP. In section 3, a flow of the memetic algorithm is given, and each operation in the algorithm is described. In section 4, we report experimental results for the memetic algorithms tested on our new problem instances and on several benchmarks derived from well-known BQP's ones in which the d_{ij} coefficients are not restricted to non-negative values. The final section contains concluding remarks.

2 Local Search for MDP

Local Search (LS) is a generally applicable approach that can be used to find approximate solutions to hard optimization problems. Many powerful heuristics are so-called *metaheuristics* such as memetic algorithm are based on LS.

The basic idea of LS is to start with a feasible solution x (e.g., randomly generated solution) and to repeatedly replace x with a better solution x' selected from the set of neighboring solutions that can be reached by a slight modification of the current solution. If no better solutions can be found in the set of neighbors, LS immediately stops and finally returns the best solution found during the search. Thus, a resulting solution cannot be improved by the slight modification. This modification is achieved by a predefined structure often referred to as *neighborhood* $\mathcal{NB}$. The resulting solution is called *locally optimal* with respect to the neighborhood. LS is an integral process in the memetic framework. The remainder of this section describes a LS, k-flip local search, for the MDP. We begin by describing the fitness (objective) function and the solution representation on which the local search and the evolutionary operators in the memetic algorithm are based.

2.1 Fitness Function and Solution Representation

In our memetic algorithm incorporating the local search for the MDP, the fitness, i.e., a solution cost, is evaluated by equation (1).

A solution to the MDP can be represented in a binary string x of length $n = |N|$, where N denotes an index set of elements $N = \{1, 2, \ldots, n\}$. In this representation, a value of 0 or 1 at the i-th bit (element) implies that $x_i = 0$ or 1 in the solution, respectively.

Let S_1 be an index set of elements with $x_i = 1$ for all $i \in N$ and S_0 be an index set of elements with $x_i = 0$ for all $i \in N$. In the MDP, we thus note that $S_1 \cup S_0 = N$ and $S_1 \cap S_0 = \emptyset$. To be a feasible solution, it is restricted that a sum of $x_i = 1$ for all $i \in N$ is equal to m $(= |S_1| = |N| - |S_0|)$ due to the constraint in the formulation (1). Note that in this paper the solution representation x always corresponds to a representation S_1 and S_0.

2.2 Neighborhoods

Although we use a k-flip local search heuristic in the LS process of the memetic framework for the MDP, 2-flip based neighborhood is mainly used in the k-flip local search as a basic move structure. In the crossover and mutation operators in the memetic algorithm, 1-flip based neighborhood is used. Thus, we here describe the two neighborhoods for the MDP.

Given a solution x, the 1-flip neighborhood $\mathcal{NB}_1$ is defined by the set of solutions that can be obtained by flipping a single bit x_i in the current solution. Thus, a hamming distance $d_H(x, x')$ between the current solution

x and the neighboring solution x' is 1. The number of all possible solutions that can be created from a current solution by the 1-flip neighborhood $\mathcal{NB}_1$ at a time is equal to n. Even if a given solution is feasible, a feasibility of the neighboring solutions that can be reached by the neighborhood is not preserved since the number of '0' and that of '1' in the neighboring solutions are changed from the feasible condition of $\sum_{i=1}^{n} x_i = m$. In order to guarantee feasibility of solutions, several considerations should be taken into account.

The 2-flip neighborhood $\mathcal{NB}_2$ is defined by the set of all solutions that can be reached by simultaneously flipping two bits x_i ($i \in S_1$) and x_j ($j \in S_0$) in the current solution x. The hamming distance between the current solution x and its neighboring solution x' can be $d_H(x, x') = 2$. Note in this neighborhood that it is not allowed to flip two bits i and j in the same set (e.g., $i, j \in S_0$). Given a feasible solution, therefore, the feasibility of neighboring solutions by the neighborhood can be always preserved. The number of possible neighboring solutions at a time is equal to $|S_1||S_0|$.

2.3 Gain Calculation for Neighbors

In order to perform an efficient search for a problem, it is crucial to calculate the difference $\Delta = f(x') - f(x)$, where f is an objective function of the problem, and x' denotes a neighboring solution obtained from a current one x by reference of a neighborhood, instead of naively calculating the cost of x' by $f(x')$ from scratch. In this paper, we refer the difference Δ to the term *gain* for a given neighboring solution x'. For the MDP, the gain can be computed much faster than the naive calculation $f(x')$.

Fast Gain Calculation for 1-flip Neighborhood

To achieve a fast calculation for the gain in the local search or memetic algorithm to the MDP, we refer to the paper of Merz and Freisleben [19] (see also [6, 20] as other related references). They showed that a calculation of all gains for the 1-flip (or 1-opt) neighbors to the BQP can be computed in linear time. The gain calculation for the BQP can be used for the MDP without modification.

Naively, the gain value g_j of flipping a single j-th bit in a current solution x can be computed by the difference between the objective function values of $f(x')$ and $f(x)$, i.e., $g_j = f(x') - f(x)$, where $x' = 1 - x_j$. However, the gain g_j can be calculated by the following formula:

$$g_j = d_{jj}(\overline{x}_j - x_j) + 2 \sum_{i=1, i \neq j}^{n} d_{ij}\, x_i\, (\overline{x}_j - x_j), \tag{3}$$

with $\overline{x}_j = 1 - x_j$. In this case, the gain g_j of flipping j-th bit in the current solution can be calculated in $O(n)$. However, the calculation of the all gains for the n candidates takes $O(n^2)$ time by using this formula.

Using the information of the all gains that have been already computed, all of new gains can be calculated efficiently, instead of recalculating them by (3). To achieve such a calculation, we take into account the update of the gains that is based on calculating the difference of gains Δg_i ($\forall i \in N$). Assuming that all g_i have been calculated and the bit j is flipped, the new gains g'_i can be computed efficiently by

$$g'_i = \begin{cases} -g_i & \text{if } i = j \\ g_i + \Delta g_i(j) & \text{otherwise} \end{cases} \quad \text{with} \quad \Delta g_i(j) = 2\, d_{ij}\, (\overline{x}_i - x_i)\, (x_j - \overline{x}_j). \tag{4}$$

The update of the gains for the n candidates of the 1-flip neighboring solutions can be performed in linear time. Furthermore, only the gains g_i for $d_{ij} \neq 0$ have to be updated [19].

This update technique [19] for the 1-flip neighbors is basically embedded with our local search and memetic algorithms for the MDP. If we consider a k-flip ($1 < k < n$) based neighborhood as used in the k-flip local search for the MDP, the following can be useful.

Generalized Gain Calculation

We now show a generalized gain calculation for a k-flip neighbor ($1 < k < n$) in the current solution in order to efficiently perform k-flip neighborhood search. The information of the matrix of a given problem instance and the gains for the 1-flip neighbors in a solution is fully used in the generalized gain calculation. Assuming that all gains g for the 1-flip neighbors are calculated and several k bits are flipped for a current solution x, a gain G of flipping the k bits (we assume in the following that the bits are stored in flip[ ] for convenience and all the bits are different) can be computed by

$$\begin{aligned} G = {} & g_\alpha && (\textit{1-flip}) \\ & + g_\beta + 2d_{\alpha\beta}(1-2x_\alpha)(1-2x_\beta) && (\textit{2-flip}) \\ & + g_\gamma + 2d_{\beta\gamma}(1-2x_\beta)(1-2x_\gamma) + 2d_{\gamma\alpha}(1-2x_\gamma)(1-2x_\alpha) && (\textit{3-flip}) \\ & + g_\delta + 2d_{\gamma\delta}(1-2x_\gamma)(1-2x_\delta) + 2d_{\delta\alpha}(1-2x_\delta)(1-2x_\alpha) \\ & + 2d_{\delta\beta}(1-2x_\delta)(1-2x_\beta) && (\textit{4-flip}) \\ & \qquad\qquad \vdots && \vdots \\ & = \sum_{i=1}^{k} g_{\mathrm{flip}[i]} + 2\sum_{i=1}^{k-1}\sum_{j=i+1}^{k} d_{\mathrm{flip}[i]\mathrm{flip}[j]}(1-2x_{\mathrm{flip}[i]})(1-2x_{\mathrm{flip}[j]}) && (\textit{k-flip}) \\ & \qquad (1 < k \leq n, \mathrm{flip}[\,] = \{\alpha, \beta, \gamma, \delta, \ldots\}). \end{aligned} \tag{5}$$

For example, assuming that two bits of α-th and β-th are flipped (i.e., 2-flip neighborhood) in the current solution, the gain G for the two bits can be given by a sum of g_α, g_β, and $2d_{\alpha\beta}(1-2x_\alpha)(1-2x_\beta)$ if the gains g for the 1-flip neighbors in the current solution is provided in advance. As shown

above, this calculation for the 2-flip neighbor can be extended to several k bits for the generalized k-flip neighborhood. The update of the gains have to be performed after each flip of k bits. The update can be done by (4).

This generalized gain calculation is valid for the MDP and the BQP.

2.4 k-flip Local Search

The larger sized neighborhoods such as k-flip neighborhood ($1 \ll k < n$) for the MDP may yield better local optima but the effort needed to search the neighborhood is too computationally expensive. An idea of the variable depth search (VDS) [13, 16] is based on efficiently searching a small fraction of the large neighborhood.

A basic idea of VDS based local search for the MDP is described as follows. Given a feasible solution x as an initial solution, in each iteration a sequence of m (or $n-m$, see below) solutions is produced by 2-flip based sub-moves leading from one solution to another, and the best solution x_{best} in the sequence is adopted as a new initial solution x for the next iteration. Such a process is repeated until no better solution is found.

To produce the sequence, the 2-flip based moves are sequentially performed so that each bit of x is flipped no more than once. All m solutions in the sequence are different and each solution x' differs two to k bits from the initial solution x. Thus, the hamming distance d_H between the initial solution x and each solution x' is $d_H(x, x') = k$, where $k = \{2, 4, \ldots, 2m-2, 2m\}$. Since the solution x_{best} with the highest cost is selected from the resulting sequence, the hamming distance $d_H(x, x_{best})$ is variable in each iteration of the algorithm, i.e., $d_H(x, x_{best}) = k$.

This specialized neighborhood may be called *k-flip neighborhood*. The k-flip neighborhood, denoted by $\mathcal{NB}_k$, for the MDP can be defined as follows:

$\mathcal{NB}_k(x) := \{x' \mid x'$ is obtained from a sequence of m solutions that can be obtained from x by exchanging an index i in one set S_1 with an index j in the other set S_0 under the following prohibition: all of the exchanged i and j are not re-exchanged$\}$.

Note in this neighborhood that the number m of the solutions in the sequence described above depends on the problem constraint. Throughout the paper, we assume that a given number m in the problem constraint is greater than one and fewer than a half of n variables [3]. If the given number m is greater than $n/2$, the elements of each S_0 and S_1 are all swapped before the search and the number of solutions produced in each iteration should be $n - m$.

[3] If the given number m is just $n/2$ in the problem constraint, we should produce $m-1$ solutions for the sequence in each iteration of the algorithm, because both a given current solution and a resulting m-th solution in the sequence become the same, that is, all elements in each set S_1 and S_0 in an initial state are only exchanged each other, if the 2-flip move is embedded with the k-flip neighborhood search.

```
    procedure k-Flip-Local-Search-QuasiBstImp2-FlipMove(x, g)
    begin
 1    repeat
 2        x_prev := x, G_max := 0, G := 0, C1 := S_1, C0 := S_0;
 3        repeat
 4            find j with g_j = max_{j∈C1} g_j;
 5            find k with gain = max_{k∈C0} (g_k + g_j + 2d_jk(1 − 2x_k)(1 − 2x_j));
 6            G := G + gain;
 7            x_j := 1 − x_j, x_k := 1 − x_k, and update gains g for each flipping;
 8            C1 := C1\{j}, C0 := C0\{k};
 9            if G > G_max then G_max := G, x_best := x;
10        until new x_best is not found for several repeats or C1 = ∅;
11        if G_max > 0 then x := x_best else x := x_prev;
12    until G_max ≤ 0;
13    return x;
    end;
```

Fig. 1. *k-flip* Local Search with Quasi-Best Improvement 2-flip Move

Quasi-Best Improvement k-flip Local Search

Our k-flip local search used in our memetic algorithm is based on the above basic idea. To produce a sequence of different m solutions in each iteration, we perform the 2-flip based sub-move with quasi-best improvement. We thus call it the *quasi-best improvement k-flip local search.* The meaning of the *quasi-best* is mentioned later.

Figure 1 shows the pseudo-code of the quasi-best improvement k-flip local search heuristic for the MDP. In the figure, we assume that 1) a feasible solution x and an associated gain vector g are provided in advance. 2) the gain vector is maintained and updated using (4) after each flip, and the generalized gain calculation (5) is used for solutions by 2-flip moves. 3) the solution x always corresponds to the representation of the sets S_1 and S_0 as mentioned in the section 2.1.

The local search consists of two loops: an inner loop in which a sequence of solutions is produced and the best solution is selected from the sequence and an outer loop in which the best solution found in the inner loop is evaluated.

To produce a sequence of different solutions in the inner loop, two candidate sets of $C1$ and $C0$ are used to ensure that each bit of a given initial solution x is flipped no more than once. Therefore, a basic stopping criterion of the search in the inner loop is expressed as $C1 = \emptyset$. To choose the best solution in the sequence, the inner loop involves a judgment process (line 9) whether a current solution x' is better than the incumbently stored best-solution x_{best}. Such a judgment plays a key role in a reduction of running time for the local search. In the k-flip local search, we change the stopping criterion of the search in the inner loop as follows: the inner loop (line 10) is terminated if new x_{best} is not found for more than t repeats or if $C1 = \emptyset$. A parameter t for the MDP is set to a range $1 < t < m$ in advance and is fixed during

```
   procedure MA
   begin
 1    initialize a population P ∈ {I_1, ..., I_PS};
 2    foreach individual I ∈ P do I := Local-Search(I);
 3    repeat
 4        for i := 1 to #crossovers do
 5            choose two parents I_a, I_b ∈ P randomly;
 6            I_c := Crossover(I_a, I_b);
 7            I_c := Repair(I_c);
 8            I_c := Local-Search(I_c);
 9            add an individual I_c to P_c;
10        endfor
11        P := Selection(P, P_c);
12        if diversification=true then
13            foreach individual I ∈ P\{best individual } do
14                I := Mutation(I);
15                I := Repair(I);
16                I := Local-Search(I);
17        endif
18    until terminate=true;
19    return best individual ∈ P;
   end;
```

Fig. 2. An outline of our evolutionary approach to the MDP

the local search. When choosing a suitable value of the parameter, it is quite expected that the running time of the local search is considerably reduced in comparison with only the basic criterion, but a sacrifice may be made in the guarantee of choosing the *true* best solution in the sequence which might be produced with the criterion $C1 = \emptyset$. Such a parameter setting is derived from the k-opt local search for the BQP [19]. In our k-flip local search for the MDP, we adopt a parameter value $t = m/5$ for larger MDP instances ($n \geq 500$), for smaller instances ($n < 500$) t is set to m. These setting show good behavior in our initial experiments.

In the outer loop, the solution x_{best} selected is evaluated whether x_{best} is better than the initial solution given at beginning of the inner loop. This can be done by a check $G_{max} > 0$. If satisfied, the k-flip neighborhood search is performed after x_{best} is set to x, otherwise, the local search is terminated after the return of the best solution found during the search.

The *quasi-best* improvement k-flip local search is a faster variant of the best improvement k-flip local search. In the best improvement version, each solution in the sequence is produced by selecting the best pair with the highest gain in the sets $C1$ and $C0$. This local search takes $O(m|S_1||S_0|)$ time per iteration. However, in the quasi-best improvement version, a first bit with the highest gain is selected from $C1$ and then a second bit with the highest gain in the 2-flip move is determined from $C0$. Thus, the time complexity of each iteration in the quasi-best improvement version is $O(m|S_1| + m|S_0|)$ time.

3 Memetic Algorithm for MDP

Memetic framework for the MDP shown in this paper is similar to one for other difficult optimization problems, which consists of a local search procedure and evolutionary operators. However, each operation in the framework has to be devised so as to work well for the MDP.

An outline of our memetic algorithm is shown in Figure 2. After the initialization of the population, new offspring are created by application of crossover and local search a predefined number of times. A new population is produced by selecting individuals from the old population and the set of generated offspring (P_c). Unless the search has converged, this process is repeated until a predefined time limit is exceeded.

In the following, the evolutionary operators are described in detail.

3.1 Creating the Initial Population

In our approach, the initial solutions (individuals) $(I_1, \ldots, I_{PS})$ of a population P are created by a randomized greedy method, where PS is a predetermined number of the individuals. The method is a variant of the randomized greedy heuristic for the BQP described in [19]. The greedy method for the MDP is devised so that m bits with '1' are appeared in a solution of length n in order to create a feasible solution. Afterwards, each of these feasible individuals is locally optimized by a local search, i.e., the quasi-best improvement k-flip local search, to create an initial population of locally optimum solutions.

3.2 Crossover

In the MDP, classical crossover operators, such as one-point, two-point, or uniform crossover can be applied, but it is not preserved in many cases that a new solution created by such a crossover is feasible even if starting with two feasible parents.

In our approach, we use the uniform crossover in which a single offspring is created from two parents, as shown in Figure 3. In the crossover process, two parents are chosen randomly from a current population such that all individuals are used with a restriction that no individual in the population is used twice in each generation. Therefore, the number of crossover processes depends on the size of population, i.e., $PS/2$.

After each crossover process, a repair method is applied to turn an infeasible offspring into a feasible one, and each feasible solution is locally improved by the k-flip local search. The repair method is described in 3.4.

3.3 Selection and Diversification/Restart Strategy

In each generation, a new population has to be formed after offspring have been generated. In our approach, the PS individuals with the highest fitness

parent1	1	0	0	1	1	0	1	0
parent2	0	0	1	1	0	1	1	0
	*	0	*	1	*	*	1	0
offspring	0	0	1	1	1	0	1	0

Comment : For the offspring, a value of 0 or 1 at each position '*' is chosen with probability 0.5.

Fig. 3. An example of uniform crossover

```
   procedure Repair(x, g)
   begin
 1    calculate a violation v := m − |S1|;
 2    if v = 0 then return x;
 3    else if v < 0 then
 4        repeat
 5            find j with g_j = max_{j∈S1} g_j;
 6            x_j := 1 − x_j, S1 := S1\{j}, and update gains g;
 7        until Σ_{i=1}^{n} x_i = m;
 8        return x;
 9    else
10        repeat
11            find j with g_j = max_{j∈S0} g_j;
12            x_j := 1 − x_j, S0 := S0\{j}, and update gains g;
13        until Σ_{i=1}^{n} x_i = m;
14        return x;
15    endif
   end;
```

Fig. 4. Repair Method

of the old population P and the set of offspring P_c are selected. However, the duplicates from the temporary set containing P and P_c are removed to ensure that no MDP solution is contained in the new population more than once.

A general drawback in evolutionary approach may be a premature convergence of the algorithm, especially in the absence of mutation. We thus perform a diversification/restart strategy, which is borrowed from [4], in order to move to other points of the search space if no new best individual in the population was found for more than 30 generations. In response to this requirement, the individuals except for the best one in the population are mutated by flipping randomly chosen $n/2$ bits for each individual of length n. After that, each of the mutated individuals is applied to the repair method. Then each individual after the repair method is locally improved by the k-flip local search to obtain a renewal set of local optima and the search is started again with the new, diverse population.

3.4 Repair Method

A repair method should be applied to an infeasible solution after the crossover and the mutation in the diversification strategy, since the feasibility of the solution by such an operator is not preserved for the MDP.

Analogous to the k-flip local search procedure shown above, the gains g corresponding to a given solution x is managed and updated in the repair procedure. When a solution x given for the repair method is infeasible, it is repeated that a violated bit with the highest gain is flipped in each repair iteration even if the highest one is negative. Such a repair process is executed until x becomes feasible. By using this repair algorithm, the given infeasible solution is turned to be feasible one that is as better cost as possible.

Our repair algorithm for the MDP is given in pseudo-code in Figure 4. At first, the violation number of the given solution x is calculated. If no violation, the solution is immediately returned as a feasible one at line 2, otherwise, the repair process is performed to obtain a feasible solution from the given infeasible one by flipping v bits in the set of S_1 or S_0 according to a judgment of line 3. The number of the repair iterations therefore depends on the number v. The each iteration consists of selecting a bit with the highest gain and flipping the bit with the update of the gains. The time complexities of each repair iteration in the algorithm are $O(|S_1|)$ or $O(|S_0|)$ time for the line 5 or 11 and $O(n)$ time for updating gains g after each flip.

4 Computational Experiments

4.1 Test Instances

For the experiments, we newly provide six problem sets for the MDP. Each problem set is characterized by the following problem sizes: n = 100, 250, 500, 750, 1,000, and 2,500 variables. We name them `mdp00100`, `mdp00250`, `mdp00500`, `mdp00750`, `mdp01000`, and `mdp02500`, respectively. The each set, i.e., the matrix d, is generated in the following way: each of d_{ij} $(i < j)$ values is given randomly between 1 and 50. Therefore, each matrix is 100% dense problem but the diagonal is off, $d_{ii} = 0$. Each problem set consists of four instances, and each of the four instances in the set is characterized by a different value of m. The four values of m are set to 10%, 20%, 30%, 40% of the variable size n, respectively.

In addition, we also use three test problem sets, which we modified benchmark instances of the BQP contained in ORLIB [2]. (This is the first attempt in research for the MDP.) Their variable sizes are n = 500, 1,000, and 2,500, and their names are `beas500-1`, `beas1000-1`, and `beas2500-1`, respectively, which are first used as test instances for BQP's heuristic algorithms in [3]. Note in each problem set that the d_{ij} coefficients in the original matrix are not restricted to non-negative values. In the three sets a density of each matrix

is 10%. However, we modified the matrix as follows: the diagonal is off, i.e., $d_{ii} = 0$, due to the definition in the MDP. Each set consists of four instances that are characterized by the four values of m as well as the new test problem instances we provided above.

The variable sizes of the first five sets in our new test problems and the first two sets in ORLIB are competitive with [14] that reported results for randomly generated instances of up to 1,000 variables, but the remainders are much larger than any reported in the literature for the MDP.

The test instances we newly provided are available from the following web page: `http://k2x.ice.ous.ac.jp/~katayama/bench/`.

4.2 Results and Discussions

We imposed a time limit for the memetic algorithm. The time limit was chosen for each variable size of the problem sets: 10 seconds for 100 variable instances, 30 (sec) for 250 variable instances, 100 (sec) for 500 variable instances, 300 (sec) for 750 variable instances, 1000 (sec) for 1,000 variable instances, and 3000 (sec) for 2,500 variable instances, on a Sun Ultra 5/10 (UltraSPARC-IIi 440MHz). The algorithm was run 30 times for each instance. Each run of the algorithm is performed with a different seed. The value of the best solution found by the algorithm in each run was saved with their corresponding generation number, running time, etc. The algorithm was implemented in *C*. The program code was compiled with the gcc compiler using the optimization flag -O2 on Solaris 8.

The parameters contained in the memetic algorithm have already described in the previous sections except for the population size. The population size PS was set to 40, which is a commonly used population size for evolutionary algorithms incorporating local search.

Table 1 shows the results for the memetic algorithm with the k-flip local search obtained for the test problem instances. In the first three columns of the table, the name of the problem sets, the variable size n, and the number of the problem constraint m are given. In the following columns, we provide the best solution value (quality % of the best solution), the average solution value (quality % of the average solution) of 30 runs, the number of times in which the best solution could be found by the algorithm "b/run", the average running time "t1" in seconds in case the algorithm could find the best solution, and the time-limit "t2" in seconds (exclusive of the case the algorithm could find the best solution). In addition, "t1" and "t2" are provided with their corresponding average generation numbers "(gens)". In the table, the number of 30/30 shown in the column of "b/run" indicates that the best-known solution could be found by the algorithm within the predefined time limit in *all* 30 trials. As an additional result, the final line in this table shows the result of the MA with longer time limit of 30000 seconds for `mdp02500` with $m = 750$. In the result, the MA found a better solution of $f(x) = 14988436$

Table 1. Results for the quasi-best improvement k-flip local search based MA

instance (off diag.)			Memetic Algorithm with k-flip Local Search						
name	n	m	best (%)	avg. (%)	b/run	t1	(gens)	t2	(gens)
mdp00100	100	10	3606 (0.000000)	3606.0 (0.000000)	30/30	0.1	(4)	—	(—)
	100	20	12956 (0.000000)	12956.0 (0.000000)	30/30	0.1	(1)	—	(—)
	100	30	27036 (0.000000)	27036.0 (0.000000)	30/30	0.1	(1)	—	(—)
	100	40	4587discussion2 (0.000000)	45872.0 (0.000000)	30/30	0.2	(2)	—	(—)
mdp00250	250	25	20834 (0.000000)	20834.0 (0.000000)	30/30	1.0	(6)	—	(—)
	250	50	75816 (0.000000)	75816.0 (0.000000)	30/30	1.8	(5)	—	(—)
	250	75	162252 (0.000000)	162252.0 (0.000000)	30/30	14.4	(40)	—	(—)
	250	100	279470 (0.000000)	279470.0 (0.000000)	30/30	2.5	(4)	—	(—)
mdp00500	500	50	78898 (0.000000)	78898.0 (0.000000)	30/30	5.1	(15)	—	(—)
	500	100	291916 (0.000000)	291916.0 (0.000000)	30/30	4.5	(5)	—	(—)
	500	150	631898 (0.000000)	631898.0 (0.000000)	30/30	12.9	(24)	—	(—)
	500	200	1096092 (0.000000)	1096092.0 (0.000000)	30/30	1.7	(2)	—	(—)
mdp00750	750	75	171704 (0.000000)	171704.0 (0.000000)	30/30	74.5	(99)	—	(—)
	750	150	641140 (0.000000)	641140.0 (0.000000)	30/30	12.6	(6)	—	(—)
	750	225	1395672 (0.000000)	1395672.0 (0.000000)	30/30	106.3	(71)	—	(—)
	750	300	2430660 (0.000000)	2430660.0 (0.000000)	30/30	34.4	(16)	—	(—)
mdp01000	1000	100	299730 (0.000000)	299721.7 (0.002758)	28/30	456.7	(249)	1000	(524)
	1000	200	1125264 (0.000000)	1125264.0 (0.000000)	30/30	165.4	(57)	—	(—)
	1000	300	2458316 (0.000000)	2458316.0 (0.000000)	30/30	375.0	(102)	—	(—)
	1000	400	4292438 (0.000000)	4292438.0 (0.000000)	30/30	81.3	(16)	—	(—)
mdp02500	2500	250	1775366 (0.000000)	1774418.9 (0.053349)	3/30	2313.4	(163)	3000	(203)
	2500	500	6794586 (0.000000)	6793782.4 (0.011827)	2/30	930.3	(30)	3000	(123)
	2500	750	14988200 (0.001575)	14987346.8 (0.007267)	0/30	—	(—)	3000	(88)
	2500	1000	26332276 (0.000000)	26332274.3 (0.000007)	29/30	927.1	(19)	3000	(82)
beas500-1	500	50	21750 (0.000000)	21750.0 (0.000000)	30/30	1.7	(25)	—	(—)
	500	100	48738 (0.000000)	48738.0 (0.000000)	30/30	1.0	(6)	—	(—)
	500	150	73544 (0.000000)	73544.0 (0.000000)	30/30	9.6	(62)	—	(—)
	500	200	95516 (0.000000)	95516.0 (0.000000)	30/30	0.5	(1)	—	(—)
beas1000-1	1000	100	62102 (0.000000)	62102.0 (0.000000)	30/30	22.1	(64)	—	(—)
	1000	200	141512 (0.000000)	141512.0 (0.000000)	30/30	41.6	(79)	—	(—)
	1000	300	218478 (0.000000)	218478.0 (0.000000)	30/30	177.8	(245)	—	(—)
	1000	400	284688 (0.000000)	284688.0 (0.000000)	30/30	57.0	(65)	—	(—)
beas2500-1	2500	250	246678 (0.000000)	246678.0 (0.000000)	30/30	581.9	(248)	—	(—)
	2500	500	566928 (0.000000)	566473.1 (0.080245)	9/30	1323.0	(318)	3000	(709)
	2500	750	882276 (0.000000)	882240.0 (0.004080)	28/30	720.2	(118)	3000	(506)
	2500	1000	1153068 (0.000000)	1151871.1 (0.103798)	1/30	1245.2	(189)	3000	(397)
mdp02500	2500	750	14988436 (0.000000)	14988307.6 (0.000857)	20/30	15000.3	(432)	30000	(807)

than the case of 3000 seconds. Thus, it indicates that the MA is capable of finding better solutions if longer running times are allowed.

Since the optimal solution for each instance is unknown yet, we reported the value of the best solution found by the algorithm for each instance as a result. It is expected that each of these best-known solutions is likely to be the very near-optimal or the optimal solution for each of the instances.

To show the effectiveness of the k-flip local search based memetic algorithm (MA-k-flip) for the MDP, we test a 2-flip local search based variant algorithm (MA-2-flip). The difference between them is only the local search process. In the variant, the same parameters and time limits set in the memetic algorithm with k-flip local search are adopted to be fair. This 2-flip local search performs the quasi-best improvement strategy as moves in each iteration.

Table 2. Results for the quasi-best improvement 2-flip local search based MA

instance (off diag.)			Memetic Algorithm with 2-flip Local Search							
name	n	m	best	(%)	avg.	(%)	b/run	t1	(gens)	t2	(gens)
mdp00100	100	10	3606	(0.000000)	3606.0	(0.000000)	30/30	0.1	(14)	—	(—)
	100	20	12956	(0.000000)	12956.0	(0.000000)	30/30	0.1	(4)	—	(—)
	100	30	27036	(0.000000)	27036.0	(0.000000)	30/30	0.1	(3)	—	(—)
	100	40	45872	(0.000000)	45872.0	(0.000000)	30/30	0.1	(7)	—	(—)
mdp00500	500	50	78898	(0.000000)	78898.0	(0.000000)	30/30	6.8	(78)	—	(—)
	500	100	291916	(0.000000)	291916.0	(0.000000)	30/30	7.1	(84)	—	(—)
	500	150	631898	(0.000000)	631898.0	(0.000000)	30/30	8.5	(100)	—	(—)
	500	200	1096092	(0.000000)	1096092.0	(0.000000)	30/30	7.6	(77)	—	(—)
mdp01000	1000	100	299730	(0.000000)	299642.5	(0.029204)	8/30	459.4	(717)	1000	(1610)
	1000	200	1125264	(0.000000)	1125180.9	(0.007382)	18/30	516.3	(832)	1000	(1518)
	1000	300	2458316	(0.000000)	2458290.6	(0.001033)	11/30	444.2	(756)	1000	(1557)
	1000	400	4292438	(0.000000)	4292438.0	(0.000000)	30/30	49.4	(100)	—	(—)
beas500-1	500	50	21750	(0.000000)	21750.0	(0.000000)	30/30	2.1	(112)	—	(—)
	500	100	48738	(0.000000)	48738.0	(0.000000)	30/30	0.9	(55)	—	(—)
	500	150	73544	(0.000000)	73544.0	(0.000000)	30/30	22.8	(955)	—	(—)
	500	200	95516	(0.000000)	95516.0	(0.000000)	30/30	0.7	(23)	—	(—)
beas1000-1	1000	100	62102	(0.000000)	62097.9	(0.006548)	29/30	188.1	(2013)	1000	(9906)
	1000	200	141512	(0.000000)	141512.0	(0.000000)	30/30	161.9	(1601)	—	(—)
	1000	300	218478	(0.000000)	218406.5	(0.032742)	1/30	46.0	(529)	1000	(9344)
	1000	400	284688	(0.000000)	284653.9	(0.011990)	14/30	179.8	(1796)	1000	(9546)

Table 2 shows the results obtained by MA-2-flip, the memetic algorithm with the 2-flip local search, only for the 100, 500 and 1000 variables instances among the nine problem sets. In the table, we give the same column entries as in Table 1.

From the results of Tables 1 and 2, the performance of MA-k-flip may be comparable with that of MA-2-flip for the instances of $n \leq 500$ since the best solution found by MA-k-flip can be obtained by MA-2-flip with a high frequency (see the column of "b/run"). However, MA-k-flip has a better advantage for the larger instances of $n > 500$: the numbers of "b/run" in MA-k-flip are greater and the running times for reaching the best-known solutions are less than those of MA-2-flip in many cases, although it seems that MA-k-flip spends more running times per generation in the predefined time limit for the computation. Thus, the effectiveness of the k-flip local search based MA is superior to MA-2-flip.

Since the difference between MA-k-flip and MA-2-flip is only the process of local search, the results of Tables 1 and 2 make it clear that a design in the local search process is quite important to obtain good solutions for the problem. Moreover, better results are expected if the optimal value of the parameters is determined and if we devise evolutionary operators instead of simple ones used in our algorithms.

Unfortunately, a comparison with the previously proposed approaches to the MDP is difficult because in most cases their algorithms were tested on smaller instances generated by them and without using of publicly available instances such as BQP contained in ORLIB as tried in this paper.

Finally, we give our experience on the setting value of mutation in the diversification/restart strategy. Although our strategy with the default setting of $n/2$ in the mutation is considerably disruptive, we believe that this setting value is a better choice than that of a smaller value in this memetic framework for the MDP. In our additional experiments, we have attempted to flip smaller bits of $n/3$ chosen randomly in the mutation for each individual except for the best one of the current population, instead of the default setting. The results showed that the default setting gave better solutions, particularly with MA-k-flip and MA-2-flip for large instances.

5 Conclusion

In this paper, we have presented a memetic algorithm for solving the maximum diversity problem. Although most of the components of our algorithm were comparable in a standard memetic framework, newly developed methods, i.e., the powerful k-flip local search, the repair method, etc. were incorporated to obtain good solutions and to preserve the feasibility of solutions for the MDP. The results showed that the k-flip local search based memetic algorithm outperformed the 2-flip local search based variant particularly for larger instances we newly provided and contained as the BQP instances in ORLIB. Due to the first report for such instances, the values of the best solution found by the algorithm were also reported for the problem instances investigated.

One of the most important issues for future research is to compare the MA with other (meta-)heuristics for the same problem instances in order to assert the effectiveness of memetic approach to the MDP.

References

1. Amini, M.M., Alidaee, B., Kochenberger, G.A. (1999) A Scatter Search Approach to Unconstrained Quadratic Binary Programs. In Corne, D., Dorigo, M., Glover, F., eds.: New Ideas in Optimization. McGraw-Hill, London 317–329
2. Beasley, J.E. (1990) OR-Library: Distributing Test Problems by Electronic Mail. Journal of the Operational Research Society **41** : 1069–1072
3. Beasley, J.E. (1998) Heuristic Algorithms for the Unconstrained Binary Quadratic Programming Problem. Tech. Rep., Management School, Imperial College, UK
4. Eshelman, L. (1991) The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination. In Rawlings, G.J.E., ed. Foundations of Genetic Algorithms 265–283
5. Glover, F., Kuo, C.-C., Dhir, K.S. (1998) Heuristic Algorithms for the Maximum Diversity Problem. Journal of Information and Optimization Sciences **19** : 109–132

6. Glover, F., Kochenberger, G., Alidaee, B., Amini, M. (1999) Tabu Search with Critical Event Memory: An Enhanced Application for Binary Quadratic Programs. In Voss, S., et al., eds.: Meta-Heuristics, Advances and Trends in Local Search Paradigms for Optimization, Kluwer Academic Pub. 93–109
7. Glover, F., Alidaee, B., Rego, C., Kochenberger, G. (2002) One-Pass Heuristics for Large-Scale Unconstrained Binary Quadratic Problems. European Journal of Operational Research **137** : 272–287
8. Ghosh, J.B. (1996) Computational Aspects of the Maximum Diversity Problem. Operations Research Letters **19** : 175–181
9. Katayama, K., Narihisa, H. (2001) Performance of Simulated Annealing-Based Heuristic for the Unconstrained Binary Quadratic Programming Problem. European Journal of Operational Research **134** : 103–119
10. Katayama, K., Tani, M., Narihisa, H. (2000) Solving Large Binary Quadratic Programming Problems by Effective Genetic Local Search Algorithm. In: Proceedings of the 2000 Genetic and Evolutionary Computation Conference 643–650
11. Katayama, K., Narihisa, H. (2001) A Variant *k-opt* Local Search Heuristic for Binary Quadratic Programming. Trans. IEICE (A) **J84-A** : 430–435 (*in Japanese*)
12. Katayama, K., Narihisa, H. (2001) On Fundamental Design of Parthenogenetic Algorithm for the Binary Quadratic Programming Problem. In: Proceedings of the 2001 Congress on Evolutionary Computation 356–363
13. Kernighan, B.W., Lin, S. (1970) An Efficient Heuristic Procedure for Partitioning Graphs. Bell System Technical Journal **49** : 291–307
14. Kochenberger, G., Glover, F. (1999) Diversity Data Mining. Tech. Rep. HCES-03-99, Hearin Center for Enterprise Science College of Business Administration
15. Kuo, C.-C., Glover, F., Dhir, K.S. (1993) Analyzing and Modeling the Maximum Diversity Problem by Zero-One Programming. Decision Sciences **24** : 1171–1185
16. Lin, S., Kernighan, B.W. (1973) An Effective Heuristic Algorithm for the Traveling Salesman Problem. Operations Research **21** : 498–516
17. Lodi, A., Allemand, K., Liebling, T.M. (1999) An Evolutionary Heuristic for Quadratic 0-1 Programming. European Journal of Operational Research **119** : 662–670
18. Merz, P., Freisleben, B. (1999) Genetic Algorithms for Binary Quadratic Programming. In: Proceedings of the 1999 Genetic and Evolutionary Computation Conference 417–424
19. Merz, P., Freisleben, B. (2002) Greedy and Local Search Heuristics for Unconstrained Binary Quadratic Programming. Journal of Heuristics **8** : 197–213
20. Merz, P., Katayama, K. (2002) Memetic Algorithms for the Unconstrained Binary Quadratic Programming Problem. BioSystems (submitted for publication)
21. Pardalos, P.M., Rodgers, G.P. (1992) A Branch and Bound Algorithm for the Maximum Clique Problem. Computers and Operations Research **19** : 363–375
22. Pardalos, P.M., Xue, J. (1994) The Maximum Clique Problem. Journal of Global Optimization **4** : 301–328
23. Weitz, R.R., Lakshminarayanan, S. (1997) An Empirical Comparison of Heuristic and Graph Theoretic Methods for Creating Maximally Diverse Groups, VLSI Design, and Exam Scheduling. Omega **25** : 473–482

Multimeme Algorithms Using Fuzzy Logic Based Memes For Protein Structure Prediction

David A. Pelta[1] and Natalio Krasnogor[2]

[1] Departamento de Ciencias de la Computación e Inteligencia Artificial
Universidad de Granada, Spain
http:\\www.ugr.es\~dpelta
dpelta@ugr.es
[2] Automatic Scheduling, Optimisation and Planning Group
School of Computer Science and IT
University of Nottingham, U.K.
http:\\www.cs.nott.ac.uk\~nxk
natalio.krasnogor@nottingham.ac.uk

Summary. In this chapter we extend our previous studies on the self-adaptation of local searchers within a Memetic Algorithm.Self-adaptation allows the MA to learn which local searcher to use during search. In particular, we extend our results in [12], where memes were instantiated as Fuzzy-Logic based local searchers, and we show that our Multimeme algorithms are capable of producing new optimum solutions to instances of the *Protein Structure Prediction Problem in the HP-model.*

1 Introduction

Fuzzy Adaptive Neighborhood Search (*FANS*) was introduced in [4, 23]. Building upon local search, a classical method often used in optimization and operational research, and some basic elements of Fuzzy Sets theory, *FANS* was shown to be a robust optimization tool. This was noted for a variety of domains like knapsack problems [4], continuous function minimization [23] and more recently [23, 24, 26] in the protein structure prediction problem.

In our previous work [4], *FANS* was compared against a genetic algorithm. It was verified that both algorithms have similar performance for the range of problems studied. However, one of the advantages of using *FANS* is the easier implementation and parameter tuning. On the other hand, *FANS* performs its search by sampling one solution at a time which in some cases compromises its global search capabilities; as the Genetic Algorithm keeps a population of solutions it (more) consistently avoid local optima and performs a more global search.

In [12] we hybridized a Multimeme Algorithm [23, 28] with a simplified version of *FANS* in order to implement the pool of local searchers that the Mul-

timeme algorithm used. We demonstrated how *FANS*, and in turn fuzzy sets and systems ideas, could be successfully used to design a wide range of memes' behaviors. Moreover, we showed some benefits of using a Fuzzy-Evolutionary hybrid to tackle the Protein Structure Prediction problem (PSP).

The problem of predicting the three-dimensional structure of a protein is, perhaps, the single most important problem that biochemistry and bioinformatics face today. Even after almost five decades of intensive research it has not been "cracked". All-atom models of the folding process are extremely expensive. Moreover, there is no unique and ideal model for folding simulations, therefore, researchers use simplified descriptions of the phenomenon and tackle the slightly simpler (yet still intractable) problem of predicting the final structure of the folding process rather than the process itself. In this research we use such model, known as the HP-model [8]. The later has been widely used to benchmark folding and structure prediction algorithms and it was the source of important theoretical insights on the Protein Folding process [10].

This paper is organized as follows: in section 2 the protein structure prediction problem is introduced. Then in section 3 a brief descriptions of Memetic and Multimeme algorithms are presented. The hybrid approach we propose, i.e. a Multimeme Algorithm that includes *FANS* as local searchers, is described in Section 4. In order to assess the usefulness of the approach, several computational experiments were performed. These are described in Section 6 and the results discussed there. A section with conclusions ends the chapter.

2 The Protein Structure Prediction Problem

A protein is a chain of amino acid residues that folds into a specific *native* tertiary structure under certain physiological conditions. Proteins unfold when folding conditions provided by the environment are disrupted, and many proteins spontaneously re-fold to their native structures when physiological conditions are restored. This observation is the basis for the belief that prediction of the native structure of a protein can be done *computationally* from the information contained in the amino acid sequence alone.

In practice, solving the structure prediction problem means finding an adequate energy formulation (that correctly identifies native states) and being able to (by means of an adequate algorithm) search for candidate native states under that energy formulation. Exhaustive search of a protein's conformational space is clearly not a feasible algorithmic strategy for *PSP*. The number of possible conformations is exponential in the length of the protein sequence, and even powerful computational hardware is not capable of enumerating this space for even moderately large proteins. As an example consider the case where a protein structure is confined to a three dimensional cubic lattice. In this case, for a protein of length n there are potentially 4.7^n accessible conformations. Furthermore, recent computational analysis of *PSP* have

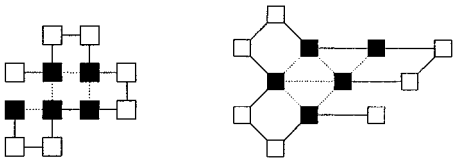


Fig. 1. HP sequence embedded in the square lattice and triangular lattice.

shown that this problem is intractable even on simple lattice models [1, 2, 7] such as the three dimensional case mentioned above.

A way of partially overcoming both the problem of the energy formulation and the enormous amount of candidate structures to analyze, is to use reduced protein models and knowledge-based potentials. Such simplified protein models are continuously playing an important role in improving our understanding of the fundamental physical properties of real-life proteins while paving the way for the development of algorithms to predict their native conformations using just the information of the amino acid sequence.

HP models abstract the hydrophobic interaction process in protein folding by reducing a protein to a heteropolymer that represents a predetermined pattern of hydrophobicity in the protein; non-polar amino acids are classified as hydrophobics and polar amino acids are classified as hydrophilics. A sequence is $s \in \{H, P\}^+$, where H represents a hydrophobic amino acid and P represents a hydrophilic amino acid.

The HP model restricts the space of conformations to self-avoiding paths on a lattice in which vertices are labelled by the amino acids. The energy potential in the HP model reflects the fact that hydrophobic amino acids have a propensity to form a hydrophobic core. To capture this feature of protein structures, the HP model adds a value ϵ for every pair of hydrophobes that form a topological contact; a topological contact is formed by a pair of amino acids that are adjacent on the lattice and not consecutive in the sequence. The value of ϵ is typically taken to be -1.

Figure 1 shows a sequence embedded in the square and the triangular lattice, with hydrophobic-hydrophobic contacts (HH contacts) highlighted with dotted lines. The conformation in Fig. 1(a) embedded in a square lattice, has an energy of -4, while the embedding in the triangular lattice (b) has an energy of -6 (there are 4 and 6 dotted lines, i.e. contacts, in the figure).

The particular version of the problem that we are going to tackle in this chapter is given by:

Maximum Protein Structure Prediction
Instance: A protein, i.e. a string over the alphabet $\{H, P\}$ ($s \in \{H, P\}^*$).
Solution: A self avoiding embedding of s into a 2D square lattice.
Measure: The number of Hs that are topological neighbors in the embedding (neighbors in the lattice but not consecutive in s)

Protein structure prediction has been shown to be NP-complete for a variety of simple lattice models (see Atkins and Hart [1] for a recent review), including the HP-model version on the square [7] and cubic lattices [2]. A wide variety of global optimization techniques have been applied to various models of the PSP problem, e.g. see the papers in Biegler et al. [3], Pardalos, Shalloway and Xue [21] and Pelta et al. [26]. Evolutionary algorithms (in their various forms) were shown to be particularly robust and effective global optimization techniques for molecular conformation problems. In particular, evolutionary methods have been used by several researchers engaged in proteomics related activities [9, 10, 11, 23, 15, 16, 22, 23, 27, 28, 29, 30, 31].

3 Memetic Algorithms

Memetic Algorithms are metaheuristics designed to find solutions to complex and difficult optimization problems. They are evolutionary algorithms that include a stage of individual optimization or learning as part of their search strategy. Memetic Algorithms are also called hybrid genetic algorithms, genetic local search, etc. A simple Memetic Algorithm scheme is shown in Fig. 2.

The inclusion of a local search stage into the traditional evolutionary cycle of crossover-mutation-selection is not a minor change of the evolutionary algorithm architecture. On the contrary, it is a crucial deviation that affects how local and global search is performed. The reader should also note that the pseudocode shown in Fig. 2 is just one possible way to hybridize a genetic algorithm with local search. In fact, a great number of distinct memetic algorithms' architectures have been presented in the literature and even integrated into formal models [23, 13].

An interesting variant of memetic algorithms are the Multimeme Algorithms (*MMA* in what follows) as introduced in [23, 28]. *MMA* are memetic algorithms where several types of local searchers, called memes, are available to the evolutive process during the local optimization phase. An individual in a *MMA* is composed of a genetic part, representing the solution to the problem being solved, and a memetic part, encoding a meme or local searcher, that is employed during the individual optimization stage.

The set of memes available to the algorithm is called the *memepool* and its design is a critical aspect for the success of the metaheuristic. Several design criteria for the memepool are described in [23]. Multimeme algorithms for the Protein Structure Prediction problem and Protein Structure Comparison Problem are reported in [11] and [5] respectively.

```
Memetic_Algorithm():
Begin
  t = 0;
  /* We put the evolutionary clock (generations), to null */
  Randomly generate an initial population P(t);
  Repeat Until ( Termination Criterion Fulfilled ) Do
    Compute the fitness f(p) ∀p ∈ P(t) ;
    Accordingly to f(p) choose a subset of P(T), store them in M(t);
    Recombine and variate individuals in M(t), store result in M'(t);
    Improve_by_local_search( M'(t));
    Compute the fitness f(p) ∀p ∈ M'(t) ;
    Generate P(t+1) selecting some individuals from P(t) and M'(t);
    t = t + 1;
  endDo
  Return best p ∈ P(t − 1);
End.
```

Fig. 2. A basic version of a memetic algorithm.

4 Fuzzy Memes for Multimeme Algorithms

The Fuzzy Adaptive Neighborhood Search Method (*FANS*) [4, 25] is a local search procedure which differs from other local searchers in two aspects. The first aspect is how the solutions are evaluated. Within *FANS* a fuzzy valuation representing some (maybe fuzzy) property P is used together with the objective function to obtain a "semantic evaluation" of the solution. In this way, we may talk about solutions satisfying P to a certain degree. Thus, the neighborhood of a solution effectively becomes a fuzzy set with the neighbor solutions as elements and the fuzzy valuation as the membership function.

The fuzzy valuation enables the algorithm to achieve the qualitative behavior of other classical local search schemes [4]. *FANS* moves between solutions satisfying P with at least certain degree, until it became trapped in a local optimum. In this situation, the second novel aspect arises: the operator used to construct solutions is changed, so solutions coming from different neighborhoods are explored next. This process is repeated once for each of a set of available operators until some finalization criterion for the local search is met.

The simplified scheme of *FANS* used here is shown in Fig. 3. The execution of the algorithm finishes when some external condition holds. In this research this happens when the number of cost function evaluations reached a prespecified limit. Each iteration begins with a call to the *neighborhood scheduler* NS, which is responsible for the generation and selection of the next solution in the optimization path. The call is done with parameters S_{cur} (the current solution), $\mu()$ (the fuzzy valuation), and $\mathcal{O}^k$ (a parameterized operator which is used to construct solutions). The neighborhood scheduler can return two

```
Procedure FANS:
Begin
  InitVariables();
  k:=maxK;;
  While ( not-end ) Do
    /* The neighborhood scheduler NS is called */
    S_new = NS(O^k, μ, S_cur);
    If (S_new is good enough in terms of μ()) Then
      S_cur := S_new;
      adaptFuzzyValuation(μ(), S_cur);
      Else
      /* NS could not obtain a good enough solution */
      /* The operator will be changed modifying the parameter k */
      If ((k=1)) Then
        k:= maxK;;
        Else
        k := k-1;;
      endIf
    endIf
  endDo
End.
```

Fig. 3. Scheme of *FANS*

alternative results; either a good enough (in terms of $\mu()$) solution (S_{new}) was found or not.

In the first case S_{new} is taken as the current solution and $\mu()$ parameters are adapted. In this way, the fuzzy valuation is changed as a function of the state of the search. This mechanism allows the local search stages to adapt during the search, hence accordingly to [23] the *FANS* based memes are adaptive helpers. If NS failed to return an acceptable solution (no solution was good enough in the neighborhood induced by the operator), the parameters of the operator are changed. In the full version of *FANS*, the strategy for this adaptation is encapsulated in the so called *operator scheduler* OS. Here we simply decrease the value of the parameter k of the operator $\mathcal{O}$. Effectively this induces, for each fixed operator, a variable radius search. At the beginning, the radius of the search is wide and it will be reduced as the search progresses. The next time NS is executed, it will have a modified operator (i.e., a different radius) to search for solutions.

The reader should note that what varies at each iteration are the parameters used in the $\mathcal{NS}$ call. The algorithm starts with $\mathcal{NS}$ $(s_0, \mathcal{O}^{t_0}, \mu_0)$. If $\mathcal{NS}$ could retrieve an acceptable neighborhood solution, the next iteration the call will be $\mathcal{NS}$ $(s_1, \mathcal{O}^{t_0}, \mu_1)$, the current solution is changed and the fuzzy valuation is adapted. If $\mathcal{NS}$ failed to retrieve an acceptable neighborhood solution

(at certain iteration l), the operator scheduler will be executed returning a modified version of the operator, so the call will be $\mathcal{NS}\ (s_l, \mathcal{O}^{t_1}, \mu_l)$.

5 Description of the Memepool

Multimeme algorithms (the overall strategy guiding the search behind our approach) have been described in detail elsewhere [5, 11, 23, 28], so we only describe here the memepool our *MMA* employs.

The memes of our *MMA* are implemented as simplified versions of *FANS* as a way to obtain a wide range of behaviors in a simple and unified fashion [12]. For the neighborhood scheduler, a *First* strategy was implemented: given the current solution s, the scheduler samples the search space with the operator $\mathcal{O}$ and returns the first solution satisfying $\mu(f(s), f(O(s))) \geq \lambda$ using at most certain number of trials (length of the local search), defined here as $n/2$ where n is the size of the instance. The value λ represents the minimum level of acceptability required for a solution to be considered as a "good enough" solution.

Each meme is identified by a 3-tuple:

$$(< basic\,operator >< fuzzy\,valuation >< value\,of\,\lambda >) \qquad (1)$$

where each element will be described below.

The $<$ *basic operator* $>$ can be instantiated to anyone of the following basic moves:

0. $Reflex(i,k)$: This operator reflects the protein structure across one of its symmetry axes. The change takes place between residues i and $i+k$.
1. $Shuffle(i,k)$: This operator performs a random re-positioning of the residues i^{th} to $(i+k)^{th}$.
2. $Stretch(i,k)$: The stretch operator unfolds a substructure of length k starting from residue i.
3. $Pivot(k)$: The pivot operator represents a rigid rotation. In this case, k random residues are selected and rigid rotations are performed sequentially on each one of them.

The operator has a parameter k indicating the number of positions to change. This value of k will be modified when the neighborhood scheduler fails to return an acceptable solution. In this case, the value of k is decremented by 1. When the failure occurs with $k = 1$, the value is set again to $k = maxK$.

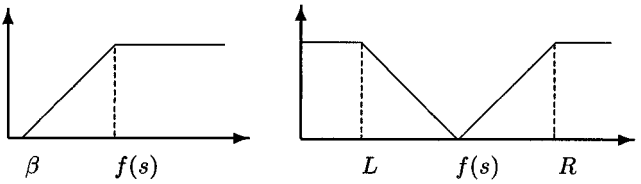


Fig. 4. Fuzzy Valuations μ_1 (left) and μ_2 (right).

There are two options available for the item $< fuzzy\ valuation >$:

1. The first fuzzy valuation proposed, μ_1, has the following definition:

$$\mu_1(s,q) = \begin{cases} 0.0 & \text{if } f(q) < \beta \\ (f(q)-\beta)/(f(s)-\beta) & \text{if } \beta \leq f(q) \leq f(s) \\ 1.0 & \text{if } f(q) > f(s) \end{cases} \quad (2)$$

where β is a threshold specifying what is, and what is not, considered an acceptable deterioration in solution quality. Given that the energy of a structure can take negative values (e.g. when the structure is not self-avoiding), the parameter β has two definitions[3]: when $f > 0$ then $\beta = f * 0.5$ (a deterioration in cost of 50% is allowed); when $f < 0$ then $\beta = f * 1.2$ (a deterioration in cost of 20% is allowed). This fuzzy valuation promotes acceptability to solutions improving the current cost. When used with $\lambda = 1$, it induces in *FANS* a hillclimber like behavior, allowing transitions only to improving solutions. The graphical representation of μ_1 is shown in Fig. 4 (left).

2. The second fuzzy valuation proposed, μ_2, has the following definition:

$$\mu_2(s,q) = \begin{cases} 1.0 & \text{if } R \leq f(q) \leq L \\ (f(s)-f(q))/(f(s)-L) & \text{if } L < f(q) \leq f(s) \\ (f(q)-f(s))/(R-f(s)) & \text{if } f(s) < f(q) \leq R \end{cases} \quad (3)$$

here, the parameters L and R are defined as follows: when $f(s) > 0$ then $L = f(s) * 0.5$ and $R = f(s) * 1.5$; when $f(s) > 0$ then $L = f(s) * 1.5$ and $R = f(s) * 0.5$. This fuzzy valuation promotes diversity, in the sense of cost. Solutions similar in cost to the current one, get very low degrees of acceptability and those differing in more than 50% with respect to the cost of the current solution gets the highest degree of acceptability. The graphical representation of μ_2 is shown in Fig. 4 (right).

[3] Although in Protein Structure Prediction one tries to minimize the energy of the conformation, in this chapter we recast the problem to a maximization problem by simply multiplying the energies by -1.

Table 1. HP model test Instances for the 2D Square Lattice.

Instance	Sequence	Opt	Size
I 1	PPHPPHHPPHHPPPPPHHHHHHHH HHPPPPPPHHPPHHPPHPPHHHH	-22	48
I 2	HHPHPHPHPHHHHPHPPPHPPPHPP PPHPPPHPPPHPHHHHPHPHPHPH	-21	50
I 3	PPHHHPHHHHHHHHPPPHHHHHHHH HHPHPPPHHHHHHHHHHHHPPPPHH HHHHPHHPHP	-34	60
I 4	HHHHHHHHHHHHPHPHPPHHPPHHP PHPPHHPPHHPPHPPHHPPHHPPHP HPHHHHHHHHHHHH	-42	64

The last element to define a meme is the $< value\,of\ \lambda >$. This parameter defines the minimum level of acceptability that a solution needs to be considered as the next solution in the search. Each pair $(\mu(), \lambda)$ defines a particular behavior for the meme. For example, with the fuzzy valuation μ_1 and $\lambda = 1$, the meme will only accept transitions to improving solutions. As $\lambda \to 0$ the chance to move to cost deteriorating solutions is increased. We can say that as λ increases, the use of μ_1 leads to exploitative memes. In turn, the use of the fuzzy valuation μ_2 leads to explorative memes. The higher values of acceptability are assigned to those solution with quite different cost with respect to that of the current solution. The lower values correspond to solutions similar in cost. In this work we consider three values for λ, where $\lambda \in \{0.4, 0.8, 1.0\}$.

Here, we want to stress the overall intended dynamics of our metaheuristic:

- At the local level (i.e. the process of individual local search) FANS memes perform a fuzzy-based variable-operator **and** variable-radius local search.
- At the global level (i.e the process of population evolutionary search) the Multimeme Algorithm is co-adapting solutions to the Protein Structure Prediction and the best local searcher (i.e. meme) to use in each individual at different stages of the search.

The metaheuristic searches concurrently on both solution and searcher spaces.

6 Description of Experiments and Results

The experiments were done with the four instances of the HP model in the square lattice shown in Table 1. For each one, the length of the sequence and the optimum value of the corresponding structure are described.

We perform two experiments which differ in the size of the mating pool. In the first one, the memepool has 12 memes which arise from the combination of the four basic moves, the fuzzy valuation μ_1 and the three values of λ. In the

second experiment, the memepool size is 24 after adding 12 more memes which arise from the use of the fuzzy valuation μ_2. In this way, we are incorporating memes promoting diversification. We use also the following parameters that were used in previous experiments by the authors [12]:

1. Replacement Strategy: $(\mu = 350, \lambda = 350)$
2. Depth of the local search, i.e. number of iterations performed by each meme application: 3
3. Length of the local search, i.e. max. number of trials allocated in the neighborhood scheduler of *FANS*: $n/2$, with n the length of the sequence.

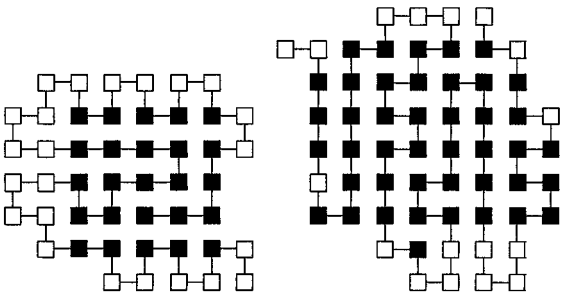
For each memepool size and instance we performed 30 runs of the *MMA*. Each run was allocated 200 generations. The initial population was generated randomly and consisted of 350 individuals. Two(consecutive)-point mutations and two-point crossover were employed with probabilities 0.2 and 0.8, respectively. In the case of mutation, the probability was per individual. The innovation rate was set to $IR = 0.2$. Tournament selection was used to select the mating parents and a tournament size of 2 individuals was used.

Three values were recorded at the end of every run: *bestF*, the fitness of the best solution found; *e2b*, the number of fitness evaluations used to reach the best solution; and *eDone*, the total amount of evaluations done in the whole run.

Tables 2, 3, 4 show the average, standard deviation, minimum and maximum values obtained for each variable over 30 runs. Each row is named $I < x > m < y >$ where $x \in \{1, 2, 3, 4\}$ stands for the instance used and $y \in \{12, 24\}$ represents the memepool size used within the *MMA*. The results obtained using the implementation of *FANS* presented in [12] are also included. *FANS* was executed 30 times, where each instance ended after 2^6 evaluations.

The first thing to notice is that for instances 1 and 3, structures with higher bonds than the known optima were obtained. For instance 1, a structure with 23 bonds was found while for instance 3, one with 35 bonds was obtained. Fig. 5 shows both structures. To the best of our knowledge, such optimal values were only achieved before in [20]. However, in that paper the authors do not measure the cost of their algorithm in number of total energy evaluations so it is impossible to provide comparisons. Moreover, they enforce a strong bias in the search to regions of the search space that contain secondary structure information derived from the native structure they are searching for. In other words, they used specific domain knowledge that is not present in our algorithm. Having this in mind, we deem our algorithms as the first blind search method (to the best of the authors knowledge) able to obtain these novel native structures.

The results in terms of *bestF* were quite similar using 12 or 24 memes. The algorithms using 24 memes had a slightly higher standard deviation and lower minimum values. The 12 memes version, achieved a higher maximum



(a) (b)

Fig. 5. New best structures obtained for instances 1, in (a), and instance 3, in (b).

Table 2. Statistics for *bestF*

Algorithm	Mean	SD	Min	Max
i1m12	20.55	0.97	19.02	23.02
i1m24	20.02	1.23	18.01	23.02
fans-i1	19.35	0.76	18.02	21.02
i2m12	19.98	1.00	18.01	21.01
i2m24	19.98	0.98	17.01	21.01
fans-i2	18.94	0.74	18.01	20.01
i3m12	32.82	0.96	31.02	35.02
i3m24	32.15	1.20	29.02	34.02
fans-i3	30.82	0.81	29.02	32.02
i4m12	33.45	1.65	30.02	38.02
i4m24	33.47	2.38	28.02	38.02
fans-i4	28.75	1.08	27.02	32.02

value on instance 3. *FANS* achieved the lowest values of standard deviation, but the higher ones in terms of the mean.

In terms of $e2b$, it is clear that the use of 24 memes allowed it to reach good results with less effort. This situation is reasonable if we consider in the number of trials that each meme has to perform to obtain an acceptable solution. Those memes using μ_2 can obtain acceptable solutions quite easily. For example, given a value $\lambda = 1$, the memes using μ_1 need to find solutions improving the cost, which may result in the use of a high number of trials. On the contrary, memes using μ_2 will accept any transition leading to a decrease in cost of more than 50%, and this is easy to achieve using a low number of trials. This aspect is confirmed looking at the statistics for *eDone*. Considering the mean, the *MMA* with 12 memes used approximately 2.35 million evaluations while the *MMA* with 24 never used more than 2 millions.

To finish the analysis, two additional aspects are considered. First, Fig. 6 shows the evolution of the *average cost of the best individual* through the generations. It can be seen that the use of 12 or 24 memes leads to very similar

Table 3. Statistics for *e2b*

Algorithm	Mean	SD	Min	Max
i1m12	1356104	481405	549691	2269640
i1m24	894661	331121	367357	1510390
fans-i1	933126	526296	160522	1960500
i2m12	1109461	545856	362400	2565580
i2m24	675173	198261	447300	1509880
fans-i2	1202764	589958	243614	2000050
i3m12	1358837	563218	605457	2790530
i3m24	990789	343991	356614	1599350
fans-i3	1066657	590136	65746	1940400
i4m12	1852648	399233	991046	2356310
i4m24	1520624	376036	469575	2005740
fans-i4	1074369	562966	76854	1965090

Table 4. Statistics for *eDone*

Algorithm	Mean	SD	Min	Max
i1m12	2310653	218389	1864000	2579590
i1m24	1581614	90237	1396960	1758450
i2m12	2384182	166804	2005530	2739440
i2m24	1630939	79154	1507610	1859810
i3m12	2511524	214088	1997610	2873460
i3m24	1678238	86323	1457230	1867790
i4m12	2468762	213454	2168150	3258060
i4m24	1964042	77697	1805970	2147600

patterns of evolution. Looking at the graph for instance 4, we can conclude that the *MMA* has not converged when the run finished. This fact may be considered an explanation about the quite low values of *bestF* obtained.

Second, Fig. 7 shows the evolution of the *average cost of the average fitness of the whole population* through the generations. It is clear that the use of diversification memes kept the overall fitness lower (i.e. better solutions)

7 Conclusions

A hybridization strategy between a fuzzy sets-based heuristic, and a Multimeme algorithm was proposed and tested.

The construction of the memepool using simplified versions of *FANS* enabled us to obtain a wide range of fuzzy memes, each one with its particular behavior. The advantage of using *FANS* as the memes for a *MMA* over using adaptive helpers as in [23, 27] is that it is much easier to tune the search of the memes. Moreover, human knowledge or instance specific knowledge (e.g. secondary structure information as that used in [20]) can be readily incorporated into *FANS* based memes.

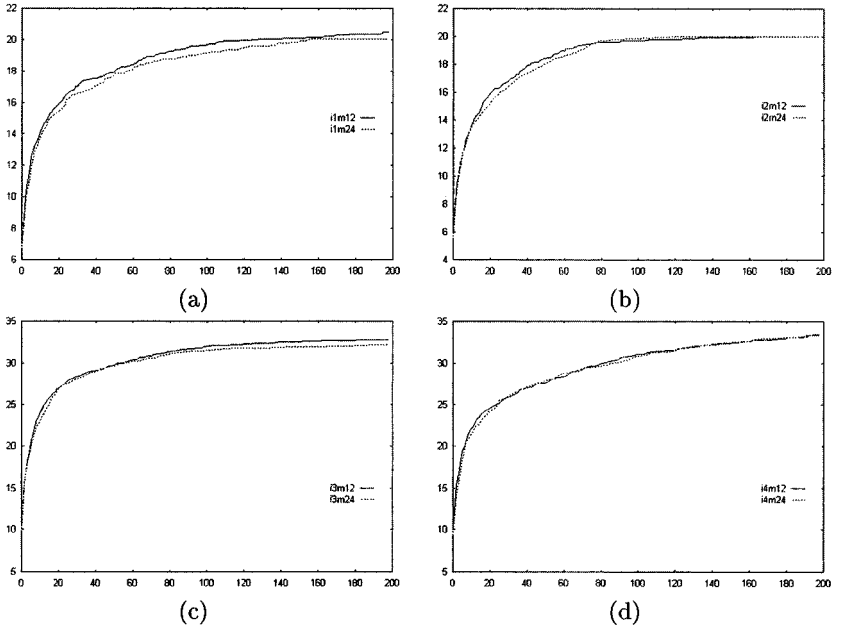


Fig. 6. Evolution of the cost of the best individual vs Generations for test instances 1 (a), 2 (b), 3 (c) and 4 (d) using a *MMA* with 12 and 24 memes.

The scheduling of memes by the simple inheritance mechanism was proven successful in the detection of the most suitable fuzzy meme for different stages of the search. This has been verified in other domains [23, 28], which deems Multimeme Algorithms a very robust metaheuristic.

The coupled effect of both elements lead to a robust and general purpose metaheuristic. In the test cases shown in this chapter it was able to improve previous results in the protein structure prediction problem. We suggest that this approach can be a powerfull metaheuristic for other combinatorial problems.

8 Acknowledgments

This research was partially founded by Fundacion Antorchas, Republica Argentina and supported in part by Project TIC2002-04242-CO3-02.

D.A. Pelta is a grant holder from Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), Republica Argentina.

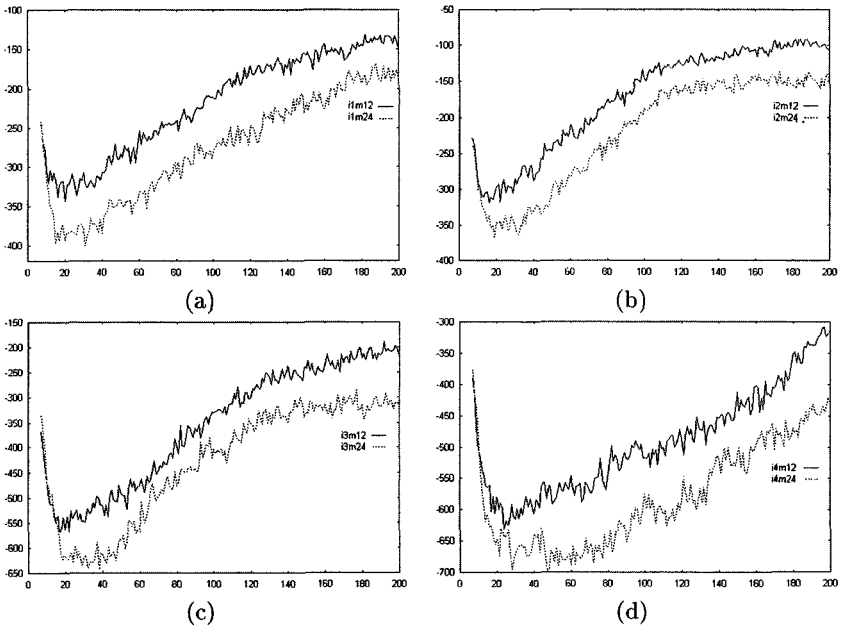


Fig. 7. Evolution of the cost of the average fitness of the population vs Generations for test instances 1 (a), 2 (b), 3 (c) and 4 (d) using a *MMA* with 12 and 24 memes.

References

1. J. Atkins and W. E. Hart. On the intractability of protein folding with a finite alphabet. *Algorithmica*, pages 279–294, 1999.
2. B. Berger and T. Leight. Protein folding in the hydrophobic-hydrophilic (HP) model is NP-complete. In *Proceedings of The Second Annual International Conference on Computational Molecular Biology, RECOMB 98*, pages 30–39. ACM Press, 1998.
3. L. T. Biegler, T. F. Coleman, A. R. Conn, and F. N. Santosa, editors. *Large-Scale optimization with applications. Part III: Molecular structure and optimization*, volume 94 of *The IMA Volumes in Mathematics and its Applications*. Springer-Verlag, New York, 1997.
4. A. Blanco, D. Pelta, and J. Verdegay. A fuzzy valuation-based local search framework for combinatorial problems. *Journal of Fuzzy Optimization and Decision Making*, 1(2):177–193, 2002.
5. B. Carr, W.E. Hart, N. Krasnogor, E. Burke, J. Hirst, and J. Smith. Alignment of protein structures with a memetic evolutionary algorithm. In *GECCO-2002: Proceedings of the Genetic and Evolutionary Computation Conference*. Morgan Kaufman, 2002.
6. T. E. Creighton, editor. *Protein Folding*. W. H. Freeman and Company, 1993.
7. P. Crescenzi, D. Goldman, C. Papadimitriou, A. Piccolboni, and M. Yannakakis. On the complexity of protein folding. In *Proceedings of The Second Annual International Conference on Computational Molecular Biology, RECOMB 98*, pages 51–62. ACM Press, 1998.

8. K. A. Dill. Theory for the folding and stability of globular proteins. *Biochemistry*, 24:1501, 1985.
9. G. Greenwood, B. Lee, J. Shin, and G. Fogel. A survey of recent work on evolutionary approaches to the protein folding problem. In *Proceedings of the Congress of Evolutionary Computation (CEC)*, pages 488–495. IEEE, 1999.
10. M. Khimasia and P. Coveney. Protein structure prediction as a hard optimization problem: The genetic algorithm approach. In *Molecular Simulation*, volume 19, pages 205–226, 1997.
11. N. Krasnogor. *Studies on the Theory and Design Space of Memetic Algorithms.* Ph.D. Thesis, University of the West of England, Bristol, United Kingdom. (http://dirac.chem.nott.ac.uk/~natk/Public/papers.html), 2002.
12. N. Krasnogor, B. Blackburne, E. Burke, and J. Hirst. Multimeme algorithms for protein structure prediction. In *Proceedings of the Parallel Problem Solving from Nature VII. Lecture notes in computer science*, 2002.
13. N. Krasnogor, W.E. Hart, J. Smith, and D. Pelta. Protein structure prediction with evolutionary algorithms. In W. Banzhaf, J. Daida, A. Eiben, M. Garzon, V. Honavar, M. Jakaiela, and R. Smith, editors, *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference.* Morgan Kaufman, 1999.
14. N. Krasnogor and D. Pelta. Fuzzy memes in multimeme algorithms: a fuzzy-evolutionary hybrid. In J. Verdegay, editor, *Fuzzy Sets based Heuristics for Optimization*, Studies in Fuzziness and Soft Computing, pages 49–66. Physica Verlag, 2003.
15. N. Krasnogor, D. Pelta, P. M. Lopez, P. Mocciola, and E. de la Canal. Genetic algorithms for the protein folding problem: A critical view. In C. F. E. Alpaydin, editor, *Proceedings of Engineering of Intelligent Systems.* ICSC Academic Press, 1998.
16. N. Krasnogor, D. Pelta, D. H. Marcos, and W. A. Risi. Protein structure prediction as a complex adaptive system. In *Proceedings of Frontiers in Evolutionary Algorithms 1998*, 1998.
17. N. Krasnogor and J. Smith. A memetic algorithm with self-adaptive local search: TSP as a case study. In *Proceedings of the 2000 Genetic and Evolutionary Computation Conference.* Morgan Kaufmann, 2000.
18. N. Krasnogor and J. Smith. Emergence of profitable search strategies based on a simple inheritance mechanism. In *Proceedings of the 2001 Genetic and Evolutionary Computation Conference.* Morgan Kaufmann, 2001.
19. N. Krasnogor and J. Smith. Memetic algorithms: Syntactic model and taxonomy. 2001. submitted to The Journal of Heuristics. Available from the authors.
20. F. Liang and W. Wong. Evolutionary monte carlo for protein folding simulations. *Journal of Chemical Physics*, 115(7):3374–3380, 2001.
21. P. M. Pardalos, D. Shalloway, and G. L. Xue, editors. *Global minimization of nonconvex energy functions: Molecular conformation and protein folding*, volume 23 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science.* American Mathematical Society, Providence, Rhode Island, 1996.
22. A. L. Patton. A standard ga approach to native protein conformation prediction. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 574–581. Morgan Kauffman, 1995.
23. D. Pelta, A. Blanco, and J. L. Verdegay. A fuzzy adaptive neighborhood search for function optimization. In *Fourth International Conference on Knowledge-Based Intelligent Engineering Systems & Allied Technologies, KES 2000*, volume 2, pages 594–597, 2000.

24. D. Pelta, A. Blanco, and J. L. Verdegay. Applying a fuzzy sets-based heuristic for the protein structure prediction problem. *Internation journal of Intelligent Systems*, 17(7):629–643, 2002.
25. D. Pelta, A. Blanco, and J. L. Verdegay. Fuzzy adaptive neighborhood search: Examples of application. In J. L. Verdegay, editor, *Fuzzy Sets based Heuristics for Optimization*, Studies in Fuzziness and Soft Computing, pages 1–20. Physica-Verlag, 2003.
26. D. Pelta, N. Krasnogor, A. Blanco, and J. L. Verdegay. F.a.n.s. for the protein folding problem: Comparing encodings and search modes. In *Fourth International Metaheuristics Conference, MIC 2001*, 2001.
27. A. Piccolboni and G. Mauri. Protein structure prediction as a hard optimization problem: The genetic algorithm approach. In N. e. a. Kasabov, editor, *Proceedings of ICONIP '97*. Springer, 1998.
28. A. A. Rabow and H. A. Scheraga. Improved genetic algorithm for the protein folding problem by use of a cartesian combination operator. *Protein Science*, 5:1800–1815, 1996.
29. S.-K. S. Genetic algorithms for protein tertiary structure prediction. In *Parallel Problem Solving from Nature - PPSN II*. North-Holland, 1992.
30. R. Unger and J. Moult. A genetic algorithm for three dimensional protein folding simulations. In *Proceedings of the 5th International Conference on Genetic Algorithms (ICGA-93)*, pages 581–588. Morgan Kaufmann, 1993.
31. R. Unger and J. Moult. Genetic algorithms for protein folding simulations. *Journal of Molecular Biology*, 231(1):75–81, 1993.

A Memetic Algorithm Solving the VRP, the CARP and General Routing Problems with Nodes, Edges and Arcs

Christian Prins and Samir Bouchenoua

LOSI, University of Technology of Troyes
BP 2060, 12 Rue Marie Curie
F-10010 Troyes Cedex, France
{Christian.Prins, Samir.Bouchenoua}@utt.fr

Summary. The VRP (Vehicle Routing Problem) and the CARP (Capacitated Arc Routing Problem) involve the routing of vehicles in an undirected network to service respectively a set of nodes or a set of arcs. Motivated by applications in waste collection, we define a more general model called NEARP (Node, Edge and Arc Routing Problem) for tackling mixed graphs with required nodes, edges and arcs. A memetic algorithm (MA) is developed for the NEARP. An evaluation on standard VRP and CARP benchmarks shows that the MA is competitive with most metaheuristics for these particular cases of the NEARP. We finally propose a set of NEARP instances, together with the solutions costs achieved by the MA, as a challenge for other researchers in vehicle routing.

Key words: memetic algorithm, vehicle routing, general routing problem.

1 Introduction

Traditionally, the literature devoted to multi-vehicle routing problems considers an undirected network and studies two distinct families of problems: *node routing problems* and *arc routing problems*, depending on the entities to be serviced in the network.

The *VRP* or *Vehicle Routing Problem* is a typical representative of node routing problems. It is usually defined on an undirected network in which some nodes correspond to customers. Each customer has a weight or demand for a commodity and a service cost. Each network edge has a travel cost. A fleet of identical vehicles of limited capacity is based at a depot node. A trip for a vehicle starts at the depot, visits a sequence of customers, and returns to the depot. The cost of a trip includes the service costs of its customers and the costs of each traversed edge.

The VRP consists of designing a set of trips of least total cost, such that each customer is visited exactly once and the total demand serviced by any

trip does not exceed vehicle capacity. The VRP has important applications in logistics, for instance in distribution networks. It is unfortunately NP-hard and exact methods [1] have a limited interest, since some instances with 75 nodes (and even 50 nodes for the distance-constrained VRP) are not yet solved to optimality. This is why heuristics are required in practice for tackling real-life VRP instances. They comprise simple algorithms [2], like the merge heuristic from Clarke and Wright, and more recent and powerful metaheuristics like tabu search [3, 4, 5].

Comparatively, arc routing problems have been neglected for a long time by researchers, but they have raised a growing interest in the two last decades, mainly because of their applications like urban waste collection or winter gritting (see the good survey from Assad and Golden [6]). The problem corresponding to the VRP in arc routing is the *CARP* or *Capacitated Arc Routing Problem.* Its definition is similar but this time the tasks to be performed by the vehicles consist of servicing some edges, for instance spreading salt or collecting municipal refuse along a street.

The CARP is also NP-hard. Theoretically, it can be converted into an equivalent node routing problem as shown by Pearn et al. [7]. This transformation converts a CARP with k required arcs into a VRP with $3k + 1$ nodes. Since the VRP itself is very hard, this increase in size is of course not acceptable and most researchers prefer to attack the CARP directly. The CARP seems more difficult than the VRP in practice: the exact solution methods published are still limited to small instances with at most 20 edges [8]. On the other hand, Belenguer and Benavent [9] have exploited the rich underlying structure of this problem to design an excellent lower bound, allowing an accurate evaluation of heuristics.

As for the VRP, the simplest heuristics published for the CARP are constructive methods, e.g. Path-Scanning from Golden et al. [10], Augment-Merge from Golden and Wong [11] and Ulusoy's tour splitting heuristic [12]. Metaheuristics have been designed more recently, like the powerful tabu search algorithm CARPET from Hertz, Laporte and Mittaz [13] and the genetic algorithms (GAs) from Lacomme, Prins and Ramdane-Chérif [14, 15]. The best of these GAs is the only algorithm able to reach the lower bound of Belenguer and Benavent [9] on 21 out of 23 standard instances proposed by DeArmon [16], containing up to 55 required edges.

Despite the success of metaheuristics for the VRP and the CARP, it is clear that these two problems cannot formalize the requirements of many real-world scenarios. Consider for instance urban waste collection. Although most tasks consist of servicing streets, the problem cannot be modeled as a pure CARP because of punctual accumulations of waste that must be modeled as required nodes (hospitals, schools, supermarkets, etc.). Moreover, an undirected graph can only model 2-way streets whose both sides are collected in parallel and in any direction (*zigzag* or *bilateral* collection, a practice reserved to low-traffic residential areas). In reality, a street can be a 2-way street with bilateral collection (giving an edge in the modeled network), a 2-way street with two

sides collected independently (giving two opposite arcs), or even a 1-way street (giving one arc).

Our research is a step towards more generic models and algorithms able to handle such complications in vehicle routing. Section 2 presents our extended model, the *NEARP* or *Node, Edge and Arc Routing Problem*. It is defined on a mixed graph with required nodes, edges and arcs and contains the VRP and the CARP as particular cases. Section 3 describes three simple heuristics for the NEARP that are used to initialize the memetic algorithm (MA). The third one, a tour splitting method, plays also a key-role in chromosome evaluation. The MA itself is developed in section 4. It undergoes in section 5 a preliminary testing on standard VRP and CARP instances to check its competitiveness with respect to existing algorithms. A generator of instances for the new problem is described in section 6. We finally propose in section 7 a set of NEARP instances with the solution costs computed by the MA, as a challenge for OR researchers of the vehicle routing community. An appendix provides the reader with detailed tables of results and a list of formal definitions for all problems discussed.

2 The Node, Edge and Arc Routing Problem (NEARP)

This section formally defines the NEARP as a new problem generalizing both the VRP and the CARP and describes data structures for the algorithms of sections 3 and 4. The NEARP allows a mixed network with required nodes, edges and arcs. Contrary to the CARP, two distinct costs are handled for each link: one *deadheading cost*, i.e., the cost for a traversal without service (called *deadhead* by transporters) and one *service cost*, when the link is traversed to be treated. The entities to be serviced are directly tackled, i.e. the model does not rely on a conversion into a CARP or a VRP.

2.1 Problem statement

The NEARP is defined on a strongly connected and loopless mixed network $G = (N, E, A)$ with three sets of entities: a set N of n nodes, a set of edges E, and a set of arcs A. We call *links* the m entities in $E \cup A$. N includes a depot node s with a fleet of K identical vehicles of capacity W. The number of vehicles K is a decision variable. Each entity u has a non-negative traversal cost c_u. This cost is null for a node. For a link, it corresponds to a deadheading traversal (i.e., without service).

Some entities, the *tasks*, are *required*, i.e., they need to be processed by a vehicle. N_R, E_R and A_R respectively denote the subset of required nodes or *node-tasks*, the subset of required edges or *edge-tasks*, and the subset of required arcs or *arc-tasks*. Their cardinalities are respectively denoted by ν, ϵ and α. $\tau = \nu + \epsilon + \alpha$ denotes the total number of tasks. Each task $u = 1, 2, \ldots, \tau$ has a non-negative demand q_u and a non-negative processing cost p_u. To

ensure feasibility, we assume that no demand exceeds W. Theoretically, all costs and demands should be integers, but our implementation accepts real numbers to handle some Euclidean instances from literature in section 5.

Any feasible vehicle trip must start from the depot, process a sequence of tasks whose total demand does not exceed W, and return to the depot. Its cost includes the processing costs of its tasks (required nodes, edges and arcs) and the traversal costs of the links used to travel from the depot to the first task, from each task to the subsequent one, and from the last task to the depot. The next subsection introduces data structures allowing to specify the cost of a trip by a concise formula.

Any feasible solution is a set of feasible trips covering all tasks. Tasks cannot be preempted, i.e., each task must appear in exactly one trip and only once in the sequence of tasks of that trip. Recall that the number of trips actually used, K, is not imposed but is part of the solution. The cost of a solution is the sum of its trip costs.

The NEARP consists of determining a least-cost solution. Clearly, this is a new problem that generalizes the VRP and the CARP: the VRP is the particular case with $A = \emptyset$ and $E_R = \emptyset$, while the CARP corresponds to $A = \emptyset$ and $N_R = \emptyset$. The *General Routing Problem* (*GRP*) is another special case of the NEARP, introduced by Orloff in 1974 [17]. In this generalization of the well-known *Traveling Salesman Problem* (*TSP*), one single vehicle must visit a subset of nodes and a subset of edges in an undirected graph to minimize the total mileage. Hence, the NEARP could also be called *Mixed Capacitated GRP* or *MCGRP*.

2.2 Internal network representation

Our algorithms rely on an internal network in which all entities (nodes and links) are encoded with the same attributes and stored in a common list L, indexed from 1 to $n + |A| + 2|E|$. The attributes for entity u are a begin node b_u, an end node e_u, a traversal cost c_u, a demand q_u, a processing cost p_u and a pointer $inv(u)$ explained below.

By convention, we set $b_u = e_u$ and $c_u = 0$ if entity u is a node: no confusion with a link is possible, since G is loopless. The required entities (tasks) are the ones with non-zero demands. Each required edge is encoded as two opposite arcs u and z linked thanks to their pointers inv, i.e., $e_u = b_z$, $e_z = b_u$, $inv(u) = z$ and $inv(z) = u$. These two arcs inherit their demands and their costs from the edge. Any arc or non-required edge u is such that $inv(u) = 0$. If u is a node, then $inv(u) = u$ by convention. Therefore, the three sets of tasks can be concisely defined by equations 1–3.

$$N_R = \{u \in L : b_u = e_u \wedge q_u > 0 \wedge inv(u) = u\} \quad (1)$$

$$E_R = \{u \in L : b_u \neq e_u \wedge q_u > 0 \wedge inv(u) \neq u\} \quad (2)$$

$$A_R = \{u \in L : b_u \neq e_u \wedge q_u > 0 \wedge inv(u) = 0\} \quad (3)$$

The costs of the shortest paths between any two entities can be precomputed between their two end-nodes using Dijkstra's algorithm [18], resulting in a distance matrix D, $n \times n$. A trip θ is defined as a list $(\theta_1, \theta_2, \ldots, \theta_t)$ of task indexes, with a total demand $load(\theta) \leq W$ and a total cost $cost(\theta)$ defined by equations 4 and 5. Implicitly, θ starts and ends at the depot and shortest paths are assumed to connect the successive steps. A solution T is a list $(T_1, T_2, \ldots, T_K)$ of K vehicle trips (recall that K is a decision variable). Its cost is the sum of its trip costs. Each task appears exactly once in T and each edge-task occurs as one of its two opposite arcs.

$$load(\theta) = \sum_{i=1}^{t} q(\theta_i) \tag{4}$$

$$cost(\theta) = d(s, b(\theta_1)) + \sum_{i=1}^{t-1} (p(\theta_i) + d(e(\theta_i), b(\theta_{i+1}))) + p(\theta_t) + d(e(\theta_t), s) \tag{5}$$

3 Three simple heuristics for the NEARP

These heuristics are briefly described before the MA, because they are used to provide the initial population of the MA with good solutions. Moreover, the splitting technique of the third heuristic is also used in the MA for chromosome evaluation.

3.1 Nearest neighbor heuristic

Our *Nearest Neighbor Heuristic* or *NNH* adapts to the NEARP the Path-Scanning heuristic proposed by Golden and Wong for the CARP [10]. NNH is a sequential heuristic building the trips one by one until all tasks are processed. In building each trip, the sequence of tasks is extended at each iteration by joining the nearest free task z, until vehicle capacity W is exhausted. In NEARP instances with a majority of required links, the distance between the last task of the trip and the nearest free tasks is often zero, for instance when the tasks correspond to adjacent streets.

So, five rules are used to break ties among nearest tasks: 1) maximize the distance d_{zs} to the depot, 2) minimize this distance, 3) maximize a kind of yield q_z/p_z, 4) minimize this yield, 5) use rule 1 if the vehicle is less than half-full, else use rule 2. NNH computes one complete NEARP solution for each rule and returns the best one. A small example is given for rule 1 in Figure 1. Each black square represents a required node and each thick segment a required link. Thin lines correspond to shortest paths. The last task of the trip in construction is link u. The two nearest free tasks are node a and edge b, since $d_{ua} = d_{ub} = 3$. NNH will select edge b because $d_{bs} > d_{as}$.

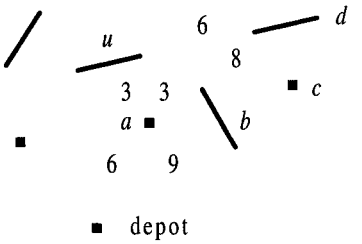


Fig. 1. Basic step of heuristic NNH with rule 1.

3.2 Merge heuristic

Our *Merge heuristic* or *MH* corresponds to the Clarke and Wright method for the VRP [2] and to the Augment-Merge heuristic for the CARP [11]. It starts with a trivial solution with τ trips reduced to one task. Then, each iteration evaluates the merger (concatenation) of any two trips, subject to W. For instance, in Figure 2, merging T_i and T_j yields a saving of $8+6-10=4$. MH merges the two trips with the largest positive savings. This process is repeated until no such merger is possible.

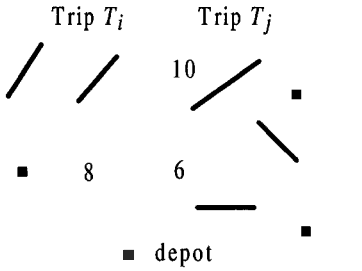


Fig. 2. Concatenation of two trips in the Merge Heuristic (MH).

Note that there exist up to 8 possible mergers for two trips T_i and T_j: one can put T_i before or after T_j and each trip may be inverted or not. In fact, the direction of each edge-task is changed in an inverted trip: e.g., if a trip contains a subsequence of two edge-tasks (u, z), then the inverted trip will contain the subsequence $(inv(z), inv(u))$. This also holds for a node u with $inv(u) = u$. Finally, the only case where a trip cannot be inverted is the presence of at least one arc-task u, since $inv(u) = 0$. In a real network, this occurs when a trip goes thru one-way streets.

3.3 Tour splitting heuristic

The *Tour Splitting Heuristic* or *TSH* extends a CARP algorithm from Ulusoy [12]. First, TSH relaxes vehicle capacity to build a giant tour S servicing

all tasks. This can be done by any heuristic, for instance NNH called with $W = \infty$. Figure 3 shows such a giant tour $S = (a, b, c, d, e)$, with two node-tasks b and d and three required links a, c and e. The demand and processing cost of each task are given in brackets. An optimal procedure *Split* is then called to cut S into capacity-feasible trips.

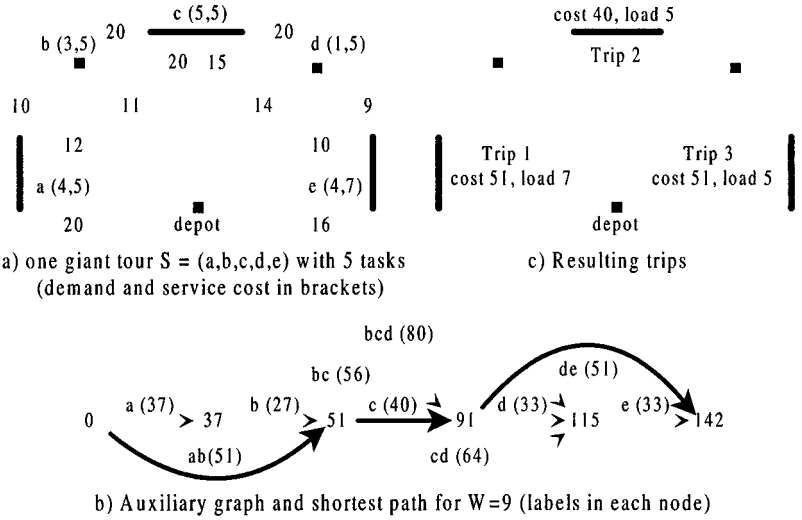


Fig. 3. Principle of the Tour Splitting Heuristic (TSH).

Split builds an auxiliary graph H with $\tau + 1$ nodes indexed from 0 to τ. Each subsequence $(S_i, S_{i+1}, \ldots, S_j)$ of S that could give a capacity-feasible trip gives in H as one arc $(i-1, j)$, weighted by the cost of the trip. This auxiliary graph is given in figure 3 for $W = 9$. Since H is acyclic by definition and contains $O(\tau^2)$ arcs, a shortest path from node 0 to node τ can be computed in $O(\tau^2)$ using Bellman's algorithm [18]. The resulting shortest path (boldface) indicates where to split the giant trip. It corresponds to a solution with 3 trips and a total cost equal to 142. Our implementation of TSH splits 5 giant trips, obtained by calling NNH with an infinite capacity and one priority rule at a time (see subsection 3.1). The best solution obtained is returned.

4 A memetic algorithm for the NEARP

4.1 Chromosomes and evaluation

A chromosome is simply defined as a sequence S of τ task indexes, *without trip delimiters*. It is almost a permutation chromosome because each task

appears exactly once in S. However, each edge-task may appear as one of its two opposite arcs. Clearly, S does not directly represent a valid NEARP solution but it can be considered as a giant tour for a vehicle of infinite capacity. The *Split* procedure described in 3.3 for the TSH heuristic is used to extract from S the best possible NEARP solution. The guiding function $F(S)$ is nothing more than the cost of this solution. The following claim shows that the validity property stressed by Moscato in [19] holds. Hence, a memetic algorithm combining such chromosomes is expected to find an optimal NEARP solution.

Claim. The proposed chromosome structure is a *valid* representation.

Proof. By definition, *Split* converts any chromosome into an optimal NEARP solution (subject to the sequence order). Moreover, *there exists at least one optimal chromosome*: consider any optimal NEARP solution and concatenate its trips in any order. □

4.2 Extended OX crossover

Thanks to chromosomes without trip delimiters, classical crossovers for permutation chromosomes can be used for the NEARP. We quickly obtained good results by adapting the classical *Order Crossover* or *OX*, developed by Oliver et al. for the TSP [20]. This chromosome works well for cyclic permutations. Although a NEARP solution is not (strictly speaking) a permutation, it can be viewed as a cyclic list of trips because there is no reason to give a special role to a "first" or "last" trip.

Given two parents P_1 and P_2 of length τ, OX randomly draws two positions i and j with $1 \leq i \leq j \leq \tau$. To build the first child C_1, the substring $P_1(i) \ldots P_1(j)$ is first copied into $C_1(i) \ldots C_1(j)$. The tasks $P_2(j+1) \ldots P2(\tau)$ and $P_2(1) \ldots P_2(i-1)$ are then examined in that order. The tasks which are not yet present in C_1 are used to fill the empty slots of C_1, in the order $C_1(j+1) \ldots C_1(\tau), C_1(1) \ldots C_1(i-1)$.

```
Rank:  1   2   3   4   5   6   7   8   9
                  i=4     j=6
                   ↓       ↓
P1  :  1   3   2 | 6   4   5 | 9   7   8
P2  :  3   7   8 | 1   4   9 | 2   5   6

C1  :  8   1   9   6   4   5   2   3   7
C2  :  2   6   5   1   4   9   7   8   3
```

Fig. 4. Example of OX crossover

This process is illustrated by Figure 4. The other child C_2 is obtained in a similar way, by inverting the roles of P_1 and P_2. For the NEARP, the classical

crossover must be adapted to take edge directions into account, i.e. a task u may be copied from P_1 to C_1 only if both u and $inv(u)$ are not already in the child. The extended crossover can be implemented in $O(\tau)$.

4.3 Local search procedure

To get a memetic algorithm, a *local search procedure* (*LSP*) replaces the mutation operator traditionally applied to new solutions created by recombination (children) after a crossover. Since LSP cannot work on chromosomes (without trip delimiters), the input chromosome S must be converted first into a NEARP solution, using the *Split* procedure of 3.3 LSP performs successive phases that scan in $O(\tau^2)$ the following types of moves, depicted in figures 5 and 6.

- Flip one task a, i.e., replace a by $inv(a)$ in its trip,
- Move one task a after another task or after the depot,
- Move two consecutive tasks a and b after another task or after the depot,
- Swap two tasks a and b,
- 2-opt moves depicted in figure 6.

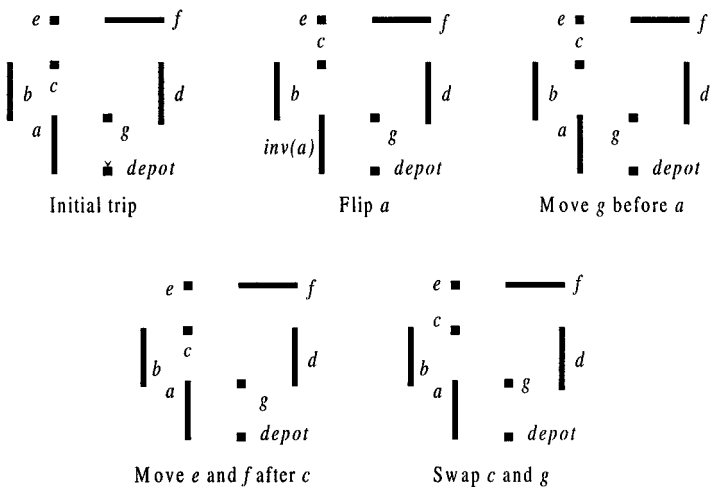


Fig. 5. Simple moves in the Local Search Procedure.

All these moves can be applied to one or two trips. Moreover, each task a moved to another location or swapped with another task may be inserted as a or $inv(a)$. For instance, the third move (move two tasks a and b) comprises in fact four distinct sub-cases: insert a and b, $inv(a)$ and b, a and $inv(b)$, or $inv(a)$ and $inv(b)$.

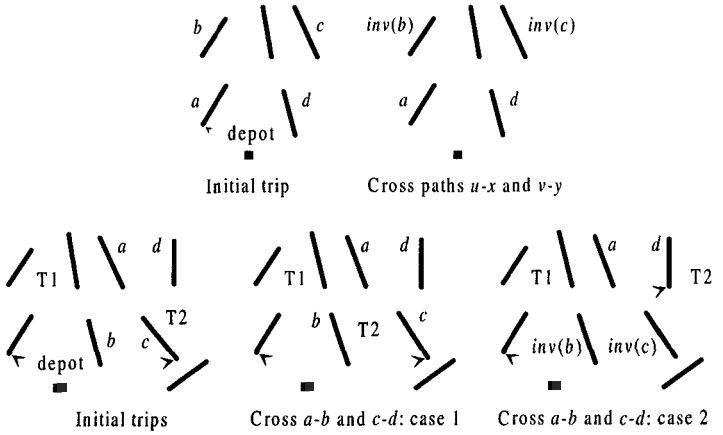


Fig. 6. 2-OPT moves on one trip and on two trips.

Each phase ends by performing the first improving move detected or when all moves have been examined. The loop on phases stops when a phase reports no improvement. The resulting NEARP solution is converted back into a chromosome by concatenating the tasks of its trips. In all cases, LSP terminates by applying *Split* to the result, because this sometimes decreases a bit the total cost.

On big instances, the neighborhood cardinality $O(\tau^2)$ leads to very time-consuming local searches, that typically absorb 95% of the total MA running time. To remedy this drawback, a classical neighbourhood reduction technique is used. We define for each task a list $neib(a)$ that contains the τ tasks sorted in increasing order of distance to a and a threshold *thresh* between 1 and τ. Then, each iteration of the local search is restricted to all pairs (a, b), such that b belongs to the *thresh* first tasks in $neib(a)$.

4.4 Population structure and initialization

The population is stored in an array Π of nc chromosomes, kept sorted in increasing order of costs (computed by *Split*). So, the best solution corresponds to Π_1. Identical solutions (*clones*) are forbidden to prevent a premature convergence of the MA (amplified by the local search) and to favour a better dispersal of solutions. Instead of an exact clone detection (e.g., using hashing methods), we adopt a simpler system in which the costs of any two solutions S_1, S_2 must be spaced at least by a constant $\Delta > 0$, i.e., $|F(S1) - F(S2)| \leq \Delta$. This condition is called the Δ-property. Its simplest form for integer costs is $\Delta = 1$, ensuring solutions with distinct costs.

At the beginning, the heuristics NNH, MH and TSH described in section 3 are executed. The local search procedure LSP of 4.3 is applied to the solutions computed by NNH and TSH, and after each merger for the Merge Heuristic MH. The resulting solutions are converted into chromosomes by concatenating their trips and stored in Π. The population is then completed by random chromosomes. On very small problems, it may be difficult to satisfy the Δ-property, especially if nc is large. In practice, we try up to mnt times to draw a random Π_k such that the Δ-property holds for $\Pi_1 \ldots \Pi_k$. In case of failure, the number of chromosomes nc is truncated to $k-1$.

Large populations raise another problem. During the MA, some crossovers are unproductive because their children violate the Δ-property and cannot be kept. The percentage of unproductive crossovers quickly increases with nc and with the local search rate. It is tolerable (less than 5%) if the population is relatively small (30-40 chromosomes) and if less than 20% of children undergo the local search.

Compared to the MA template proposed by Moscato [19], note that the local search is applied to the three initial heuristic solutions, but not to the random ones: because of the small population size, we are obliged to do so to have a sufficient dispersal of initial solutions and a better exploration of the solution space.

4.5 Basic iteration and stopping criteria

Each iteration of the MA starts by selecting two parents P_1 and P_2 by binary tournament: two chromosomes are randomly selected and the best one becomes P_1, this process is repeated to get P_2. The extended OX crossover (4.2) is applied to generate two children C_1 and C_2. One child C is selected at random, evaluated by *Split*, and improved by local search (4.3) with a fixed probability *pls*. An existing chromosome Π_k is drawn above the median cost ($k \geq nc/2$) to be replaced by C. The replacement is performed only iff the Δ-property holds for $(\Pi \setminus \{\Pi_k\}) \cup \{C\}$.

The MA stops after a maximum number of iterations *mni*, after a maximum number of crossovers without improving the best solution (Π_1) *mniwi*, or when a lower bound *LB* known for some instances is achieved.

4.6 Overall MA structure

The overall MA structure is given by Algorithm 1. The parameters are the population size *nc*, the minimal cost spacing Δ between any two solutions, the maximum number of tries *mnt* to get each initial random chromosome, the local search rate *pls*, the maximum number of iterations (crossovers) *mni*, the maximum number of iterations without improving the best solution *mnwi* and the lower bound *LB*.

```
Memetic Algorithm:
Begin
  run heuristics NNH, MH, TSH and improve solutions with LSP;
  discard solutions violating the Δ-property;
  convert the remaining solutions into chromosomes, by concatenating their trips;
  Π ← {resulting chromosomes};
  complete Π with random chromosomes satisfying the Δ-property;
  sort Π in increasing cost order;
  ni, niwi ← 0;
  Repeat Until ( (ni = mni) or (niwi = mniwi) or (F(Π1) = LB) ) Do
    ni ← ni + 1;
    select two parents P1 and P2 by binary tournament;
    apply OX to P1, P2 and choose one child C at random;
    evaluate C with Split;
    If (random < pls) Then
      improve C with the local search procedure LSP;
    endIf
    draw k at random between ⌊nc/2⌋ and nc included;
    If (Π \ {Πk} ∪ {C} satisfies the Δ-property) Then
      Πk ← C;
      If (F(C) < F(Π1)) Then
        niwi ← 0;
        Else
        niwi ← niwi + 1;
      endIf
      shift Πk to keep Π sorted;
    endIf
  endDo
End.
```

Fig. 7. Overall MA structure

5 Preliminary testing on VRP and CARP instances

5.1 Implementation and instances

The heuristics and the memetic algorithm have been programmed in the Pascal-like language Delphi version 5 and tested on a 1 GHz Pentium III PC with Windows 98. Before running the MA on NEARP instances, for which no published algorithm is available for comparison, we decided to test it on standard VRP and CARP instances.

The selected set of CARP instances (*gdb* files) contains 25 undirected problems built by DeArmon [16] and used by almost all algorithms published for the CARP. They can be downloaded on the Internet [21]. Instances 8 and 9 are discarded by all authors because they contain inconsistencies. The other

files contain 7 to 27 nodes and 11 to 55 edges. All data are integers and all edges are required.

An excellent lower bound [9] is available for all these instances. The optimum is known for 21 instances out of 23, thanks to the tabu search CARPET of Hertz et al. [13] and the genetic algorithm of Lacomme et al. [15]. The only two remaining open instances are *gdb10* and *gdb14*. In spite of their relatively small size, the *gdb* instances are not so easy: for example, no constructive heuristic is able to solve more than two problems to optimality.

The set of VRP instances contains 14 Euclidean problems proposed by Christofides et al. [22]. They can be downloaded for instance from the OR Library [23]. They have 50 to 199 nodes. The network is complete and the costs are real numbers corresponding to the Euclidean distances between nodes. Files 6 to 10, 13 and 14 contain a route-length restriction. This constraint is easily handled by the MA, by ignoring the too long trips in the auxiliary graph built by the chromosome evaluation procedure *Split* (see 3.3).

The best-known solution costs to Christofides instances have been computed by various tabu search algorithms (TS) and simulated annealing procedures. They can be found for example in Gendreau et al. [3] and in Golden et al. [4]. As underlined by these authors, double-precision computations must be used to avoid cumulating rounding errors and to guarantee meaningful comparisons between final solution costs. No tight lower bound is available, but the best exact methods have proved that the solution values found for files 1 and 12 are in fact optimal.

5.2 Results for CARP instances

The MA parameters used for the *gdb* instances are $nc = 30$, $\Delta = 1$, $mnt = 60$, $pls = 0.1$, $mni = 20000$ and $mniwi = 6000$. Since these instances are not too large, the local search is set to a full aperture, i.e., $thresh = \tau$ (see 4.3).

Table 1 gathers the results for the CARP. The columns show, from left to right, the file name, the number of nodes n, the number of links m (equal to τ, since all edges are required), the best known solution value (*BKS*), the results obtained by the heuristics NNH, MH and TSH (followed by one call to the local search) and by the MA. The same setting of parameters is applied to all instances, except in the last column *Best MA* that reports the best solutions found with various settings during our experiments. The CPU time is given in seconds for all algorithms. The two last rows give the average deviation to the lower bound in % and the number of best solutions retrieved.

The MA solves 17 out of 23 instances to optimality, within reasonable CPU times (42 seconds on average, max. 4 minutes). The average deviation to the bound is quite small: 0.43%. To compare with, the best tabu search published [13] finds 18 optima, but with a slightly greater deviation of 0.48%. Using various settings, only two instances are improved (*gdb11* and *gdb24*).

5.3 Results for VRP instances

Table 2 reports the results found for the VRP in nearly the same format as table 1. However, the numbers of edges are here omitted because the networks are complete and, due to the lack of good lower bounds, the *Average* row now gives the average deviation to best-known solutions.

The MA parameters used this time are $nc = 30$, $\Delta = 0.5$, $mnt = 60$, $pls = 0.5$, $mni = 20000$ and $mniwi = 6000$. Neighborhood aperture is reduced to $thresh = 2 \times max\{10, \tau^{0.5}\}$. After the first phase with up to 20000 crossovers, the MA performs four short restarts of 2500 crossovers, in which the 7 worst chromosomes are replaced by random ones.

The MA finds 3 best-known solutions and the average deviation to best solutions is very small: 0.39%. The CPU time (10 minutes on average) exceeds 30 min only for one of the two largest instances with 199 nodes (*vrpnc10*, 42 min). In [4], Golden et al. list the results obtained by the 10 best TS methods for the VRP, which find 3 to 12 best-known solutions. Using several settings of parameters (*Best MA* column), the MA would be at rank 3 in this comparison, after three TS methods that respectively retrieve 12, 10 and 8 best-known solutions.

In a preliminary version of the MA, there was no neighborhood reduction technique in the local search and no restart. The average solution cost was only a bit larger, but the CPU time was excessive, exceeding 1 hour for four instances and reaching 2 hours and 53 minutes on *vrpnc10*.

The possibility of using a tabu search step for diversification has not been used for three reasons. Firstly, tabu search competitors are already available for the CARP and the VRP, so we wanted to develop in contrast a "pure" evolutionary algorithm. Secondly, a sufficient diversification seems to be provided by the restarts. Thirdly, we do not strictly follow Moscato's template and two features favour a good dispersal of solutions in the search space: a) the local search is not systematic and b) the population contains at any step distinct solutions.

6 Random generator of NEARP instances

A random generator has been designed to build NEARP instances. These networks are mixed, planar, strongly connected and imitate the shape of real street networks. The generation starts with a rectangle of basic squares. At the beginning, only the nodes at the corners of the basic squares exist, see a) in figure 8.

Four modifications can be applied to each square (see b) in figure 8): split vertically (V), horizontally (H), along the 1st diagonal (D1) and along the 2nd one (D2). Note that V and H create two new nodes and that a square may undergo up to four modifications. As from two, a central node is created to preserve planarity. We obtain in that way a planar undirected graph (see c)

in figure 8). Since this graph is too regular, each node is randomly moved in a small circle. At this stage, provisional lengths in meters can be computed from node coordinates. To simulate streets that are a bit curved (without drawing them), a second perturbation consists of applying a random growth factor to each length (between 0 and 10% for instance).

Each edge is then converted into a one-way street with a given probability, by suppressing at random one of the two internal arcs that code the edge. Of course, strong connectivity is preserved. Traversal costs are computed from the length of each link, assuming an average deadheading speed of vehicles. Then, we decide for each link if its is required. If yes, we draw a non-zero demand at random.

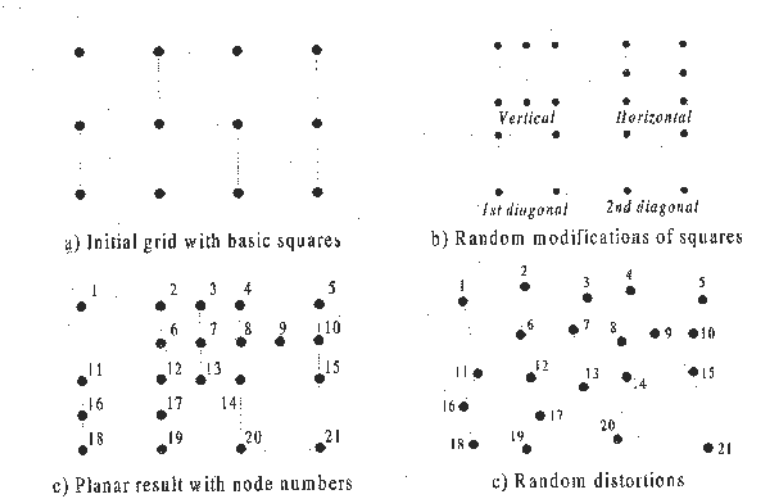


Fig. 8. Principles of random generation.

Finally, we decide for each 2-way street if it must be considered as one edge. If yes, the two arcs are linked with the *inv* pointer (see subsection 2.2) and the edge demand is the sum of quantities of the two sides. The processing cost of each task is computed as a function of its length, its demand, and a given vehicle processing speed. The instance generation ends by drawing vehicle capacity and depot location.

7 Selected set of NEARP instances with MA solutions

The generator has been used to build 23 large scale NEARP instances listed in table 3, with $n = 11 - 150$ nodes, $m = |A| + 2|E| = 29 - 311$ internal arcs and $\tau = 20 - 212$ tasks. The tasks comprise $\nu = 3 - 93$ node-tasks, $\epsilon = 0 - 94$

edge-tasks, and $\alpha = 0 - 149$ arc-tasks. All these files can be requested by an e-mail sent to the authors.

These networks are comparable in size to the ones observed in waste collection applications. Of course, the whole network of a big town can be much larger, but the collecting process is divided into sectors in practice. This defines an independent NEARP in each sector, with typically 100-200 street segments.

The MA parameters already applied to the VRP (including the restarts) are used, except $\Delta = 1$ instead of $\Delta = 0.5$, because all costs are integers in our NEARP instances. The results of the MA are listed in table 3. The average running time is 8 minutes of CPU time (max. 23 minutes). No published algorithm can be used for comparison and no good lower bound is available. This is why the table reports average deviations to the best MA solutions obtained by using various sets of parameters. However, by extrapolating the very good results achieved on the CARP and on the VRP, we think that the solutions values computed for the NEARP are quite good and other researchers are invited to try to obtain better results.

8 Conclusion

This paper presents a new problem, the NEARP, that generalizes the VRP and the CARP, and a memetic algorithm to solve it. Computational testing on standard VRP and CARP instances show that the MA can compete with the best metaheuristics published for these particular cases of the NEARP. Using a dedicated random network generator, we have built a set of 23 NEARP instances to evaluate the MA in the general case. The results are promising but the time spent in the local search procedure seems affected by the number of required nodes and should be improved by using more efficient neighborhoods for the instances with a majority of node-tasks. Beyond these interesting results, the main interest of this research is to solve several classical routing problems with one single algorithm.

References

1. Toth P, Vigo D (1998) Exact solution of the Vehicle Routing Problem. In: Crainic TG, Laporte G (eds) Fleet management and logistics, 1–31. Kluwer, Boston.
2. Laporte G, Gendreau M, Potvin JY, Semet F (2000) Classical and modern heuristics for the Vehicle Routing Problem. International Transactions in Operational Research 7:285–300.
3. Gendreau M, Laporte G, Potvin JY (1998) Metaheuristics for the Vehicle Routing problem. GERAD research report G-98-52, Montréal, Canada.

4. Golden BL, Wasil EA, Kelly JP, Chao IM (1998) The impact of metaheuristics on solving the Vehicle Routing Problem: algorithms, problem sets, and computational results. In: Crainic TG, Laporte G (eds) Fleet management and logistics, 33–56. Kluwer, Boston.
5. Toth P, Vigo D. The granular tabu search and its application to the Vehicle Routing Problem. To appear in INFORMS Journal on Computing.
6. Assad AA, Golden BL (1995) Arc routing methods and applications. In: Ball MO et al. (eds) Handbooks in OR and MS, volume 8, 375–483. Elsevier.
7. Pearn WL, Assad A, Golden BL (1987) Transforming arc routing into node routing problems. Computers and Operations Research 14:285–288.
8. Hirabayashi R, Saruwatari Y, Nishida N (1992) Tour construction algorithm for the Capacitated Arc Routing Problem. Asia-Pacific Journal of Operational Research 9:155–175.
9. Belenguer JM, Benavent E (2003) A cutting plane algorithm for the Capacitated Arc Routing Problem. Computers and Operations Research 30(5):705–728.
10. Golden BL, DeArmon JS, Baker EK (1983) Computational experiments with algorithms for a class of routing problems. Computers and Operations Research 10:47–59.
11. Golden BL, Wong RT (1981) Capacitated arc routing problems, Networks 11:305–315.
12. Ulusoy G (1985) The fleet size and mix problem for capacitated arc routing. European Journal of Operational Research 22:329–337.
13. Hertz A, Laporte G, Mittaz M (2000) A tabu search Heuristic for the Capacitated Arc Routing Problem. Operations Research 48:129–135.
14. Lacomme P, Prins C, Ramdane-Chérif W (2001) A genetic algorithm for the Capacitated Arc Routing Problem and its extensions. In: Boers EJW et al. (eds) Applications of evolutionnary computing. Lecture Notes in Computer Science 2037, 473–483. Springer, Berlin.
15. Lacomme P, Prins C, Ramdane-Chérif W. Competitive memetic algorithms for arc routing problems. To appear in Annals of Operations Research.
16. DeArmon JS (1981) A comparison of heuristics for the Capacitated Chinese Postman Problem. Master's thesis, The University of Maryland at College Park, MD, USA.
17. Orloff CS (1974) A fundamental problem in vehicle routing. Networks 4:35–64.
18. Cormen TH, Leiserson CL, Rivest ML, Stein C (2001) Introduction to algorithms, 2nd edition. The MIT Press, Cambridge, MA.
19. Moscato P (1999) Memetic algorithms: a short introduction. In: Corne D, Dorigo M and Glover F. (eds) New ideas in optimization, 219–234. McGraw-Hill.
20. Oliver IM, Smith DJ, Holland JRC (1987) A study of permutation crossover operators on the traveling salesman problem. In: Grefenstette JJ (ed) Proceedings of the 2nd Int. Conf. on Genetic Algorithms, 224–230. Lawrence Erlbaum, Hillsdale, NJ.
21. Belenguer JM, Benavent E. Directory with 3 sets of CARP instances. Web site: `http://www.uv.es/~belengue/carp.html`.
22. Christofides N, Mingozzi A, Toth P (1979) The Vehicle Routing Problem. In: Christofides N, Mingozzi A, Toth P, Sandi C (eds) Combinatorial optimization, 315–338. Wiley.

23. Beasley JE. Set of VRP instances from the OR library. Web site: http://mscmga.ms.ic.ac.uk/jeb/orlib/vrpinfo.html.
24. Crescenzi P, Kann V. A compendium of NP optimization problem. Web site: http://www.nada.kth.se/~viggo/wwwcompendium/node103.html.

Appendix

The problems studied in this paper can be described in the style used by Crescenzi and Kann [24] for their compendium of NP optimisation problems. The most general problem is the NEARP. Its highest-level particular cases are the CARP and the VRP. These three problems are not listed in the compendium.

Node, Edge and Arc routing problem (NEARP)

- INSTANCE: Mixed graph $G = (V, E, A)$, initial vertex $s \in V$, vehicle capacity $W \in IN$, subset $V_R \subseteq V$, subset $E_R \subseteq E$, subset $A_R \subseteq A$, traversal cost $c(u) \in IN$ for each "entity" $u \in V \cup E \cup A$, demand $q(u) \in IN$ and processing cost $p(u) \in IN$ for each required entity (task) $u \in V_R \cup E_R \cup A_R$.
- SOLUTION: A set of cycles (trips), each containing the initial vertex s, that may traverse each entity any number of times but process each task exactly once. The total demand processed by any trip cannot exceed W.
- MEASURE: The total cost of the trips, to be minimized. The cost of a trip comprises the processing costs of its serviced tasks and the traversal costs of the entities used for connecting these tasks.

Vehicle Routing Problem (VRP)

- INSTANCE: Complete undirected graph $G = (V, E)$, initial vertex $s \in V$, vehicle capacity $W \in IN$,length $c(e) \in IN$ for each $e \in E$, demand $q(i) \in IN$ for each $i \in V$.
- SOLUTION: A set of cycles (trips), each containing the initial vertex s, that collectively traverses every node at least once. A node must be serviced by one single trip and the total demand processed by any trip cannot exceed W.
- MEASURE: The total cost of the trips, to be minimized. The cost of a trip is the sum of its traversed edges.

Capacitated Arc Routing Problem (CARP)

- INSTANCE: Undirected graph $G = (V, E)$, initial vertex $s \in V$, vehicle capacity $W \in IN$, subset $E_R \subseteq E$, length $c(e) \in IN$ and demand $q(e) \in IN$ for each edge $e \in R$.

- SOLUTION: A set of cycles (trips), each containing the initial vertex s, that collectively traverse each edge of E_R at least once. Each edge of E_R must be serviced by one single trip and the total demand processed by any trip cannot exceed W.
- MEASURE: The total cost of the trips, to be minimized. The cost of a trip comprises the costs of its traversed edges, serviced or not.

However, the following special cases can be found in the compendium:

- the *minimum travelling salesperson*, an uncapacitated version of the VRP,
- the *minimum Chinese postman for mixed graphs* (an uncapacitated version of the CARP, but with a mixed network instead of an undirected one,
- the *minimum general routing problem*, which is an uncapacitated and undirected particular case of the NEARP.

Table 1. Computational results for CARP instances (see 5.2)

File	n	m	LB	BKS	NNH+LS	MH+LS	TSH+LS	MA	Time	BestMA
1	12	22	316	316*	345	323	316*	316*	< 0.01	316*
2	12	26	339	339*	345	359	352	339*	15.87	339*
3	12	22	275	275*	285	296	283	275*	1.10	275*
4	11	19	287	287*	287*	315	322	287*	< 0.01	287*
5	13	26	377	377*	395	395	383	377*	16.47	377*
6	12	22	298	298*	313	319	315	298*	1.81	298*
7	12	22	325	325*	346	325*	333	325*	< 0.01	325*
10	27	46	344	348	387	372	389	350	198.77	350
11	27	51	303	303*	339	329	346	311	115.12	309
12	12	25	275	275*	285	285	301	275*	1.21	275*
13	22	45	395	395*	430	427	412	395*	48.72	395*
14	13	23	450	458	498	474	520	458=	16.09	458=
15	10	28	536	536*	556	548	548	544	28.78	544
16	7	21	100	100*	100*	106	106	100*	< 0.01	100*
17	7	21	58	58*	58*	60	58*	58*	< 0.01	58*
18	8	28	127	127*	129	135	133	127*	8.24	127*
19	8	28	91	91*	91*	93	91*	91*	< 0.01	91*
20	9	36	164	164*	167	182	174	164*	0.88	164*
21	8	11	55	55*	55*	61	63	55*	< 0.01	55*
22	11	22	121	121*	123	125	125	121*	37.51	121*
23	11	33	156	156*	158	162	160	156*	7.36	156*
24	11	44	200	200*	205	205	207	202	235.96	201
25	11	55	233	233*	235	239	239	235	229.31	235
Average				0.13%	4.01%	5.47%	5.71%	0.43%	41.88s	0.36%
BKS retrieved					5	1	3	18		18

Times in seconds on a 1 GHz PC. Average deviations to LB in %.

Asterisks denote proven optima, '=' best-known solutions retrieved

Table 2. Computational results for VRP instances (see comments in 5.3)

File	n	BKS	NNH+LS	MH+LS	TSH+LS	MA	Time	BestMA
1	50	524.61*	578.84	548.58	566.70	524.93	62.40	524.61=
2	75	835.26	910.74	896.30	906.01	836.81	149.45	835.26=
3	100	826.14	863.23	872.52	867.40	829.72	300.71	827.39
4	150	1028.42	1138.94	1127.93	1062.56	1032.82	788.89	1032.82
5	199	1291.45	1356.11	1377.05	1371.20	1303.09	1698.62	1303.09
6	50	555.43	611.79	618.39	567.98	555.43=	112.32	555.43=
7	75	909.68	942.82	967.52	998.77	912.89	171.75	909.68=
8	100	865.94	924.65	910.23	898.75	866.87	320.22	865.94=
9	150	1162.55	1255.80	1280.49	1263.57	1169.69	1121.47	1169.69
10	199	1395.85	1530.77	1512.90	1524.74	1417.57	2539.26	1409.76
11	120	1042.11	1112.95	1052.81	1059.67	1045.55	342.41	1042.11=
12	100	819.56*	825.38	820.92	828.75	819.56*	1.70	819.56=
13	120	1541.14	1600.49	1573.23	1575.03	1548.58	924.61	1546.78
14	100	866.37	958.70	868.62	893.24	866.37=	284.18	866.37=
Average			7.13%	5.62%	5.20%	0.39%	629.86s	0.25%
BKS retrieved			0	0	0	3		8

Times in s on a 1 GHz PC. Average deviations to BKS in %.

Asterisks denote proven optima, '=' best-known solution retrieved.

Table 3. Computational results for the new NEARP instances (see section 7)

File	n	m	τ	ν	ϵ	α	NNH+LS	MH+LS	TSH+LS	MA	Time	BestMA
1	21	66	48	11	0	37	2853	2972	2878	2632	108.31	2589
2	68	246	185	36	0	149	13275	13195	13371	12336	1078.46	12241
3	31	96	79	16	8	55	4223	4164	4158	3702	157.04	3669
4	53	186	98	10	75	13	8121	8395	8436	7583	548.10	7583
5	32	65	65	23	4	38	5160	5048	5144	4562	100.02	4544
6	49	100	108	40	4	64	7760	7515	7826	7087	204.49	7087
7	75	166	168	54	8	106	10585	10536	10572	9974	662.62	9748
8	77	174	177	63	6	108	11685	11695	12112	10714	767.64	10658
9	29	86	50	6	39	5	4274	4274	4272	4041	140.83	4038
10	56	204	107	4	94	9	8190	8312	8248	7755	843.17	7582
11	69	241	82	65	6	11	5057	4716	4907	4503	414.68	4494
12	38	71	53	1	0	52	3429	3308	3532	3235	71.30	3235
13	150	294	141	79	2	60	10530	9756	10018	9339	550.57	9110
14	94	332	93	93	0	0	9296	8834	9106	8615	357.24	8566
15	52	182	91	0	91	0	9154	8785	9066	8359	390.19	8340
16	71	138	169	36	0	133	10205	9847	10498	9389	536.13	8933
17	42	134	63	16	16	31	4555	4775	4617	4165	116.12	4037
18	117	212	127	39	0	88	8001	8311	8087	7411	475.65	7254
19	126	311	212	61	9	142	17877	17605	18166	17036	1273.39	16554
20	43	133	73	38	2	33	5555	5127	5405	4918	164.56	4844
21	60	206	180	55	68	57	20023	19883	19702	18509	1370.61	18201
22	25	68	42	7	10	25	2187	2321	2222	1941	65.75	1941
23	11	29	20	3	2	15	784	780	820	780	20.38	780
Average							10.07%	8.87%	10.49%	1.26%	452.92s	0%
Solns of Best MA retrieved							0	1	0	5		23

Times in seconds on a 1 GHz PC. Average deviations to best MA taken as reference, in %.

Using Memetic Algorithms for Optimal Calibration of Automotive Internal Combustion Engines

Kosmas Knödler, Jan Poland, Peter Merz, and Andreas Zell

University of Tübingen, Computer Science Department
Sand 1, D-72076 Tübingen, Germany
knoedler@informatik.uni-tuebingen.de

Summary. Many combinatorial optimization problems occur in the calibration of modern automotive combustion engines . In this contribution, simple hill–climbing algorithms (HCs) for three special problems are incorporated in Memetic Algorithms (MAs) using specific crossover and mutation operators. First, the k-exchange algorithm as a well known technique of D-optimal design of experiments (DOE) is improved. Second, a (near-)optimum test bed measurement scheduling (TBS) as a variant of the traveling salesman problem (TSP) is calculated, and third, the final design of look-up tables (LTD) for electronic control units is optimized. It is shown that in all cases MAs that work on locally optimal solutions calculated by the corresponding HCs significantly improve former results using Genetic Algorithms (GAs). The algorithms have been successfully applied at BMW Group Munich.

1 Introduction

Nowadays, a vastly increasing number of technical functions satisfy the customer demands for optimal performance of automotive combustion engines. Moreover, they provide the only way to fulfill the legal rules for fuel consumption and exhaust emissions. On the other side, the calibration of the corresponding control functions running within the micro controller of an electronic control unit becomes more and more sophisticated. Many tasks that have been tackled manually for former generations of combustion engines need to be solved in a new way to guarantee optimality of control strategies. For clarity reasons, the engine's parameter space is considered to be spanned by five main parameters, the *engine speed*, the *air mass flow*, the *inlet valve spread* and the *exhaust valve spread*, and the *ignition timing angle*. Here, the engine speed and the air mass flow define the operating point of an engine. Most other parameters, particularly the named parameters, correspond to special engine functions that are controlled depending on the current operating point. For every control function there is a look-up table stored within the electronic

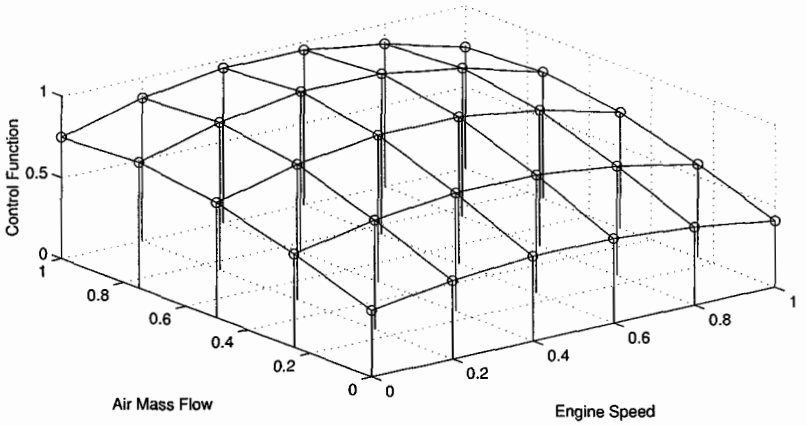


Fig. 1. A lookup table approximates a continuous control function depending on the operating points, i.e. the engine speed and the air mass flow. For the current operating point, the micro controller within the electronic control unit calculates the optimal value of the control function by bi-linear interpolation.

control unit. The micro controller uses the current operating point of the engine to calculate the new settings for the valve spreads and the ignition timing angle by bi-linear interpolation. Figure 1 shows an example look-up table for a general control function. At every operating point there is a unique value for the control function.

Here, three combinatorial optimization problems are considered. To see where these problems are located within the calibration process for automotive combustion engines, figure 2 shows the main steps in the workflow. The three blocks with gray background label the three combinatorial problems.

At the beginning there is the design of experiments (DOE). Consider here the example of a full factorial grid in the above mentioned parameter space with 10 units in each dimension. This would lead to a total number of 10^5 measurement points. To record all engine characteristics at one specific measurement point up to three minutes are necessary, which is very expensive. The idea of DOE is therefore, to generate a reduced list of measuring points, that contains only those measurement candidates that lead to the best regression model of predefined order. Section 3 will give further details on this topic.

Although DOE can reduce the number of measurements significantly, the measuring process at test beds is still a significant factor in the calibration process. It is important to note that the change of parameter settings at test beds can lead to strong oscillations of the engine system and hence to a long relaxation time after which the recording of measuring channels can start. Especially the variations of the engine speed and of the air mass flow can cost a lot. Therefore an optimal test bed schedule (TBS), i.e. an optimal

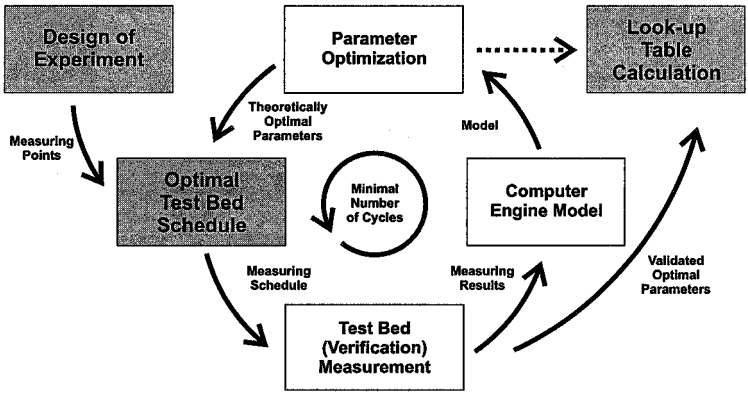


Fig. 2. A workflow for the calibration of modern automotive combustion engines. The combinatorial problems D-optimal design of experiments, optimal test bed schedules, and final design of look-up tables are tackled by Memetic Algorithms.

measuring order is calculated to reduce these oscillations. This problem is a higher dimensional variant of the traveling salesman problem (TSP) considering Hamiltonian paths instead of cycles and using a non-standard metric. It will be discussed in section 4.

The next three steps are discussed together. First, the results of the measurement process at test beds are primarily the fuel consumption and exhaust emissions at the different points within the parameter space. Second, by means of these results computer engine models for the global functions for the fuel consumption and for the exhaust emissions are calculated. Here, artificial neural networks and polynomial regression models are used. Third, the generated models are used to determine the parameter combinations at every operating point that lead to the optimum fuel consumption and exhaust emissions. As optimization algorithms, both classical optimization and Evolutionary Algorithms are used. In order to decide whether the models are accurate enough to represent the real engine behavior, verification measurements need to be performed at the test bed. The previous steps might be repeated until the results are satisfying.

Another combinatorial optimization problem occurs in the final step of the workflow, i.e. a final look-up table design (LTD). After the modeling and the optimization process, there might be more than one parameter combination available at each operating point. One reason for this situation is the use of different model types with most probably different landscapes and therefore different optimum points. Second, one model usually provides multiple local optima. All the candidates lead to acceptably good values for both the fuel consumption and the exhaust emissions. Hence, it is not clear a priori, which candidate should be chosen at each operating point in order to define the unique look-up tables for the electronic control unit. First, consider only one

look-up table, e.g. the one in figure 1. At each grid point there would be a set of candidates with quite different function values. Now, a criterion for the usefulness of one candidate is needed. In order to provide optimal behavior of the engine, the global smoothness of the map defined by the look-up table is considered. Especially mechanically adjusted actuators need smooth maps to guarantee easy transitions between parameter settings. In this case, it is "only" necessary to select those candidates simultaneously, which define the smoothest map. For this case it could be shown in [1], that the corresponding selection problem is NP-hard. Within the application process normally several look-up tables need to be considered simultaneously, which constitutes a multi-objective combinatorial NP-hard optimization problem: Consider the situation, that one candidate has already been selected at every operating point. To perform a new candidate choice at a specific operating point within the next iteration, you are forced to select all components of an optimum parameter vector. I.e., the new choice might improve the smoothness of the inlet valve spread look-up table, but on the other side it worsens the smoothness of the exhaust valve spread and of the ignition timing angle look-up tables.

In section 2.1, the general framework of the Memetic Algorithms (MAs, see e.g. [2] or [3]) and the hill–climbing algorithms (HCs) employed in the studies is presented. In the following sections 3, 4, and 5 the three combinatorial optimization problems introduced above are described in more technical detail. Every section includes results of the MAs for several test instances. For comparison, results of the heuristics and of the pure or hybrid Genetic Algorithms (GAs, see e.g. [4]) are given, too. Here, hybrid GAs use the well-known problem specific heuristics to mutate individuals with typically small mutation rates.

2 Hill–Climbing and Memetic Algorithms

MAs are characterized by the strict application of local search algorithms after the initial generation of individuals, and after each evolutionary operation, i.e. after each crossover and mutation (see [2] for an introduction). The pseudo code for the MAs presented in this paper in figure 3 is taken from [3]. The framework is rather simple, since it does not utilize spatially structured or tree-structured populations. The same framework has been used in the studies of several other combinatorial optimization problems, including the graph-bipartitioning problem ([26]), the quadratic assignment problem ([27]), the Euclidean traveling salesman problem ([28]), and binary quadratic programming ([29]), with great success.

In contrast to other hybrid Evolutionary Algorithms, local search is applied to all newly created individuals after recombination or mutation. The standard MA performs crossover and mutation operation strictly separated from each other (MA1), which is unlike the GAs. Here, also a second non-standard MA following the sequential use of crossover and mutation in GAs is used (MA2).

```
procedure Memetic Algorithm;
begin
    for j := 1 to μ do
        I := generateSolution();
        I := Local-Search(I);
        add individual I to P;
    endfor;
    repeat
        for i := 1 to p_cross · μ do
            select two parents I_a, I_b ∈ P randomly;
            I_c := Recombine(I_a, I_b);
            I_c := Local-Search(I_c);
            add individual I_c to P';
        endfor;
        for i := 1 to p_mut · μ do
            select an individual I ∈ P randomly;
            I_m := Mutate(I);
            I_m := Local-Search(I_m);
            add individual I_m to P';
        endfor;
        P := select(P ∪ P');
        if converged(P) then P := Local-Search(Mutate(P));
    until terminate=true;
end;
```

Fig. 3. The Standard Memetic Algorithm: All individuals in the populations represent local optima. Crossover and mutation are applied independently from each other.

The number of individuals that are chosen from a population of size μ to apply crossover and mutation operations are given by $p_{cross} \cdot \mu$ and $p_{mut} \cdot \mu$, respectively.

If the algorithm has converged, e.g. if there was no change in the fitness value for a specific number of generations, every individual but the best one is mutated. Afterwards, local search is applied to produce a population of local optima. This diversification is a high level mutation operation, working on the whole population, not on single individuals. Therefore, this kind of mutation is called *meta mutation* ([3]) or *cataclysmic mutation* ([9]). Here, the convergence is tested by controlling the number of generations without changes in optimal fitness, and by comparing the individuals in the current population.

D-optimal design of experiments, for the optimal test bed scheduling as TSP variant, and for the optimal final lookup table design are given. Since in all three combinatorial optimization problems considered in this contribution

```
procedure Hill-Climber;
    repeat N times
        Generate neighboring solution s' from s;
        if fitness(s') > fitness(s) then s := s';
    end repeat;
    return s;
```

Fig. 4. A Hill–Climbing Algorithm as example for a possible local search operation in the Memetic Algorithm given in figure 3.

hill–climbing algorithms (HCs) are used as local search, the pseudo code is given in figure 4. These HCs are characterized by the repetition of specific mutations to search in the neighborhood of the current solution for a certain number of times. A new solution of the corresponding problem is accepted, if its fitness is better than the fitness of the former solution. Here, the neighborhood of the current solution is searched in random order and the number of iterations is limited to N. A local optimum with respect to the neighborhood may not be reached after N iterations. Therefore, additional applications of the hill–climber may result in improved solutions. More systematic and therefore more sophisticated local search heuristics might bring further benefit.

In the following sections after the description of each problem, results of the MAs compared to the conventional and hybrid GAs are presented. Hybrid GA means the use of local search algorithms for mutation with (typical for GAs) small mutation probability. Hybrid GAs are used for the two problems, D-optimal design of experiment and for the optimal test bed scheduling. All the figures for result presentation use mean fitness values from 20 runs of the algorithms. Since minimization problems are considered here, smaller values within the plot mean better results. The error bars display the range between the minimum and the maximum result achieved by the corresponding algorithm.

3 D-optimal Design of Experiments

Given are n candidates $x_1, \ldots, x_n$ defined by n points $u_1, \ldots, u_n$ in the engine's parameter space and a regression type, e.g. a polynomial model for the fuel consumption or the exhaust emission. For the 2-dimensional case the $j = 1 \ldots p$ candidates $x_j = (1, u_{1j}, u_{2j}, u_{1j}^2, u_{2j}^2, u_{1j}u_{2j})^T$ define a 2-nd order model for the points $(u_{1j}, u_{2j})_{j=1}^p$. By the choice of $p < n$ candidates indicated by $\xi = (j_1, \ldots, j_p) \in \{1 \ldots p\}^p$, the design matrix is defined by

$$X_\xi = (x_{j_1} \ldots x_{j_p})^T,$$

i.e. X_ξ is the matrix formed by the chosen candidates. Consider a model

$$y = X_\xi \beta + \epsilon,$$

which is linear in the coefficients. The random vector ϵ is normally distributed with $N(0, \sigma^2 \cdot Id)$, i.e. a normal distribution with mean value zero and variance $\sigma^2 \cdot Id$. The observation vector y has size p. A least square estimate,

$$\hat{\beta} = (X_\xi^T X_\xi)^{-1} X_\xi^T y,$$

is optimized by its minimal covariance matrix $(X_\xi^T, X_\xi)^{-1}\sigma^2$ using appropriate candidates $j_1, \ldots, j_p$. Of course, there are alternative minimum criteria for a matrix. Here, the D-optimality criterion is considered, that is characterized by a minimized $\det((X_\xi^T X_\xi)^{-1})$ or equivalently maximized $\det(X_\xi^T X_\xi)$.

Designs with size $|p| > p_0$ may dominate the evolution since they might have larger determinants. To avoid this development, a dynamic fitness function (see e.g. [10] for this topic) should be used. An initial *estimate* d_0 of the optimum obtained by a single run of the heuristic algorithm helps to define a non-stationary fitness function:

$$\boxed{\phi(\xi) = \det\left((X_\xi^T X_\xi)^{-1}\right) + C(t) \cdot 1_{|\xi|>p_0} \cdot (|\xi| - p_0) \cdot d_0.}$$

Larger designs are punished depending on the size difference $|\xi| - p_0$ and the actual generation t. An increasing sigmoid function $C(t)$ is used, that ensures in each generation, that the design ξ minimizing the fitness function will have a size of $|\xi| = p_0$.

For the use of GAs and MAs, an efficient representation of a design individual is a list coding, consisting of the numerically ordered indices of all points contained in a design ([11]). If the desired size of the design p_0 is not too small, the optimal design may contain repetitions, i.e. candidates that occur two or more times. This fact seems not to be very intuitive at a first glance, but it can be illustrated by a simple example. Consider a linear model in one dimension with $n = 10$ equidistant candidate points. Then the best way to choose $p_0 = 4$ candidates is to take the minimum point and the maximum point each twice. This kind of encoding allows the use of standard mutation. For a GA all the lists should have a fixed length, but on the other hand during the process of optimization it is desirable to try and combine designs of different size. Therefore, the lists have the fixed length of $2 \cdot p_0$, and the unused entries are filled with 0. Note that the alphabet used for this representation has size $n+1$. The use of binary coded individuals, where bit position j is set to 1 if candidate j is included in the design, performed worse and does not allow repetitions. In contrast to binary representation, of course the list representation requires a different crossover operator (see figure 5 for an illustration of a crossover operation on designs). More explicitly, the uniform crossover operator on two lists c_1 and c_2 producing c_3 and c_4 reads as follows: Take the smaller of the

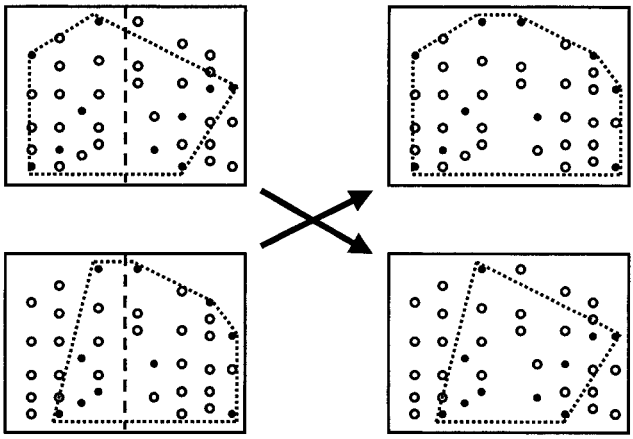


Fig. 5. Illustration of a suitable crossover operation on designs in 2 dimensions. Filled circles indicate points that are included within the design. The upper offspring design inherits the good properties of both parent designs.

first elements of c_1 or c_2 and remove it. With probability 1/2, add it to c_3 or c_4, respectively. If the first elements of c_1 and c_2 are equal, remove them from both c_1 and c_2 and add them to both c_3 and c_4. Repeat these steps until both c_1 and c_2 are empty.

3.1 The k-exchange Algorithm for DOE

The DETMAX and the k-exchange algorithms are common hill–climbing algorithms for the construction of D-optimal DOE. These algorithms are heuristics based on sequentially exchanging *bad* candidates for *better* ones (see [12] for a comparison). Here, only the k-exchange algorithm is used. Depending on the actual size p and the desired final size p_0 of the design ξ, the algorithm adds or removes a candidate x_j, if this leads to a larger or smaller determinant. The new determinant can be expressed in terms of the old one, e.g. for the addition process:

$$\det\left((X_\xi^T\ x_j)\left(X_\xi x_j^T\right)\right) = \det(X_\xi^T X_\xi)\cdot\left(1 + x_j^T(X_\xi^T X_\xi)^{-1}x_j\right).$$

The candidate which maximizes the term $(1 + x_j^T(X_\xi^T X_\xi)^{-1}x_j)$ is added. For the removal process, the candidate x_j that minimizes $1 - x_j^T(X_\xi^T X_\xi)^{-1}x_j$ is chosen. The k-exchange algorithm takes into account, that it might not be optimal to add the candidate for which $1+x_j^T(X_\xi^T X_\xi)^{-1}x_j$ attains its maximum, if afterwards the formula $1 - x_j^T(X_\xi^T X_\xi)^{-1}x_j$ forces the removal of the *wrong* candidate. The addition and the removal form one step, which requires $N \cdot p$ instead of p examinations. To keep this number lower, one can consider only

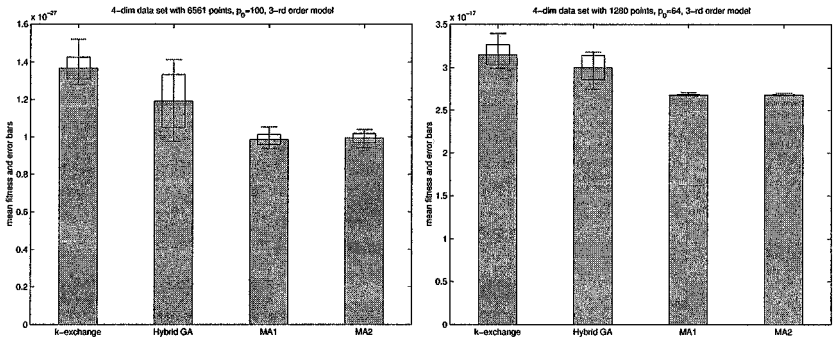


Fig. 6. Comparison of the D-optimal construction of designs for two data sets. k-exchange labels the pure local search algorithm, the hybrid GA uses local search as mutation operation with $p_{mut} = 0.08$. The standard and the non-standard Memetic Algorithms MA1 and MA2 perform comparably well.

the best k candidates for addition and the worst k candidates for removal. Thus, there are k^2 terms of which the maximum is to be found. Since the addition or removal of candidates can be evaluated with the above mentioned efficient update formulas for the determinant, the resulting heuristic does not need to evaluate the whole fitness function and thus becomes very efficient, too.

3.2 Results for the D-optimal Design of Experiment

For the construction of D-optimal DOEs two different data sets are tested. The larger one consists of 6561 points in four dimensions. By means of a third order polynomial model, a design of size 100 candidates is calculated. The left part of figure 6 shows the results of 20 runs. To compare the results of the Memetic Algorithms a hybrid GA is used. It performs local search as mutation operation with typically small mutation rate $p_{mut} = 0.08$. Both Memetic Algorithms find better solutions than the hybrid GA and significantly better than the k-exchange heuristic algorithm. The standard MA (MA1) performs slightly better than the non-standard MA (MA2). The second data set is given by real world application. It consists of 1280 points in four dimensions. The final design size is 64 and a third order polynomial model is used, too. The results are shown in the right part of figure 6.

The parameters for the GAs are: Population size $\mu = 40$, number of offspring $\lambda = 40$, maximum number of generations $t_{max} = 100$, mutation probability $p_{mut} = 0.08$, crossover probability $p_{cross} = 0.6$, tournament selection with 7 individuals, a niching technique with $\nu = 4$ and a niche factor of 0.1. The additional parameters of the MA are the number of individuals for crossover and mutation which are calculated by $p_{cross} \cdot \mu$ with $p_{cross} = 0.5$ and

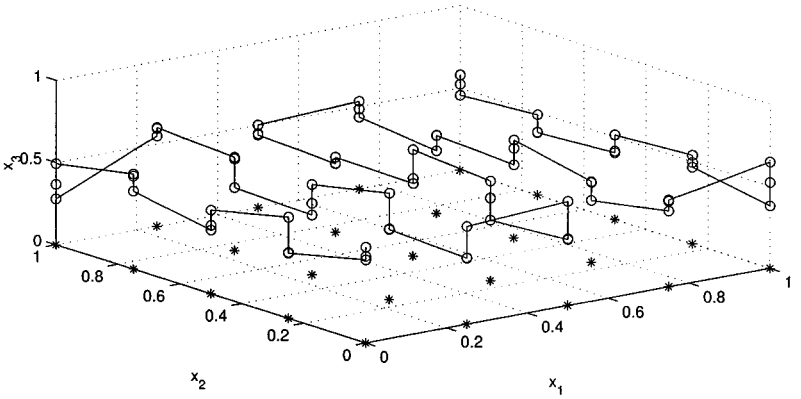


Fig. 7. Visualization of a small data set for optimal test bed scheduling in 3 dimensions. On the path through the parameter space first all x_3 and x_2 variations are measured before the x_1 value is changed.

by $p_{mut} \cdot \mu$ with $p_{mut} = 0.5$ respectively, and a number $N = 1000$ iterations for the hill–climbing.

4 Test Bed Scheduling as TSP Variant

Although DOE improves the situation, concerning the total time requirement of the calibration process, the measuring process at test beds is still a significant factor. Certain changes in the parameter setting during the measurements result in stronger undesired system oscillations than others, and therefore slow down the data recording. The idea is to calculate an optimized measuring order before starting the measuring process ([13]). Less oscillations also yield more robustness and better reproducibility.

Assume a parameter space of dimension D with N measuring points x. Concerning the engine behavior, changing the operating points, i.e. the engine speed (x_1) and the air mass flow (x_2), is more critical than changing the other engine parameters like the valve spreads ($x_3, \ldots, x_d$). Therefore, in the calculation of an ordered list of measuring points, more weight lies on the range spanned by x_1 and x_2, i.e. the operating range. Figure 7 shows a way in a 3-dimensional parameter space, that takes this knowledge into account.

The described problem is a TSP variant. The ordering of the measuring points differs in two respects from the original TSP: First, Hamiltonian paths instead of cycles are of interest, where the x-point with minimal parameter value for each dimension is used as a starting point. Second, the path must be driven in certain directions in order to achieve sufficiently short relaxation times. Hence, instead of the standard Euclidean distance a non-standard metric is used to determine the length of a path. For the use of GAs and MAs,

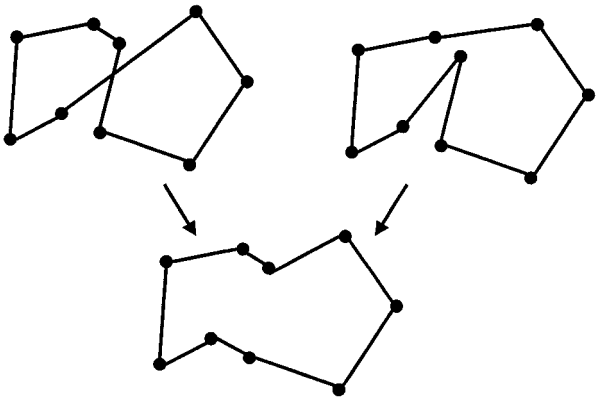


Fig. 8. Visualization of two TSP heuristics: the left arrow represents the 2-opt algorithm, the right arrow represents the random node insertion algorithm.

it was shown earlier, that adjacency coding for list representations of TSP paths, i.e. a locus-based coding, works much better than time-based coding. This was originally suggested in [14]. Again, this kind of coding does not directly allow standard crossover and standard mutation techniques, because this would lead to infeasible offspring. Therefore, repairing algorithms need to be performed after each genetic operation. They are described in detail in [15] and [16]. Roughly speaking, they replace or insert multiple or missing points according to the order within the parent individuals.

To introduce an objective function for this problem, the distance between two points x and $\tilde{x}$ is defined by their Euclidean distance plus the sum of weighted x_i distances:

$$d(x, \tilde{x}) = \|x - \tilde{x}\|_2 + \sum_{i=1}^{d} w_i \cdot |x_i - \tilde{x}_i|.$$

Now an appropriate fitness function for an individual p is defined by

$$\boxed{\phi(p) = \sum_{j=1}^{N-1} d(x^{p(j)}, x^{p(j+1)}),}$$

where p is a permutation of $\{1, \ldots, N\}$ representing a path. Note that the first entry in p is fixed since the starting point is fixed. The weights w_i help to prefer paths that run mainly parallel to the coordinate axis x_i and change the critical parameter less frequently.

4.1 A Hill–Climber for the TSP

For the problem of calculating test bed schedules (TBSs), an algorithm similar to the 2-opt TSP heuristic ([17],[18]) is used to calculate local optimal

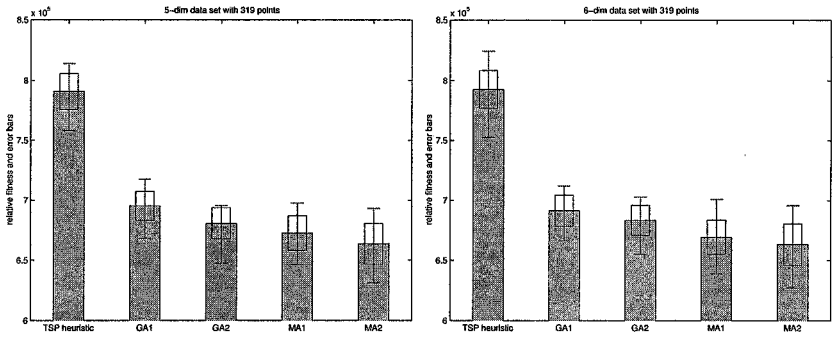


Fig. 9. Comparison of the mean values of the results for the TBS for two data sets. The error bars show the minimum and the maximum values. GA1/2: Hybrid GA with $p_{mut} = 0.1/0.5$. The non-standard Memetic Algorithm MA2 performs better than the standard Memetic Algorithm MA1.

paths through the system's parameter space. This algorithm changes 2 edges between randomly chosen points (see left arrow in figure 8). Here, a combination of this algorithm with a node insertion is used as hill–climbing algorithm. Node insertion takes a randomly selected point and positions it to a randomly chosen other position in the path ([19]). The right arrow in figure 8 visualizes this mechanism. Of course, the more sophisticated systematic Lin-Kernighan local search algorithm ([20]) can be used as well (see e.g. [3] for this topic). Nevertheless, the 2-opt variant is sufficient for the scope of this paper to demonstrate the improvement of a MA. Also the 2-opt variant needs no complete fitness evaluation and is hence very efficient.

4.2 Results for the Test Bed Scheduling

For the TBS problem, instead of mutation, the hill–climbing algorithm is performed. Here, additional mutation operations like the non-sequential 4-change for the TSP ([20]) may lead to further improvement, because of more efficient diversification. Here, 2 data sets with 319 points in 5 and in 6 dimensions are used. The algorithm parameters are: $\mu = 100$, $\lambda = 100$, $t_{max} = 1000$, heuristic mutation with $p_{mut} = \{0.1, 0.5\}$, $p_{cross} = 0.6$, tournament selection with 4 individuals, $\nu = 10$, niche factor 0.4. The MA use $319 \cdot 1000$ hill–climbing iterations, $p_{cross} = 0.5$, and $p_{mut} = 0.5$. Again the hybrid Genetic Algorithms GA1 and GA2 perform local search as mutation operation with $p_{mut} = 0.1$ and $p_{mut} = 0.5$ respectively. Figure 9 shows the results of 20 runs. The non-standard Memetic Algorithm MA2 performs best. This is probably due to the higher number of hill–climbing iterations.

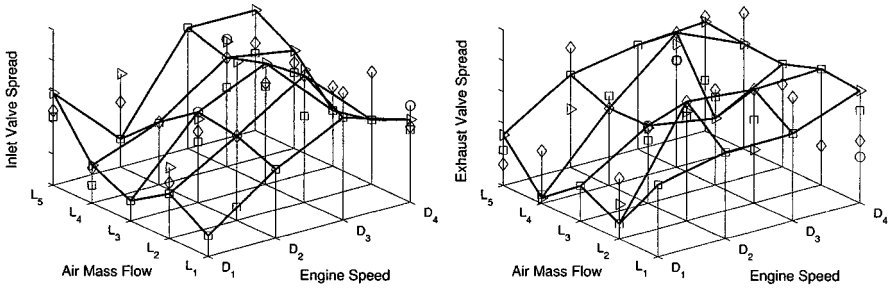


Fig. 10. Result of modeling and optimization: The look-up tables for two control functions are shown. Taking a candidate that is labeled with a diamond at one grid position of the first look-up table, automatically forces to take the diamond at the same grid position of the second look-up table.

5 Final Look-up Table Design

After the optimization of the engine by means of different engine models and optimization methods, normally several optimum parameter candidates result at each grid position of a look-up table. These candidates vary only slightly in their quality, i.e. the resulting fuel consumption and exhaust emission, but may differ significantly in their parameter combinations. Since the final look-up table that is stored within the electronic control unit has to be well-defined, a final selection task has to be performed. Figure 10 shows two example look-up tables, where the actual choice might be sub-optimal. At the same grid point of the set of look-up tables it is the only way to choose equally labeled candidates, because the resulting parameter vector of one modeling and optimization branch can't be split. This would surely lead to a worse engine behavior, i.e. an increased fuel consumption and worse exhaust emissions. But at different grid points, squares, diamonds, triangles or circles could be better. Before the final optimum candidates are selected at each operating point, a distinction is necessary: There are mechanically adjusted parameters like the valve spreads that are set by the camshaft, and there are electrically or electronically adjusted parameters like the ignition timing angle. Mechanical adjustment means a waiting time, until the desired value is reached. Therefore it is important that the maps defined by look-up tables are sufficiently smooth to ensure fast transitions. Other actuators or parameters are adjusted electrically or electronically, e.g. the ignition timing angle. In this case there is less need for smooth transitions.

At each operating point the candidate has to be selected, which together with the chosen candidates at the other grid points leads to the smoothest map. In [1] the problem of composing the smoothest map from such a set of candidates was shown to be NP-hard for an abstract smoothness criterion. For the use of GAs, an appropriate representation of the possible solutions

has to be defined. Here, the direct encoding of the chosen candidates, which results in a variable alphabet is used. Each grid point j corresponds to one position of a chromosome to take n_j different values, one for each candidate available. A chromosome v has the form:

$$v = (v_j)_{j=1}^{N} \in \bigotimes_{j=1}^{N} \{1 \dots n_j\}.$$

Here, N is the number of grid points defining the look-up tables, and n_j is the number of available candidates at (x_1^j, x_2^j). Consider the look-up tables for the inlet valve spread and for the exhaust valve spread displayed in figure 10. At 20 grid points there are $(n_j)_{j=1}^{N} = \{1,1,2,4,3,3,3,2,1,4,3,2,3,2,4,3,3,3,1,3\}$ candidates available. Candidates labeled by the same symbol result from the modeling step using the same model. E.g. candidates labeled by squares and diamonds result from two kinds of artificial neural networks. Candidates labeled by triangles and circles result from two polynomial models of different order respectively. The individual that defines the meshes in figure 10 is

$$v = (1,1,1,3,1,2,2,1,1,4,3,1,3,2,2,3,3,1,1,3)$$

Note that variable alphabet coding allows the application of standard mutation, and standard crossover operations, like uniform crossover or n-point crossover.

We use an objective function similar to the one introduced in [21] to get a suitable smoothness criterion for a map M_i defined by the corresponding look-up table,

$$\boxed{\phi(M_i) = \sum_{1 \le i < j \le N} neigh(i,j) \cdot |y^{(i)} - y^{(j)}|,}$$

where the term $neigh(i,j)$ is 1 for neighboring grid points, otherwise 0. For the multi-objective problem of smoothing $i = 1, \dots, n$ maps M_i simultaneously, e.g. the look-up tables for the valve spreads, an aggregation with weighting factors w_i is used:

$$\Phi(M_1, \dots, M_n) = \sum_{i=1}^{n} w_i \cdot \phi(M_i).$$

Using more sophisticated Pareto techniques for multi-objective optimization turned out to give no further improvements but needs significantly more function evaluations.

5.1 A Neighborhood Algorithm for the LTD

In a certain number of iterations, a random grid point is chosen. At this grid point those candidates, which define the most similar optimal parameter vector to the vectors defined by the presently chosen candidates at its

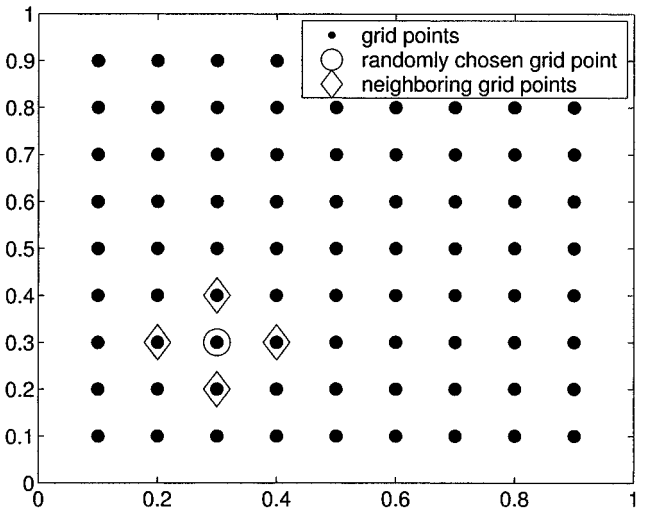


Fig. 11. One step of the neighborhood algorithm for the final look-up table design.

4 neighboring grid points is selected. The similarity between the vectors is given by the Euclidean distance between the vector at the chosen grid point and the mean of the vectors defined by the already selected candidates at the surrounding grid points over all look-up tables (see figure 11). Since the neighborhood heuristic works locally, it is not necessary to evaluate the complete fitness function for each iteration.

5.2 Results for the Final Look-up Table Design

Figure 12 shows the results of the final look-up table design for two data sets. For every data set a number of 20 runs was performed. A grid of size (25×25) with 6 to 10 candidates per grid point is used. The number of simultaneously composed look-up tables is 2 for the first data set, and 4 for the second. For the solution of the second problem with 4 look-up tables the described hill–climbing algorithm turned out to reach its limits. For the second data set, the GA with heuristic initialization performs better than the MAs.

The algorithm parameters are: $\mu = 40$, $\lambda = 40$, $t_{max} = 10000$, $p_{mut} = 0.0016$, 3-point crossover on 2-dimensional encoded individuals, $p_{cross} = 0.6$, tournament selection with 4 individuals, $\nu = 2$, niche factor 0.1. The MA uses hill–climbing with 5000 iterations, $p_{cross} = 0.5$, $p_{mut} = 0.5$.

6 Conclusions

In this contribution, three combinatorial optimization problems occurring in the optimal calibration of automotive combustion engines have been discussed.

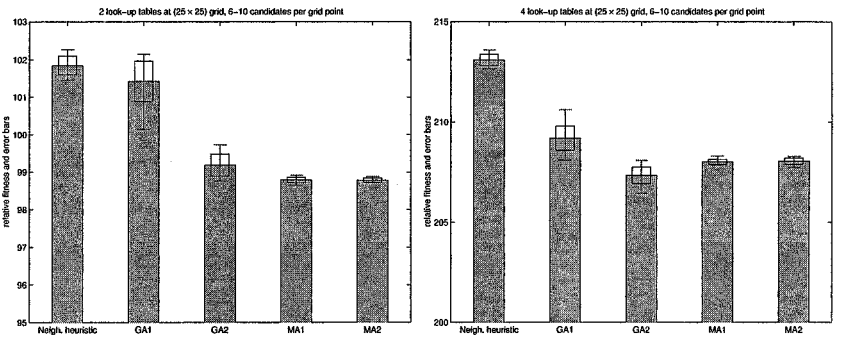


Fig. 12. Results for the final design of look-up tables (LTD). GA1: Standard Genetic Algorithm, GA2: Standard Genetic Algorithm with an initial population calculated by the heuristic. For the case of two look-up tables the best results were achieved by the standard and the non-standard Memetic Algorithm MA1 and MA2. The best results for four look-up tables are given by the GA2. This shows that the power of the heuristic decreases with increasing numbers of look-up tables.

Since the D-optimal design of experiments (DOE), test bed scheduling (TBS), and final look-up table design (LTD) are NP-hard problems, effective heuristics are required to arrive at (near-)optimum solutions. New Memetic Algorithms (MAs) using simple hill–climbers as local search have been proposed for the three problems and several experiments have been conducted to assess their effectiveness. The results demonstrate that MAs give further improvements to the solution of these three problems compared to previously developed algorithms. In particular, the MAs have been shown to be superior to GAs. In most cases, not only traditional GAs but also the hybrid GAs are inferior to the consequent application of local search algorithms after each evolutionary operation. To compensate the high number of fitness evaluations of MAs, the number of generations was increased for the use of GAs. Thereby, approximately the same number of fitness evaluations were achieved. This did not result in any further improvements of the results. Moreover, another possibility was considered: The increase of population size. Here, population sizes up to $\mu = 200$ individuals turned out to give no benefit, also.

The results of the experiments can be summarized as follows: For LTD, the new results for the first data set with 2 look-up tables obtained by the MAs were up to 4% better than those obtained by the GAs. For the second data set, the GA with a near-optimum initial population performed best. For the other two problems the differences are even more severe: For the TBS, improvements of 17% compared to the hill-climbing and of 6% compared to the hybrid GAs were achieved. For the DOE improvements of up to 26% compared to the hill-climbing and of 4% compared to the hybrid GAs were achieved.

Future work will cover further crossover and mutation operators, and systematic local search algorithms for all three problems. Especially for the TBS problem, systematic local search algorithms like the Lin-Kernighan algorithm with the non-sequential 4-change mutation ([20]) will be considered.

Acknowledgments.

We would like to thank Thomas Fleischhauer, and Frank Zuber-Goos at BMW Group Munich for helpful discussions. This research has been supported by the BMBF (grant no. 01 IB 805 A/1).

References

1. Poland, J.: Finding smooth maps is NP-complete. Information Processing Letters **85** (2003) 249–253
2. Moscato, P.: Memetic algorithms: A short introduction. In D. Corne, M.D., Glover, F., eds.: New Ideas in Optimization, London, McGrawHill (1999) 219–234
3. Merz, P.: Memetic Algorithms for Combinatorial Optimization Problems. PhD thesis, Universität-Gesamthochschule Siegen (2000)
4. Goldberg, D.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley (1989)
5. Merz, P., Freisleben, B.: Fitness Landscapes, Memetic Algorithms and Greedy Operators for Graph Bi-Partitioning. Evolutionary Computation **8** (2000) 61–91
6. Merz, P., Freisleben, B.: Fitness Landscape Analysis and Memetic Algorithms for the Quadratic Assignment Problem. IEEE Transactions on Evolutionary Computation **4** (2000) 337–352
7. Merz, P., Freisleben, B.: Memetic Algorithms for the Traveling Salesman Problem. Complex Systems (2002) To appear.
8. Merz, P., Katayama, K.: A Hybrid Evolutionary Local Search Approach for the Unconstrained Binary Quadratic Programming Problem. Bio Systems (2002) To appear.
9. Eshelman, L.: The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination. In Rawlings, G.J.E., ed.: Foundations of Genetic Algorithms. Morgan Kaufmann (1991) 265–283
10. Michalewicz, Z.: A survey of constraint handling techniques in evolutionary computation methods. In J. R. McDonnell, R.G.R., Fogel, D.B., eds.: Fourth Annual Conference on Evolutionary Programming, Cambridge, MA (1995)
11. Poland, J., Mitterer, A., Knödler, K., Zell, A.: Genetic algorithms can improve the construction of d-optimal experimental designs. In: Advances In Fuzzy Systems and Evolutionary Computation, Proceedings of WSES Conference. (2001)
12. Cook, R.D., Nachtsheim, C.J.: A comparison of algorithms for constructing exact d-optimal designs. Technometrics **22** (1980)
13. Knödler, K., Poland, J., Mitterer, A., Zell, A.: Optimizing data measurements at test beds using multi-step genetic algorithms. In: Advances In Fuzzy Systems and Evolutionary Computation, Proceedings of WSES Conference. (2001)

14. Grefenstette, J., Gopal, R., Rosmaita, B., Gucht, D.V.: Genetic algorithms for the travelling salesman problem. Proceedings of the first International Conference on Genetic Algorithms and Application (1985)
15. Bui, T.N., Moon, B.R.: A new genetic approach for the traveling salesman problem. International Conference on Evolutionary Computation (1994)
16. Homaifar, A., Guan, S., Liepins, G.E.: A new approach on the traveling salesman problem by genetic algorithms. 5th International Conference on Genetic Algorithms (1993)
17. Croes, G.: A method for solving traveling salesman problems. Operations Research **5** (1958) 791–812
18. Lin, S.: Computer solutions of the traveling salesman problem. Bell System Technical Journal **44** (1965) 2245–2269
19. Reinelt, D.: The Traveling Salesman: Computational Solutions for TSP Applications. Volume 840. Springer-Verlag, Berlin, Germany (1994)
20. Lin, S., Kernighan, B.: An effective heuristic algorithm for the traveling salesman problem. Operations Research **21** (1973) 498–516
21. Poland, J., Knödler, K., Zell, A., Mitterer, A., Fleischhauer, T., Zuber-Goos, F.: Evolutionary search for smooth maps in motor control unit calibration. In: Stochastic Algorithms: Foundations and Applications (SAGA). (2001)

The Co-Evolution of Memetic Algorithms for Protein Structure Prediction

J.E. Smith

Faculty of Computing, Engineering and Mathematical Sciences,
University of the West of England,
Bristol BS16 12QY, U.K.
james.smith@uwe.ac.uk
http://www.cems.uwe.ac.uk/~jsmith

Summary. This paper describes a co-evolutionary learning-optimisation approach to Protein Structure Prediction which uses a Memetic Algorithm as its underlying search method. Instance-specific knowledge can be learned, stored and applied by the system in the form of a population of rules. These rules determine the neighbourhoods used by the local search process, which is applied to each member of the co-evolving population of candidate solutions.

A generic co-evolutionary framework is proposed for this approach, and the implementation of a simple Self-Adaptive instantiation is described. A rule defining the local search's move operator is encoded as a $\{condition : action\}$ pair and added to the genotype of each individual. It is demonstrated that the action of mutation and crossover on the patterns encoded in these rules, coupled with the action of selection on the resultant phenotypes is sufficient to permit the discovery and propagation of knowledge about the instance being optimised.

The algorithm is benchmarked against a simple Genetic Algorithm, a Memetic Algorithm using a fixed neighbourhood function, and a similar Memetic Algorithm which uses random (rather than evolved) rules and shows significant improvements in terms of the ability to locate optimum configurations using Dill's HP model. It is shown that this "meta-learning" of problem features provides a means of creating highly scalable algorithms.

1 Introduction

The performance benefits which can be achieved by hybridising evolutionary algorithms (EAs) with local search operators, so-called *Memetic Algorithms* (MAs), have now been well documented across a wide range of problem domains such as combinatorial optimisation [27], optimisation of non-stationary functions [42], and multi-objective optimisation [20] (see [29] for a comprehensive bibliography). Commonly in these algorithms, a local search improvement step is performed on each of the products of the generating (recombination

and mutation) operators, prior to selection for the next population There are of course many variants on this theme, for example one or more of the generating operators may be absent, or the order in which the operators are applied may vary. The local search step can be illustrated by the pseudo-code below:

```
Local_Search(i) :
Begin
  /* given a starting solution i and a neighbourhood function n */
  set best = i;
  set iterations = 0;
  Repeat Until ( iteration condition is satisfied ) Do
    set counter = 0;
    Repeat Until ( termination condition is satisfied ) Do
      generate the next neighbour j ∈ n(i);
      set counter = counter + 1;
      If (f(j) is better than f(best)) Then
        set best = j;
      endIf
    endDo
    set i = best;
    set iterations = iterations + 1;
  endDo
End.
```

There are three principal components which affect the workings of this local search. The first is the choice of pivot rule, which can be *Steepest Ascent* or *Greedy Ascent*. In the former the termination condition is that the entire neighbourhood $n(i)$ has been searched, i.e. $counter = |n(i)|$, whereas the latter stops as soon as an improvement is found; i.e. the termination condition is $(counter = |n(i)|) \vee (best \neq i)$. Note that some authors resort to only considering a randomly drawn sample of size $N << |n(i)|$ if the neighbourhood is too large to search.

The second component is the depth of the local search, i.e. the iteration condition which lies in the continuum between only one improving step being applied ($iterations = 1$) to the search continuing to local optimality ($(counter = |n(i)|) \wedge (best = i)$). Considerable attention has been paid to studying the effect of changing this parameter within MAs e.g. [14]. Along with the choice of pivot rule, it can be shown to have an effect on the performance of the Local Search algorithm, both in terms of time taken, and in the quality of solution found.

The third, and primary factor that affects the behaviour of the local search is the choice of neighbourhood generating function. This can be thought of as defining a set of points $n(i)$ that can be reached by the application of some move operator to the point i. An equivalent representation is as a graph $G = (v, e)$ where the set of vertices v are the points in the search space, and

the edges relate to applications of the move operator i.e $e_{ij} \in G \iff j \in n(i)$. The provision of a scalar fitness value, f, defined over the search space means that we can consider the graphs defined by different move operators as "fitness landscapes" [15]. Merz and Freisleben [28] present a number of statistical measures which can be used to characterise fitness landscapes, and have been proposed as potential measures of problem difficulty. They show that the choice of move operator can have a dramatic effect on the efficiency and effectiveness of the Local Search, and hence of the resultant MA.

In some cases, domain specific information may be used to guide the choice of neighbourhood structure within the local search algorithms. However, it has recently been shown that the optimal choice of operators can be not only instance specific within a class of problems [28, pp254–258], but also dependent on the state of the evolutionary search [26]. This result is not surprising when we consider that points which are locally optimal with respect to one neighbourhood structure may not be with respect to another (unless of course they are globally optimal). Thus if a set of points has converged to the state where all are locally optimal with respect to the current neighbourhood operator, then changing the neighbourhood operator may provide a means of progression, in addition to recombination and mutation. This observation forms the heart of the *Variable Neighbourhood Search* algorithm [49].

This paper describes one mechanism whereby the definitions of local search operators applied within the MA may be changed during the course of optimisation, and in particular how this system may usefully be applied to a simplified model of the Protein Structure Prediction Problem. This system is called Co-evolving Memetic Algorithms (COMA). The rest of this paper proceeds as follows:

- Section 2 discusses some previous work in this area, describes the proposed approach, and the development of a simplified model within that framework. It also summarises the results of initial investigations published elsewhere.
- Section 3 draws some parallels between this work and related work in different fields, in order to place this work within the context of more general studies into adaptation, development and learning.
- Section 4 details the particular application under concern, namely Protein Structure Prediction using Dill's HP model [8].
- Section 5 presents the results and analysis of a set of preliminary experiments designed to investigate whether the use of adaptive rules is able to benefit the optimisation process.
- Section 6 goes on to investigate the benefits of restricting the search to feasible solutions, rather than using a penalty function approach.
- Section 7 presents some analyses of the behaviour of the evolving rule-bases, and then Section 6 discusses the implications of these results, before drawing conclusions and suggesting future work.

2 A Rule-Based Model for the Adaptation of Move Operators

2.1 The Model

The aim of this work is to provide a means whereby the definition of the local search operator (LSO) used within a MA can be varied over time, and then to examine whether evolutionary processes can be used to control that variation, so that a beneficial adaptation takes place. Accomplishing this aim requires the provision of five major components, namely:

- A means of representing a LSO in a form that can be processed by an evolutionary algorithm
- Intimately related to this, a set of variation operators, such as recombination and mutation that can be applied to the LSO representation, and a means for initialising a population of LSO operators.
- A means of assigning fitness to the LSO population members
- A choice of population structures and sizes, along with selection and replacement methods for managing the LSO population
- A set of experiments, problems and measurements designed to permit evaluation and analysis of the behaviour of the system.

The representation chosen for the LSOs is a tuple *<Pivot_Rule, Depth, Pairing, Move, Fitness>*.

The first two elements in the tuple have been described above and can be easily mapped onto an integer or cardinal representation as desired, and manipulated by standard genetic operators.

The element *Pairing* effectively co-ordinates the evolution of the two populations. When a candidate solution is to be evaluated, a member of the LSO population is chosen to operate on it, hopefully yielding improvements. The fitness of the candidate solution is thus affected by the choice of LSO to operate on it, and the fitness assigned to the LSO is in turn affected by the candidate solution to which it is applied.

Values for *Pairing* are taken from the set {*linked, fitness_based, random*}. The purpose of this element is to allow the system to be varied between the extremes of a fully unlinked system, in which although still interacting the two populations evolve separately, and a fully linked system in which the LS operators can be considered to be self-adapted. The different values have the following effects:

- For a *linked* pairing strategy, the LSOs can be considered to be extra genetic material which is inherited and varied along with the problem representation. Thus if the k^{th} candidate solution is created from parents i and j, then a LSO is created by the actions of recombination and mutation on members i and j of the current LSO population. This new LSO is used to evaluate the new candidate solution and becomes the k^{th} member of

the next LSO population. Note that this assumes the two population are the same size. The fitness is assigned to the new LSO is immaterial since selection to act as parents happens via association with good members of the solution population.

- For a *fitness-based* pairing strategy, when a candidate solution requires evaluation, a LSO is created and put into the next LSO population as above. However the two LSOs which acts as parents for recombination are now chosen using a standard selection mechanism acting on those members of the current LSO population which do not have $Pairing = linked$. A number of methods can be used to define the fitness of an LSO.
- For a *random* pairing strategy, the same process occurs as for the fitness-based method, except that parents are selected randomly from the unlinked members of the LSO population, without regard to their fitness.

Although the long-term goal is to examine a "mixed-economy" of paring strategies, for the purposes of this paper the system is restricted to the situation where the whole population uses the same value, initially $Pairing = linked$.

The representation chosen for the move operators was as *condition:action* pairs, which specify a pattern to be looked for in the problem representation, and a different pattern it should be changed to. Although this representation at first appears very simple, it has the potential to represent highly complex moves via the use of symbols to denote not only single/multiple wildcard characters (in a manner similar to that used for regular expressions in Unix) but also the specifications of repetitions and iterations. Further, permitting the use of different length patterns in the *condition* and *action* parts of the rule gives scope for *cut* and *splice* operators working on variable length solutions.

In themselves, the degrees of freedom afforded by the five components listed above provide basis for a major body of research, and the framework described above is intended to permit a full exploration of these issues which is currently underway [37, 36].

This paper presents results from a simplified instantiation of this framework, focusing on the benefits of knowledge discovery and re-use. In order to achieve this focus, some of the adaptive capabilities are restricted, i.e., the LSOs always use one of greedy or steepest ascent, a single improvement step, and full linkage. These choices are coded into the LSO chromosomes at initialisation, and variation operator are not used on them. This restriction to what are effectively self-adaptive systems provides a means of dealing with the credit assignment and population management issues noted above

The COMA system is also restricted to considering only rules where the *condition* and *action* patterns are of equal length and are composed of values taken from the set of permissible allele values of the problem representation, augmented by a "don't care" symbol # which is allowed to appear in the *condition* part of the rule but not the *action*, although this could be interpreted as "leave alone". The neighbourhood of a point i then consists of all those

points where the substring denoted by *condition* appears in the representation of i and is replaced by the *action*. The neighbourhood of i therefore potentially includes i itself, for example by means of a rule with identical *condition* and *action* parts.

To give an example, if a solution is represented by the binary string 1100111000 and a rule by 1#0:111, then this rule matches the first, second, sixth and seventh positions, and the neighbourhood is the set {1110111000, 11111111000, 1100111100,1100111110}. In practice a random permutation is used to specify the order in which the neighbours are evaluated, so as not to introduce positional bias into the local search when greedy ascent is used. Note that in this work the string is not considered as toroidal (although this will be considered in later work).

In practice, each rule was implemented as two 16 bit strings, and was augmented by a value *rule_length* which detailed the number of positions in the pattern string to consider. This permits not only the examination of the effects of different fixed rule sizes, but also the ability to adapt via the action of mutation operators on this value. This representation for the rules means that "standard" genetic operators (uniform/1 point crossover, point mutation) can be used to vary this part of the LSO chromosome.

2.2 Initial Results

The results of initial investigations using this system were reported in [37]. The test suite was problems made out of a number of sub-functions either interleaved or concatenated. Two different classes of sub-function were used which posed either entropic (Royal Road) or fitness (Deceptive) barriers to the discovery of the global optimum. Greedy versions of the COMA (GComa) algorithm were tested against the GA,MA, GRand algorithms described below, and it was shown that a version of the system with adaptive rule lengths was able to perform better than these three, and comparably with variants of GComa with optimal fixed rule-lengths for the different problems. Analysis showed that these algorithms discovered and used problem specific information (such as optimal patterns for different sub-problems).

Subsequent work [36] has shown them to be highly scalable with respect to problem length on problems where there are repeated patterns in the regions of the search space corresponding to high quality solutions. This behaviour arises from the discovery and re-use of knowledge about these patterns. It was also shown that in the absence of such patterns, the systems still displays better performance (both in terms of mean best fitness and the reliability of locating the global optimum). In this case the improved performance arose from the maintenance of a diverse set of move operators, and hence from the examination of multiple search landscapes, which provides a better means of escaping local optima.

3 Related Work

The COMA system can be related to a number of different branches of research, all of which offer different perspectives and means of analysing it's behaviour. These range from MultiMemetic Algorithms and the Self-Adaptation of search strategies, through co-evolutionary, learning and developmental systems, to the evolutionary search for generalised rules as per Learning Classifier Systems. Space precludes a full discussion of each of these, so the more important are briefly outlined below.

Although the authors are not aware of other algorithms in which the LSOs used by an MA are adapted in this fashion, there are other examples of the use of multiple LS operators within evolutionary systems. Krasnogor and Smith [26] describe a "MultiMemetic Algorithm", in which a gene is added to the end of each chromosome indicating which of a fixed set of static LS operators ("memes") should be applied to the individual solution. Variation is provided during the mutation process, by randomly resetting this value with a low probability. They report that their systems are able to adapt to use the best meme available for different instances of TSP.

Krasnogor and Gustafson have extended this and proposed a grammar for "Self-Generating MAs" which specifies, for instance, where in the evolutionary cycle local search takes place [22]. Noting that each meme potentially defines a different neighbourhood function for the local search part of the MA, we can also see an obvious analogy to the Variable Neighbourhood Search algorithm [49], where a heuristic is used to control the order of application of a set of local searchers (using different, fixed, neighbourhood structures) to a single improving solution. The difference here lies in the population based nature of COMA, so that not only do we have multiple candidate solutions, but also multiple adaptive neighbourhood functions in the memes.

As noted above, if the populations are of the same size, and are considered to be linked, then this instantiation of the COMA framework can be considered as a type of Self Adaptation. The use of the intrinsic evolutionary processes to adapt step sizes governing the mutation of real-valued variables has long been used in Evolution Strategies [35], and Evolutionary Programming [11]. Similar approaches have been used to self-adapt mutation probabilities [2, 39] and recombination operators [34] in genetic algorithms as well as complex generating operators which combined both mutation and recombination [38]. This body of work contains many useful results concerning the conditions necessary for strategy adaptation, which could be used to guide implementations of COMA.

If the two populations are not linked, then COMA is a co-operative coevolutionary system, where the fitness of the members of the LSO population is assigned as some function of the relative improvement they cause in the "solution" population. Paredis has examined the co-evolution of solutions and their representations [31], and Potter and DeJong have also used co-operative co-evolution of partial solutions in situations where an obvious problem de-

composition was available [33], both with good reported results. Bull [5] conducted a series of more general studies on co-operative co-evolution using Kauffmann's static NKC model [17]. In [7] he examined the evolution of linkage flags in co-evolving "symbiotic" systems and showed that the strategies which emerge depend heavily on the extent to which the two populations affect each others fitness landscape, with linkage preferred in highly interdependent situations. He also examined the effect of different pairing strategies [6], with mixed results, although the NKC systems he investigated used fixed interaction patterns, whereas in the systems used here are more dynamic in nature.

There has also been a large body of research into competitive-coevolution, (an overview can be seen in [32]) whereby the fitnesses assigned to the two populations are directly related to how well individuals perform "against" the other population, what has been termed "predator-prey" interactions.

In both the co-operative and competitive co-evolutionary work cited above, the different populations only affect each other's perceived fitness, unlike the COMA framework where the LSO population can directly affect the genotypes within the solution population. A major source of debate and research within the community has focused around the perception that this phase of improvement by LS can be viewed as a kind of lifetime learning. This has lead naturally to speculation and research into whether the modified phenotype which is the outcome of the improvement process should be written back into the genotype (Lamarkian Learning) or not (Baldwinian Learning). Note that although the pseudo code of the local search, and the discussion above assumes Lamarkian learning, this is not a prerequisite of the framework. However, even if a Baldwinian approach was used, the principal difference between the COMA approach and the co-evolutionary systems above lies in the fact that there is a selection phase within the local search, that is to say that if all of the neighbours of a point defined by the LSO rule are of inferior fitness, then the point is retained unchanged within the population.

If one was to discard this criterion and simply apply the rule (possibly iteratively), the system could be viewed as a type of "developmental learning" akin to the studies in Genetic Code e.g. [16] and the "Developmental Genetic Programming" of Keller and Banzhaf [18, 19]

Finally, and perhaps most importantly, it should be considered that if a rule has an improving effect on different parts of a solution chromosome over as number of generations, then the evolution of rules can be seen as learning generalisations about patterns within the problem representation, and hence the solution space. This point of view is akin to that of Learning Classifier Systems. For the case of unlinked fitness-based selection of LS operators, insight from this field can be used to guide the credit assignment process.

It is tempting to draw a further generalisation which would see the *conditions* as representing schema and the *actions* as representing higher fitness (and possibly higher order) alternatives, but this is a more dubious analogy as the conditions are allowed to match anywhere within the string, i.e. even a

fully specified rule of length l matches $L - l$ schema within a string of length L.

4 Dill's HP model of Protein Structure Prediction

The problem of Protein Structure Prediction (PSP), i.e. the prediction of the "native" three-dimensional form of a protein from knowledge of the sequence of its constituent amino-acid residues is one of the foremost challenges facing computational biology. Current approaches to PSP can be divided into three classes; comparative modelling, fold recognition, and *ab initio* methods. The first two explicitly search the ever-growing databases of known structures for similar sequences (homologues) and sub-sequences. In contrast, the third approach represents the "last chance" scenario of trying to predict the tertiary structure by minimising a free energy model of the structure. Approaches that make use of existing knowledge currently represent the state of the art (and are likely to remain so), however ab initio approaches are important for two main reasons. The first of these relates to the situation where a sequence does not correspond to any known fold. The second, and more fundamental reason is that the development of true ab initio methods can give greater insight into the relationship between different fold families, and to the dynamical process of folding.

Current approaches to ab initio PSP can be divided according to two criteria, namely the nature of the choice of energy function, and the number of degrees of freedom in the conformation, as exemplified by the granularity (all atom models vs. virtual atom) and locational constraints (e.g. lattice based models vs. off-lattice models). Although most lattice based models are physically unrealistic, they have proved a useful tool for exploring issues within the field. Some of the more complex models, e.g. SICHO [21] have been shown to be capable of accurate predictions of the conformations of simple proteins, especially when used in conjunction with techniques for subsequent refinement to an all-atom model [10].

The HP model for PSP [8] provides an estimate of the free energy of a fold of a given instance, based on the summation of pair-wise interactions between the amino acid residues. It is a "virtual residue" model, that is to say that each amino acid residue is modelled by a single atom, whose properties are reduced to a quality of being hydrophobic or hydrophilic, thus simplifying the energy calculations still further. Hydrophobic residues avoid interacting with the water molecules of the solvent, whereas hydrophilic (or polar) residues are able to form hydrogen bonds with the water molecules. Thus, polar residues are often found at the surface of the protein and hydrophobic residues are normally found buried in the inner part, or core, of the protein. The HP model captures this behaviour, despite its extreme simplicity. In the model, a sequence of l amino acid residues is represented by $s \in \{H, P\}^l$, where H represents a hydrophobic amino acid and P represents a hydrophilic one. The

space of valid conformations is restricted to self-avoiding paths on a selected lattice, with each amino acid located on a vertex. The torsion angles of the peptide bonds between residues are thus restricted by a finite set determined by the shape of the lattice. The first amino acid of the sequence is located on a randomly selected vertex, and an orientation is assumed for it. From there, according to the orientation, the chain grows, placing every subsequent amino acid either ahead of the previous one, at 90 degrees to the left or at 90 degrees to the right (assuming a square lattice). Hydrophobic units that are adjacent in the lattice but non-adjacent in the sequence add a constant negative factor to the energy level. All other interactions are ignored. In some cases, to make feasible conformations more attractive, the infeasible folds suffer penalisation in the form of adding a substantial positive factor to their energy levels. In this way, the model reflects the tendency of hydrophobic amino acids to form a hydrophobic core. Despite the apparent simplicity of this model, the search for the global energy minimum in the space of possible conformations of a given sequence has been shown to be NP complete on various lattices [4].

Evolutionary algorithms (in particular Genetic Algorithms) have been applied, with some success, to the PSP using the HP and all-atom off-lattice models, by a number of authors since [41, 40]. In [23] the effect of different encoding schemes and constraint management techniques were examined, and a modified fitness function was developed which extends the basic HP model to permit the allocation of reward for non-adjacent pairs of Hydrophilic residues. More recent work has demonstrated the use of self-adaptation within a memetic algorithm to permit the selection from amongst a fixed set of predetermined local search strategies, using different move operators such as local "stretches", reflections etc [25, 30]. The work described here extends this by not relying on a fixed set of move operators encoding domain-specific knowledge, but rather evolving a set of move operators, thus *learning* that domain-specific knowledge.

5 Experimental Results

5.1 The Test Suite and Experimental set-up

In order to investigate the value of this approach, 20 instances and parameter settings from [24], were used, which use a two-dimensional triangular lattice. These instances are detailed in Table 1.

The representation used is a *relative* encoding. In this, where the alleles come from the set {*leftback, leftforward, front, rightforward, rightback*} and represent the direction of the next move on the lattice from the point of view of the head of the growing chain. This is an alternative to the *absolute* encoding used by Unger and Moult [41], where alleles specify directions to move relative to an external frame of reference. Results presented in [23] have suggested that this relative encoding is preferable, not least because the absence of a "back"

Table 1. HP instances used in these experiments

Id	Sequence	Length	Optimum
1	HHPHPHPHPHPH	12	11
2	HHPPHPHPHPHPHP	14	11
3	HHPPHPPHPHPHPH	14	11
4	HHPHPPHPPHPPHPPH	16	11
5	HHPPHPPHPHPHPPHP	16	11
6	HHPPHPPHPPHPPHPPH	17	11
7	HHPHPHPHPHPHPHPHH	17	17
8	HHPPHPPHPHPHPPHPHPHH	20	17
9	HHPHPHPHPHPPHPPHPPHH	20	17
10	HHPPHPPHPHPPHPHPPHPHH	21	17
11	HHPHPPHPPHPHPHPPHPPHH	21	17
12	HHPPHPHPHPPHPHPPHPPHH	21	17
13	HHPPHPPHPHPHPPHPPHPPHH	22	17
14	HHHPHPHPHPHPHPHPHPHPHHH	23	25
15	HHPPHPPHPPHPPHPPHPPHPPHH	24	17
16	HHHPHPHPPHPHPHPHPHPHPHHH	24	25
17	HHHPHPHPHPPHPHPHPHPHPHHH	24	25
18	HHHPPHPPHPPHPPHPHPPHPHPPHPPHHH	30	25
19	HHHPPHPPHPPHPHPPHPHPPHPPHPPHHH	30	25
20	HHHPPHPPHPPHPHPHPPHPPHPPHPPPPPHPHPHHH	37	29

move means that all conformations that can be represented are one-step self-avoiding.

The generational genetic algorithm used (500+500) selection. One Point Crossover was applied with probability 0.8 and a Double Mutation was made with probability 0.3. Viewed from an external frame of reference the mutation operator has the effect of causing the mutation point to act as a pivot, about which one half of the structure is rotated through some multiple of $\pi/6$ (for a triangular lattice). Mutation was applied to the rules with a probability of 0.0625 of selecting a new allele value in each locus (the inverse of the maximum rule length).

For each combination of algorithm and instance, 25 runs were made, each run continued until the global optimum was reached, subject to a maximum of 1 million evaluations. Note that since one iteration of a local search may involve several evaluations, this allows more generations to the GA, i.e. algorithms are compared strictly on the basis of the number of calls to the evaluation function. The algorithms used (and the abbreviations which will be used to refer to them hereafter) are as follows:

- A GA i.e. with no use of Local Search (GA).
- A simple MA using a bit-flipping neighbourhood, with one iteration of greedy ascent (SMA).

- Versions of COMA using a randomly created rule in each application, i.e. with the learning disabled. One iteration of steepest (SRand) or greedy (GRand) ascent local search was applied.
- Adaptive versions of COMA with the two pivot rules (SComa and GComa). In these the rule lengths are randomly initialised in the range [1,16]. During mutation, a value of $+/-1$ is randomly chosen and added with probability 0.0625.

These results are analysed according to three different performance criteria: firstly the Success Rate (the number of runs in which the global optimum was found), secondly in terms of efficiency, as measured by the average number of evaluations to solution (AES) in those successful runs, and thirdly in terms of the mean performance measured in terms of the best value found in the maximum time alloted, averaged over 25 runs.

5.2 Success Rate

Table 2 shows the Success Rate for each algorithm itemised by instance and in total. Using a non-parametric Friedman's test for k-related variables shows that the differences in success rate between algorithms is significant, and a series of paired t-tests confirms that the results for the SComa algorithm are significantly better than any of the others with over 95% confidence. This difference is particularly noticeable on the longer instances. Of the other results, the simple MA (SMA) performs well on the shorter instances, and the GComa and GRand results are surprisingly similar. This may well be due to the noise inherent in the greedy ascent mechanism making it hard for the credit assignment mechanism to function properly as was previously noted in [36]. Significantly, whatever the form of the local search phase, all but one of the Memetic Algorithms perform much better than the simple GA. The least reliable algorithm was SRand, and possible reasons for this will be discussed further in the following section.

5.3 Efficiency

Figure 1 shows the Average Evaluations to Solution (i.e., the globally optimal conformation) for the runs in which algorithms were successful. Immediately we can see that even when it is successful, the SRand algorithm is far slower than all of the other algorithms. Like the more successful GRand algorithm, it is using a randomly created rule to define the neighbourhood for each solution in each generation. However, unlike the GRand algorithm it is searching the whole of each neighbourhood, and the increase in the AES values suggests that the neighbourhoods are generally quite large. This suggests the frequent use of short, low rules of low specificity, i.e. with lots of #'s. It is possible that left to run for longer, the Success Rate of the SRand algorithm would have been improved.

Table 2. Number of runs (out of 25) in which the minimum energy conformation was identified

	algorithm					
instance	GComa	SComa	GRand	SRand	SMA	GA
1	13	25	16	16	25	13
2	14	25	15	7	23	13
3	15	24	10	11	22	7
4	19	25	17	2	24	13
5	13	25	13	7	22	9
6	10	24	11	0	20	9
7	9	24	5	1	14	3
8	7	25	6	0	11	2
9	4	22	5	0	4	2
10	4	21	4	0	10	2
11	5	21	7	0	7	2
12	7	22	7	0	12	4
13	6	21	3	0	7	2
14	0	7	0	0	0	0
15	0	9	1	0	0	2
16	1	7	0	0	1	0
17	0	8	0	0	0	0
18	0	1	0	0	0	0
19	0	1	0	0	0	0
Total	127	337	120	44	202	83

Of the others, the GA is always fastest, followed by the SMA. The rest of the picture is less clear, although the greedy versions are usually faster than their steepest ascent counterparts. A two way Analysis of Variance, with instance and algorithm as factors, shows that both are significant, and post-hoc analysis using the Least-Significant Difference test shows that the ordering GA < SMA < {GRand,GComa} < SComa < SRand is significant with 95% confidence. If we do not assume equal variance, Tamhane's T2 test shows that the GA is significantly faster, but under these more cautious assumptions the SMA is only significantly faster than GRand with 93% confidence and is not significantly faster than GComa. Similarly GRand and SComa are no longer significantly different in speed of finding solutions.

5.4 Mean Best Fitness

As was evidenced in Table 2 it is not hard to find solutions for the shorter instances. Therefore when comparing performance on the basis of the quality of the best solutions found, i.e., mean best fitness (MBF), only results for

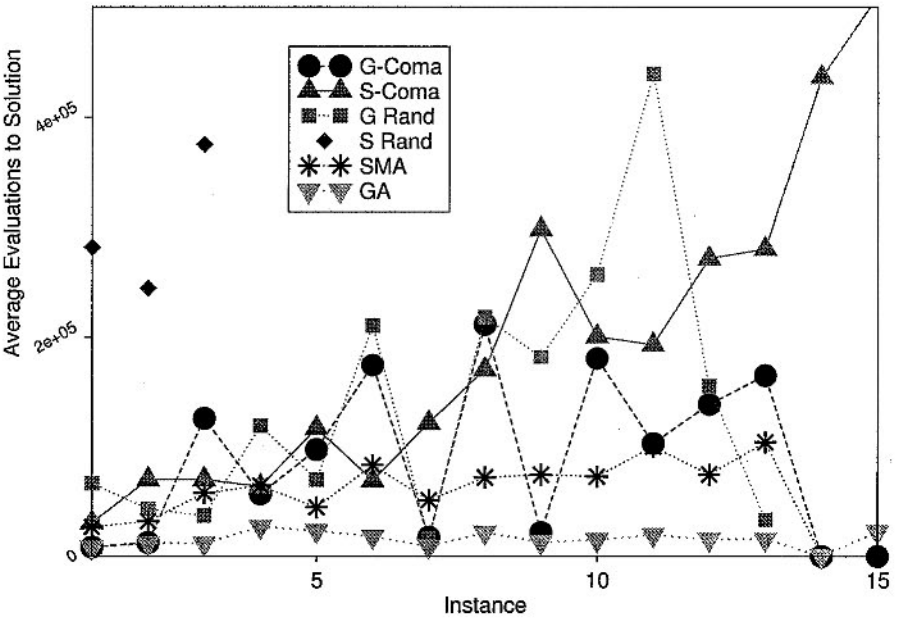


Fig. 1. Average Evaluations to Solution (when found) by algorithm.

the longer and harder instances 14-20 have been considered. Figure 2 shows these results graphically for each algorithm, sorted by instance. From these it is clear that the SComa reaches consistently higher values and with a smaller variance in performance than the others, and that the SRand algorithm is correspondingly worse.

In order to investigate the statistical significance of these results, a two-way ANOVA test was performed on the values for the best solution found in each run, with instance number and algorithm as the factors. This confirmed the significance of the algorithm in determining the performance, and so two sets of post-hoc tests were performed to analyse the differences between pairs of algorithms. These were Least-Significant Difference, and Tamhane's T2 test (the latter is more conservative as it does not make any assumptions about the samples having equal variances). The results of these tests are summarised in Table 3. An entry r or R indicates that the algorithm indicated by the row index was significantly better than the one indicated by the column index, with 95% confidence according to the LSD or T2 test respectively. Similarly an entry of c or C indicates that the column algorithm is better than the row algorithm with 95% confidence according to the LSD or T2 test respectively.

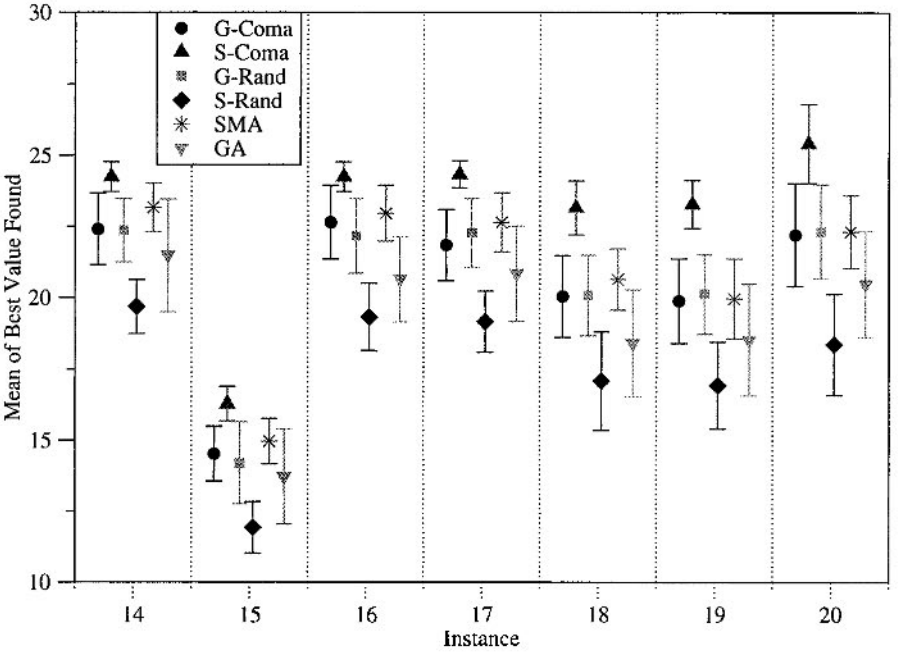


Fig. 2. Mean and std deviation of best values found for instances 14-20, analysed by algorithm

Table 3. Statistical significance of pairwise comparisons between algorithms on basis of best values found. – indicates no significant difference. r[c] denotes algorithm indicated by row[column] is better with 95% confidence. Lower triangle (lower case) is for LSD test, upper quarter (upper case) is for Tamhane's T2 test.

SComa	-	R	R	R	R	R
GComa	c	-	R	-	-	R
SRand	c	c	-	C	C	C
GRand	c	-	r	-	-	-
SMA	c	r	r	r	-	R
GA	c	c	r	c	c	-
Algorithm	SCOMA	GComa	SRand	GRand	SMA	GA

6 Restricting the Search to Feasible Solutions

In [9] results are reported from a detailed study of the fitness landscape of HP model proteins which suggests that the feasible regions of the search space are more highly connected than has previously been thought, and that correspondingly there may be performance advantages arising from a restriction of the search process to only considering feasible solutions.

In order to investigate this, the crossover and mutation operators were modified so that they only produced feasible offspring. This process is less lengthy than it would first appear since in practice infeasible offspring can almost always be quickly identified during the path growth process and the evaluation stopped. However no attempt was made to restrict the initial population to feasible solutions, as the infeasible ones are quickly weeded out by selection, and preliminary experimentation revealed that creating a feasible initial population by random generation of values takes an extremely long time.

The mutation operator still applied one double mutation - a random permutation of the loci was generated, and for each of these a random permutation of the possible changes was created. Offspring were produced and tested in this order until a feasible one was created. The crossover operator was modified similarly: if the offspring produced using a given crossover point was infeasible the operator next tested all of the different possible orientation of the two substrings by varying the allele value in the locus corresponding to that crossover point, before moving on to trying the next.

6.1 Success Rate

Table 4 shows the results from running the GA, SMA and SComa algorithms with the modified crossover and mutation operators, alongside those for the unmodified versions. As can be seen (and statistical testing confirms) there is far better reliability for the GA-F and SMA-F algorithms than their unrestricted counterparts. The results for the SComa are less clear - if anything the performance is better for short instances and worse for long ones, but the difference is not statistically significant.

6.2 Efficiency

Figure 3 shows the efficiency (AES) comparisons for the same set of algorithms, again restricted to successful runs. As when comparing Success Rates, there is little difference between the SComa and SComa-F algorithms, but under this metric the performance of the GA and GA-F algorithms are not significantly different, i.e., the GA is still very efficient on those runs when it does find the optimum, and with the restricted operators it does so far more often. In contrast to this, the SMA algorithm exhibits much greater AES values when restricted to feasible solutions, despite being more successful.

6.3 Mean Best Fitness

As evidenced in Table 4, restricting the search to feasible solutions makes it even easier to find solutions for the shorter instances. Therefore when comparing performance on the basis of the quality of the best solutions found,

Table 4. Effect on Success Rate of restricting search to feasible solutions. Results for GA, SMA and SComa algorithms are shown alongside those using modified crossover and mutation (indicated by -F)

instance	algorithm					
	GA	GA-F	SMA	SMA-F	SCOMA	SCOMA-F
1	13	23	25	25	25	25
2	13	20	23	25	25	25
3	7	17	22	25	24	25
4	13	20	24	25	25	25
5	9	18	22	24	25	25
6	9	18	20	24	24	25
7	3	9	14	23	24	24
8	2	9	11	24	25	25
9	2	8	4	21	22	23
10	2	8	10	22	21	23
11	2	5	7	24	21	21
12	4	12	12	23	22	23
13	2	4	7	24	21	23
14	0	0	0	5	7	9
15	2	1	0	8	9	6
16	0	1	1	2	7	5
17	0	0	0	4	8	0
18	0	0	0	0	1	0
19	0	0	0	0	1	0
total	83	173	202	328	337	332

i.e., mean best fitness (MBF), only results for the longer and harder instances 14-20 have been considered again. Figure 4 shows these results graphically for each algorithm, sorted by instance.

In order to investigate the statistical significance of these results, a two-way ANOVA test was performed on the values for the best solution found in each run, with instance number and algorithm as the factors. This confirmed the significance of the algorithm in determining the performance, and so two sets of post-hoc tests were performed to analyse the differences between pairs of algorithms. These were Least-Significant Difference, and Tamhane's T2 test (the latter is more conservative as it does not make any assumptions about the samples having equal variances). The results of these tests are summarised in Table 5. An entry r or R indicates that the algorithm indicated by the row index was significantly better than the one indicated by the column index, with 95% confidence according to the LSD or T2 test respectively. Similarly an entry of c or C indicates that the column algorithm is better than the row algorithm with 95% confidence according to the LSD or T2 test respectively.

In general it is plain that the rank order is GA < GA-F < SMA< SMA-F < SComa-F < SComa. These differences are generally statistically significant

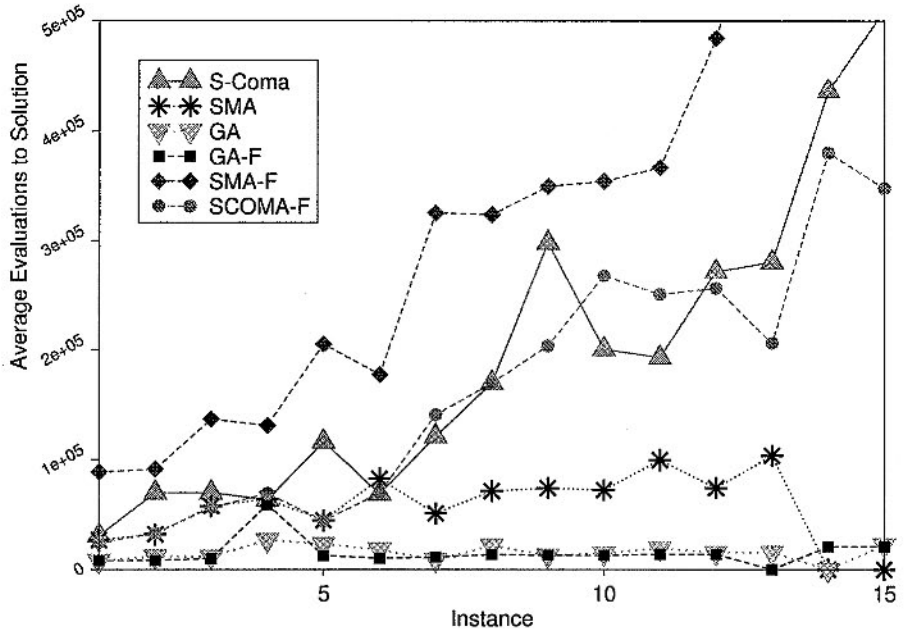


Fig. 3. Effect on efficiency of restricting search to feasible solutions. Plot shows Average Evaluations to Solution for successful runs of GA, SMA, SComa and their restricted counterparts (indicated by –F).

according to both tests, although it should be noted that this depends to some extent on the choice of instances considered. If we include all instances, then the general success on the shorter ones makes the differences less significant, whereas if we restrict ourselves to only considering a few harder instances, the significance increases.

Table 5. Statistical significance of pairwise comparisons between algorithms on basis of best values found. – indicates no significant difference. r[c] denotes algorithm indicated by row[column] is better with 95% confidence. Lower triangle (lower case) is for LSD test, upper quarter (upper case) is for Tamhane's T2 test.

GA	-	C	C	C	C	C
GA-F	r	-	-	C	C	C
SMA	r	-	-	C	C	C
SMA-F	r	r	r	-	C	-
SComa	r	r	r	r	-	-
SComa-F	r	r	r	-	-	-
Algorithm	GA	GA-F	SMA	SMA-F	SComa	SComa-F

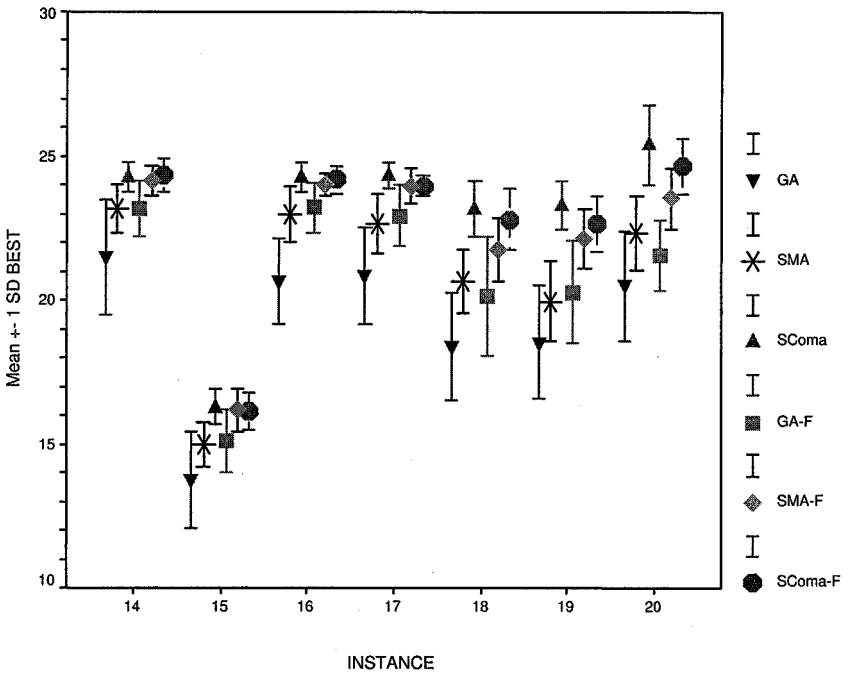


Fig. 4. Mean and std deviation of best values found for instances 14-20, analysed by algorithm

7 Analysis of LSO Evolution

In order to gain a greater understanding of the behaviour of the SComa algorithm, a number of test runs were made in which the contents of the LSO population were output to file at regular intervals.

Examination of the form of the evolving LSOs showed that there was a strong tendency towards short rules of the form $\#\# \rightarrow lr$ or $\#\# \rightarrow lL$. Here l = *leftback*, r = *rightback*, and L = *leftforward* relative to the previous direction of growth. Both of these rules act to bring residues i and $i+2$ into contact, via causing a torsion angle of $\Pi/6$ at residue $i+1$.

Given that the system is evolving conformations in a two-dimensional plane, these patterns these could possibly be thought of as the two-dimensional equivalent of representing a single turn of an alpha helix. Experimentation on a square two-dimensional lattice showed that the rules which evolved on a number of instances tended to have length three and be of the form $\#\#\# \rightarrow lll$ or $\#\#\# \rightarrow rrr$ which is the shortest path that can be made bringing two residues into contact.

The use of the word "tended" should be noted here: in most cases the rule-set continued to contain a number of different rules of varying lengths. It has been argued elsewhere [36] that in addition to the extra scalability attained by identifying and re-applying regular structural motifs, the presence of a diverse, evolving rule-set means that the neighbourhood structure defining which points around the current population are examined, is continuously changing. Thus, even if the population is converged to a single point, which is locally optimal according to most neighbourhood structures, eventually a rule may evolve for which the neighbourhood of that point contains a fitter solution. This can be thought of as continually testing new search landscapes to look for "escape routes" from local optima.

Looking back to the results for the GRand algorithm, in which the rules defining neighbourhoods are created at random, this "changing landscape" effect is noticeable in the superior success rates to the SMA. The fact that the SComa algorithm is the best performer according to both Success Rate and MBF metrics points to both modes of operation having a positive effect.

8 Discussion and Conclusions

As can be seen from the results section above, the S-Coma algorithm provides better performance according to Success Rate and Mean Best Fitness metrics than the GA, MA or a comparable system with the rule-learning turned off (SRand, GRand). These results are especially noticeable for the longer instances where the COMA system is able to learn and then exploit regularities within energetically favourable conformations, corresponding to secondary structural motifs. This happens at some expense of speed - the AES results show that the addition of any local search to a GA slows down the rate of discovery of globally optimal solutions, and that searching the whole neighbourhood (steepest ascent) rather than stopping once a better neighbour is found (greedy ascent) also imposes a cost. Nevertheless it must be emphasised that the results for the GA and the greedy algorithms come from many fewer successful runs. In other words, when the genetic search is able to find the optimum, it does so quickly, but it is prone to premature convergence.

Restricting the crossover and mutation operators to producing feasible solutions has mixed results. The Success Rate and Mean Best Fitness are much improved for the GA and SMA, and for the SComa on the shorter problems but if anything is slightly worse for SComa on the long instances. It was suggested in the previous section that the SComa had two modes of operation, re-use of secondary structural motifs, and continuously changing neighbourhoods. These results suggests that possibly the former mode is enhanced by the restriction to feasible solutions, but that the latter, which permits escape from local optima on the longer instances, is inhibited. Clearly this warrants further attention. Considering the efficiency with which solutions are found,

this is not significantly changed for the GA or SComa, but is much worse for the SMA algorithm.

There is a clear place for the use of expert knowledge in the design of search algorithms, and its encapsulation in the form of carefully designed move operators. Nevertheless the approach outlined in this paper represents a highly promising prospect given its ability to discover and *explicitly represent* structural motifs. As an example, the reliability results reported above are better, especially for the longer instances, than those reported elsewhere using a self-adaptive multi-memetic algorithm, with the meme set especially designed after a comprehensive study of the literature and extensive experimentation [24]. This suggests that there is a clear role for adaptation of some kind within the specification of memes, rather than using a fixed set. The results presented here and elsewhere suggest that evolution may well be a suitable way of achieving that adaptation.

One obvious path for future work would be to examine the effects of seeding the rule population with expert-designed rules. Another, perhaps more pressing path is to examine the behaviour on more complex lattices and for different energy functions. As indicated above, these results are only the beginning of a process of investigation, clearly more analysis of the evolving rule-sets is needed, as well as a thorough investigation of the other algorithmic possibilities. It seems likely however that this represents a promising direction for the future development of scalable optimisation techniques which may yield new insights into the energy landscapes of the HP and other lattice models of proteins.

9 Acknowledgements

The author would like to thank Natalio Krasnogor for many fruitful discussions during the initial stages of this work, and for introducing him to the Protein Structure Prediction problem.

References

1. ., editor. *2003 Congress on Evolutionary Computation (CEC'2003).* IEEE Press, Piscataway, NJ, 2003.
2. Thomas Bäck. Self adaptation in genetic algorithms. In F.J. Varela and P. Bourgine, editors, *Toward a Practice of Autonomous Systems: Proceedings of the 1st European Conference on Artificial Life*, pages 263–271. The MIT Press, Cambridge, MA, 1992.
3. W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela, and R.E. Smith, editors. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-1999).* Morgan Kaufmann, 1999.
4. B. Berger and T. Leight. Protein folding in the hydrophobic-hydrophilic (hp) model is NP-complete. In *Proc. 2nd Annual Intnl. Conf. Computational Molecular Biology RECOMB98*, 1998.

5. Larry Bull. *Artificial Symbiology.* PhD thesis, University of the West of England, 1995.
6. Larry Bull. Evolutionary computing in multi agent environments: Partners. In Th. Bäck, editor, *Proceedings of the 7th International Conference on Genetic Algorithms*, pages 370–377. Morgan Kaufmann, San Francisco, 1997.
7. Lawrence Bull and Terence C. Fogarty. Horizontal gene transfer in endosymbiosis. In Christopher G. Langton and Katsunori Shimohara, editors, *Proceedings of the 5th International Workshop on Artificial Life : Synthesis and Simulation of Living Systems (ALIFE-96)*, pages 77–84, Cambridge, May 16–18 1997. MIT Press.
8. K. Dill. *Biochemistry*, 24:1501, 1985.
9. S. Duarte-Flores and J.E. Smith. Study of fitness landscapes for the HP model of Protein Structure Prediction. In . [1], page to appear.
10. M. Feig, P. Rotkiewicz, A. Kolinski, J. Skolnick, and C. Brooks. Accurate reconstruction of all-atom protein representations from side-chain-based low-resolution models. *Proteins:Structure Fucntion and Genetics*, 41:86–97, 2000.
11. David B. Fogel. *Evolving Artificial Intelligence.* PhD thesis, University if California, 1992.
12. J.J. Merelo Guervos, P. Adamidis, H.-G. Beyer, J.-L. Fernandez-Villacanas, and H.-P. Schwefel, editors. *Proceedings of the 7th Conference on Parallel Problem Solving from Nature*, number 2439 in Lecture Notes in Computer Science. Springer, Berlin, 2002.
13. P. Hansen and N. Mladenovič. An introduction to variable neighborhood search. In S. Voß, S. Martello, I. H. Osman, and C. Roucairol, editors, *Meta-Heuristics: Advances and trends in local search paradigms for optimization. Proceedings of MIC 97 Conference.* Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998.
14. W. E. Hart. *Adaptive Global Optimization with Local Search.* PhD thesis, University of California, San Diego, 1994.
15. Terry Jones. *Evolutionary Algorithms, Fitness Landscapes and Search.* PhD thesis, The University of New Mexico, Albuquerque, NM, 1995.
16. Hillol Kargupta and Samiran Ghosh. Towards machine learning through genetic code-like transformations. Technical Report TR-CS-01-10, Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, 2001.
17. S.A. Kauffman. *Origins of Order: Self-Organization and Selection in Evolution.* Oxford University Press, New York, NY, 1993.
18. Robert E. Keller and Wolfgang Banzhaf. Genetic programming using genotype-phenotype mapping from linear genomes into linear phenotypes. In J.R. Koza, D.E. Goldberg, D.B. Fogel, and R.L. Riolo, editors, *Proceedings of the 1st Annual Conference on Genetic Programming*, pages 116–122. MIT Press, 1996.
19. Robert E. Keller and Wolfgang Banzhaf. The evolution of genetic code in genetic programming. In Banzhaf et al. [3], pages 1077–1082.
20. Joshua Knowles and David Corne. A comparative assessment of memetic, evolutionary and constructive algorithms for the multi-objective D-MSAT problem. In *Gecco-2001 Workshop Program*, pages 162–167, 2001.
21. A. Kolinski and J. Skolnick. Assembly of protein structure from sparse experimental data: An efficient Monte-Carlo method. *Proteins: Structure Function and Genetics*, 32:475–494, 1998.

22. N. Krasnogor and S. Gustafson. Toward truly "memetic" memetic algorithms: discussion and proofs of concept. In David Corne, Gary Fogel, William Hart, Joshua Knowles, Natalio Krasnogor, Rajkumar Roy, Jim Smith, and Ashutosh Tiwari, editors, *Advances in Nature-Inspired Computation: The PPSN VII Workshops*, pages 9–10, Reading, UK, 2002. PEDAL (Parallel, Emergent & Distributed Architectures Lab), University of Reading.
23. N. Krasnogor, W. Hart, J.E. Smith, and D. Pelta. Protein structure prediction with evolutionary algorithms. In Banzhaf et al. [3], pages 1596–1601.
24. N. Krasnogor and J. Smith. A memetic algorithm with self-adaptive local search: TSP as a case study. In D. Whitley, D. Goldberg, E. Cantu-Paz, L. Spector, I. Parmee, and H.-G. Beyer, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, pages 987–994. Morgan Kaufmann, 2000.
25. Natalio Krasnogor. *Studies in the Theory and Design Space of Memetic Algorithms.* PhD thesis, University of the West of England, 2002.
26. Natalio Krasnogor and Jim Smith. Emergence of profitiable search strategies based on a simple inheritance mechanism. In L. Spector, E. Goodman, A. Wu, W.B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. Garzon, and E. Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 432–439. Morgan Kaufmann, 2001.
27. Peter Merz. *Memetic Algorithms for Combinatorial Optimization Problems: Fitness Landscapes and Efective Search Strategies.* PhD thesis, Department of Electrical Engineering and Computer Science, University of Siegen, Germany, 2000.
28. Peter Merz and Bernd Freisleben. Fitness landscapes and memetic algorithm design. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 245–260. McGraw Hill, 1999.
29. Pablo Moscato. Memetic algorithms' home page. Technical report, http://www.densis.fee.unicamp.br/~moscato/memetic_home.html, 2002.
30. N. Krasnogor, B.P. Blackburne, E.K. Burke and J. D. Hirst. Multimeme algorithms for protein structure prediction. In Guervos et al. [12], pages 769 –778.
31. Jan Paredis. The symbiotic evolution of solutions and their representations. In L.J. Eshelman, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 359–365. Morgan Kaufmann, San Francisco, 1995.
32. Jan Paredis. Coevolutionary algorithms. In T. Bäck, D. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation.* Institute of Physics Publishing, Bristol, and Oxford University Press, New York, 1998.
33. M. A. Potter and K.A. DeJong. A cooperative coevolutionary approach to function optimisation. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Proceedings of the 3rd Conference on Parallel Problem Solving from Nature*, number 866 in Lecture Notes in Computer Science, pages 248–257. Springer, Berlin, 1994.
34. J.David Schaffer and Amy Morishima. An adaptive crossover distribution mechanism for genetic algorithms. In J.J. Grefenstette, editor, *Proceedings of the 2nd International Conference on Genetic Algorithms and Their Applications*, pages 36–40. Lawrence Erlbaum Associates, 1987.
35. H.-P. Schwefel. *Numerical Optimisation of Computer Models.* John Wiley and Sons, New York, 1981.
36. J.E. Smith. Co-evolving memetic algorithms: A learning approach to robust scalable optimisation. In . [1], page to appear.

37. Jim Smith. Co-evolution of memetic algorithms : Initial investigations. In Guervos et al. [12], pages 537–548.
38. Jim Smith and T.C. Fogarty. Adaptively parameterised evolutionary systems: Self adaptive recombination and mutation in a genetic algorithm. In Voigt et al. [43], pages 441–450.
39. Jim Smith and T.C. Fogarty. Self adaptation of mutation rates in a steady state genetic algorithm. In *Proceedings of the 1996 IEEE Conference on Evolutionary Computation*, pages 318–323. IEEE Press, Piscataway, NJ, 1996.
40. R. Unger and J. Moult. Genetic algorithms for protein folding simulations. *Journal of Theoretical Biology*, 231(1):75–81, 1993.
41. Ron Unger and John Moult. A genetic algorithm for 3D Protein Folding Simulations. In S. Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 581–588. Morgan Kaufmann, San Francisco, 1993.
42. F. Vavak, T.C Fogarty, and K. Jukes. A genetic algorithm with variable range of local search for tracking changing environments. In Voigt et al. [43], pages 376–385.
43. H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors. *Proceedings of the 4th Conference on Parallel Problem Solving from Nature*, number 1141 in Lecture Notes in Computer Science. Springer, Berlin, 1996.

Hybrid Evolutionary Approaches to Terminal Assignment in Communications Networks

X. Yao[1], F. Wang[2], K. Padmanabhan[1] and S. Salcedo-Sanz[1]

[1] The Centre of Excellence for Research in Computational Intelligence and Applications (CERCIA), School of Computer Science, The University of Birmingham Edgbaston, Birmingham B15 2TT, UK. Email: x.yao@cs.bham.ac.uk

[2] Intelligent Systems Lab, BTExact Orion 1/12, Adastral Park, Ipswich, IP5 3RE, UK. Email: fang.wang@bt.com

Summary

Terminal assignment is an NP-hard problem in communications networks. It involves assigning a set of terminals to a set of concentrators with a cost for each assignment. The objective is to minimize the total cost of the assignment and the number of concentrators used. A number of heuristic algorithms, including genetic algorithms, have been proposed for solving this problem. This chapter studies several evolutionary and hybrid approaches to terminal assignment. Firstly, a novel chromosome representation scheme based on concentrators is proposed. This representation compares favourably against the existing terminal-based representation, which scales poorly for large problems. Extensive experiments have been carried out. The results show that our evolutionary algorithms using the concentrator-based representation outperform significantly existing genetic algorithms using the terminal-based representation. Secondly, a number of new search operators used in our algorithms are also investigated empirically in order to evaluate their effectiveness for the terminal assignment problem. Finally, different combinations of evolutionary algorithms and local search are studied in this chapter. Both Lamarckian evolution and Baldwin effect have been examined in combining an evolutionary algorithm and local search. Our results show that hybrid algorithms perform better than either evolutionary algorithms or local search. However, there is no significant difference between Lamarckian-evolution-style combination and Baldwin-effect-style combination.

1 Introduction

Evolutionary algorithms (EAs) and their hybridisation with local search have been widely studied and applied to solve many real world problems. Communications network design is a typical combinatorial optimization problem for which no efficient algorithm exists unless P=NP. A good design of communications networks requires certain constraints to be met and at the same time, one or more objectives to be optimized. The algorithms for designing communications networks must have good scalability and be able to deal with large-scale applications with a large number of network nodes.

The optimal design of communications networks considering both cost and capacity has been investigated in the literature using different heuristic algorithms, such as tabu search, simulated annealing and greedy search. Recently, EAs have been shown to perform well in communications network design, especially for the terminal assignment problem, which has been shown to be NP hard [11]. However, the performance of such EAs is still unsatisfactory for large problems.

This chapter studies novel hybrid EAs. Unlike previous EAs which used a terminal-based chromosome representation, a concentrator-based evolutionary approach for solving the terminal assignment problem is proposed in this chapter. This evolutionary approach uses a novel concentrator-based representation and associated search operators. It is hybridised with local search methods to form hybrid EAs. The concentrator-based representation is proposed to overcome the difficulties encountered by the terminal-based representation previously used by other evolutionary approaches. Attempts are also made to design appropriate search operators that work well with the concentrator-based representation. In addition to minimising costs, we also consider reducing the number of concentrators used. The objective of minimising the total cost is explicitly dealt with by the fitness function during the evolution. Minimising the total number of concentrators used is considered as an implicit constraint for the cost objective, or a second objective encoded in the fitness function. Hence, there are two problem formulations for the terminal assignment problem, i.e., single-objective and multi-objective optimisation. In this chapter, both formulations will be studied using the hybrid EAs and the concentrator-based representation.

In our hybrid EAs, two methods are considered for hybridization with local search, i.e., Lamarckian evolution and Baldwin effect. Lamarckian evolution forces the genotype to reflect the result of local improvement. The improved individual is placed back into the population and allowed to compete for reproductive opportunities [10]. The Baldwin Effect allows an individual's fitness (phenotype) to be determined after local search. Similar to natural evolution (Darwinian evolution), the result of the improvement is not reflected in the genetic structure (genotype) of the individual. It only changes the individual's chance of survival [10]. Baldwin effect as used in EAs may introduce undesirable offspring after crossover. When crossing two individuals, which after

local search converge to the same local basin, it is likely that the offspring may be similar to the parents and will converge to the same basin. To avoid this problem, the use of memory is considered for both Lamarckian evolution and Baldwin effect in the work presented here.

The concentrator-based evolutionary approach and its hybrid evolution are fully tested and examined by a series of computational experiments designed for the terminal assignment problem. The results have shown that the EA's performance was better with concentrator-based representation than with the terminal-based representation. The generation of a feasible initial population is simpler and more scalable in the concentrator-based representation even for a large number of terminals. The concentrator-based hybrid EAs outperformed EAs without local search. However, there is no significant difference between two different approaches to hybridise EAs with local search, i.e., Lamarckian evolution and Baldwin effect.

The remainder of this chapter is organised as follows. The next section introduces the terminal assignment problem and the previous work in solving this problem. Section 3 presents our concentrator-based representation and the search operators designed for it. A set of experiments are carried out to test the performance of the representation and operators and to compare them with the traditional terminal-based EAs. Concentrator-based hybrid EAs that integrate Lamarckian evolution or Baldwin effect are studied in Section 4. Lamarckian-style and Baldwin-style evolution with and without memory are investigated using the terminal assignment problems with single-objective or multi-objectives. Section 5 concludes this chapter with a brief summary of our work and some future work.

2 The Terminal Assignment Problem

2.1 Problem Representation

In this chapter, we will focus on the two-terminal network (also called source-link network) design. The work, however, can also contribute to the design of other kinds of networks, i.e., all-terminal networks. In the two-terminal network, a set of pre-specified source nodes communicate with the pre-specified sink nodes through non-specified paths. This can be simplified as a terminal assignment problem that concerns the assignment of certain terminals to some concentrators. This assignment should keep the total cost minimum. The cost may include material cost of cabling, installation cost and connection or communication cost between the concentrators and terminals. The cost may be fixed or varied per connection depending on the real situation. In general, it can be summarized as a weight that is used as the complete cost for each connection [2], [3], and [14].

In addition to minimizing the total cost, the terminal assignment problem should take the concentrators' capacity limit into account by satisfying two constraints:

1. Every terminal is assigned to one and only one concentrator.
2. The sum of weights of connections between terminals and a concentrator should not exceed the capacity of that concentrator.

Single-objective Optimisation Formulation

Given

K: number of concentrators,
T: number of terminals,
C_i: Capacity of concentrator $i = 1, 2, \ldots, K$,
d_ij: weight of the connection between concentrator i and terminal j, where $i = 1, 2, \ldots, K$, $j = 1, 2, \ldots T$,

the single objective optimisation problem of terminal assignment is to minimise the total cost,

$$\sum_{i=1}^{K} \sum_{j \in J} d_{ij}$$

subject to

$$\sum_{j \in J_i} d_{ij} \leq C_i,$$

where $j \in J_i$ is the terminal j assigned to concentrator i, and is the set of all terminals connected to concentrator i.

Multi-objective Optimisation Formulation

The most common objective of the terminal assignment problem is to minimise the total cost of the network. However, in many situations, it makes sense to also minimise the number of concentrators used so that the whole network can work with a less cost. The minimisation of the number of concentrators can be treated as an implicit constraint to be considered in the above single objective optimisation, or as another objective to optimise. In the latter case, the problem becomes a true multi-objective optimisation problem that minimises both the cost and the number of concentrators used at the same time. A weighted sum approach for this problem is described below.

Given

f_i: objective i, $i = 1, 2, \ldots, n$, and

w_i: weight of objective,

then the purpose of the multi-objective optimisation is to minimise

$$G = \sum_{i=1}^{n} w_i f_i.$$

In the problem presented here, n is 2 (for two objectives), f_1 is the total cost of all the connections between concentrators and terminals, which is F as described in the single objective optimisation; and f_2 is the total number of concentrators used.

2.2 Previous Work on Terminal Assignment in Communications Networks

Various approaches have been applied to the optimisation of communications networks. Previous work in [3] utilised simulated annealing to find the optimal design of small-scale networks (less than five nodes). Simulated annealing was also adopted in [15] to find solutions for packet switched networks with considerations of delay and capacity. Tabu search was used in [7] and in [13] to find an appropriate design of communications networks by considering cost and capacity together.

Using greedy algorithms and genetic algorithms (GAs) to assign terminal nodes to concentrators was investigated by [1]. The greedy algorithm assigns terminals to nearby (but maybe not the nearest) concentrators, if this assignment can help other terminals to be assigned to nearby concentrators. This kind of assignment can lead to infeasible solutions even if a feasible solution exists. This means that sometimes there are unassigned terminals that cannot be allocated to any concentrator.

The GA used in [1] had two possible chromosome representations for the terminal assignment problem, LC1 and LC2. Both representations are composed of an integer string. Each integer indicates the concentrator to which a terminal is assigned. The integers are arranged in the sequence of terminals, so the length of the string is the same as the number of terminals. In LC1, the first n1 terminals are assigned to n2 different concentrators, one terminal per concentrator. The remaining terminals are assigned in a greedy fashion considering the different costs of the concentrators. [1], used a seeding strategy to initialise the population in order to reduce the number of infeasible individuals in the initial population. Unfortunately, this kind of representation sometimes may cause inappropriate assignments with a great waste of concentrator capacities after the first n1 terminals are allocated to n2 concentrators. In case of large-scale problems with large numbers of terminals and concentrators, the computation time may increase considerably due to the continuous evaluation of the lowest costs for the assignment of the remaining terminals.

The second representation LC2 do not adopt the strategy of assigning the first n1 terminals to n2 concentrators. All the terminals are assigned in a greedy fashion. Therefore, unlike LC1, the infeasibility in the initial population of LC2 is likely to be high. In case of large-scale problems, the computation of the cost can be very high as well. The results in [1] showed that GAs outperformed the greedy algorithm.

[12] compared greedy algorithms, GAs and grouping GAs (GGAs) for solving the terminal assignment problem. A terminal is assigned to the nearest concentrator which has sufficient capacity to take this terminal, and if not, the next closest concentrator is chosen for assignment. In this algorithm, there are many chances that some terminals may not be allocated and hence make the solutions infeasible. If the number of terminals or concentrators is large, it may take a long time to search for concentrators with the least cost for assigning all terminals.

The GAs in [12] used both binary and non-binary representations indicating the concentrators with which the terminals are connected. If the terminal size is large, e.g., 1000, the chromosome length will also be large, e.g., 1000. Therefore, generating a feasible initial population and evolving such long chromosomes can be a challenge to GAs. [12] have incorporated a penalty term in the fitness function to deal with infeasibility. Infeasible solutions are not discarded but included in the population with the penalty incorporated in it. The penalty term clearly distinguishes infeasible solutions from feasible ones. A higher penalty imposes more selective pressure on infeasible solutions.

[12] used a GGA as a third approach to solve the terminal assignment problem. The representation in GGAs consists of two parts. The first part is the same as the representation used in GAs, but there is an additional part which groups the terminals and their connected concentrators together. The first part of the representation is only used for selection and fitness evaluation. A special crossover operator is designed for the group part in the group representation, which selects an entire group from one parent and inserts it into the other parent at the crossover point. After crossover there is a high possibility that the individuals may become infeasible. So the infeasible chromosomes have to be repaired. The repair process needs to remove the duplicate concentrators and re-assign the associated terminals to other concentrators. [12] demonstrated that GAs with the non-binary representation outperformed greedy algorithms in most cases, but GGAs did not perform very well comparatively.

It can be seen from previous work that using evolutionary approaches (especially GAs) in communications network design has potentials. These approaches showed better performance than other search algorithms such as greedy algorithms. However, there is a crucial limitation in the previous evolutionary approaches concerning their encoding methods, which is usually a list of all possible connections to concentrators, arranged in the sequence of terminals. Such encoding methods usually cannot work well with large-scale problems, and in particular, they have extreme difficulties in generating a feasible initial population within a reasonable time. Because those encoding schemes are all based on terminals and cannot reflect well the relationship between terminals and concentrators, good couplings between terminals and concentrators discovered in evolution may not be maintained after search operations such as crossover and mutation. This makes the evolution more difficult to find and keep optimal solutions. Though [12] introduced the concept

of group based representation, it was used together with the terminal based representation and only for crossover. The performance of GGAs was not satisfactory. In order to overcome the difficulties presented in the previous work, a new chromosome representation is proposed for the terminal assignment problem in the next section.

3 Concentrator-based Evolutionary Approach

In this chapter, a novel concentrator-based evolutionary approach is proposed to make use of the group structure in the terminal assignment problem. This approach is especially used to overcome the incapability of previous evolutionary approaches in handling large-scale networks. The concentrator-based evolutionary approach differs from classic EAs in two aspects. First, a special encoding scheme is designed to introduce the structure of groups into the genes of chromosomes. Second, given the distinctive encoding, special genetic operators are designed to evolve the concentrator-based chromosomes for solving the terminal assignment problem.

In the remaining of this section, we will introduce the encoding method and the corresponding search operators. The concentrator-based EA is then examined and compared with other EAs by a series of experiments with different experimental settings.

3.1 Concentrator-based Representation

The concentrator-based representation is composed of a set of trees in one level, in each of which the concentrator is the root node and the terminals associated with the concentrator are the leaves. Each tree therefore indicates a concentrator together with its terminals. An example of the representation is shown in Figure 1,

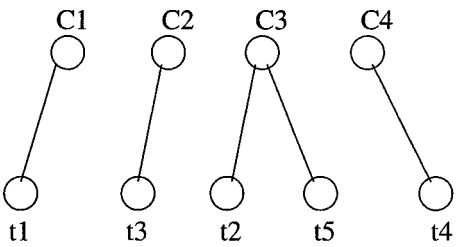


Fig. 1. An example of the concentrator-based representation.

In this example, there are 4 concentrators (c1 to c4) and 5 terminals (t1 to t5). Terminal t1 is assigned to concentrator c1, terminal t3 to concentrator c2, and so forth. The representation of this example can be written as:

$$c1(t1)c2(t3)c3(t2,t5)c4(t4)$$

When composing the concentrator-based representation, both constraints of the terminal assignment problem must be met. Any infeasible representations should be either repaired or eliminated in evolution.

The initial population of the concentrator-based EAs is generated in a way similar to that of the terminal-based EA (e.g., the non-binary representation used in [12]). Every concentrator has equal probability to serve terminals. A terminal is first assigned to a randomly selected concentrator. If the concentrator has not enough capacity to serve the terminal, then another concentrator is randomly chosen. An individual is included in the population only if it is feasible.

The concentrator-based representation, i.e. the tree-based representation, allows for variable length genotypes, so the chromosomes are not restrained by terminal or concentrator numbers. It is both efficient and flexible. Because there is no need to search and evaluate the least cost concentrators when generating individuals, the concentrator-based representation works well even with a large number of terminals or concentrators. The generation of the initial population is simpler than the terminal-based representation. The terminals that are to be assigned to a concentrator are taken from a pool where terminals are stored, eliminating any duplicates. By generating populations in this way the constraints of assigning a terminal to only one concentrator is implicitly satisfied.

3.2 Search operators

A series of search operators including selection, crossover, and mutation have been designed to work with the new concentrator-based representation, as neither the standard nor the ordering genetic operators are suitable for grouping problems [4]. These operators are introduced below.

Selection

Selection is the operation by which individuals are selected from a population for mating. There are many different models of selection such as ranking, roulette wheel selection and tournament selection. Because these models select chromosomes according to their ranks or fitness values, they can be easily applied to the concentrator-based evolution without major changes. In the following experiments, tournament selection is used due to its good performance in selecting optimum or nearly optimum solutions.

Crossover

The purpose of crossover is to pass on the genetic material from the current generation to the next one. A typical crossover recombines two individual parents to produce two offspring. Several crossover operators can be used on the concentrator-based representation.

One Point Crossover

This is one of the most common crossover methods used in EAs. A crossover point is randomly chosen and children are obtained by swapping the tails of the parents' chromosomes. Figure 2 is an example of how one point crossover works on the concentrator-based representation. If the crossover point divides the parent in equal halves then equal information is inherited. Sometimes repair has to be done to make the children feasible. The process of repair is explained later. In this type of crossover, the order of concentrators in a chromosome is not very important.

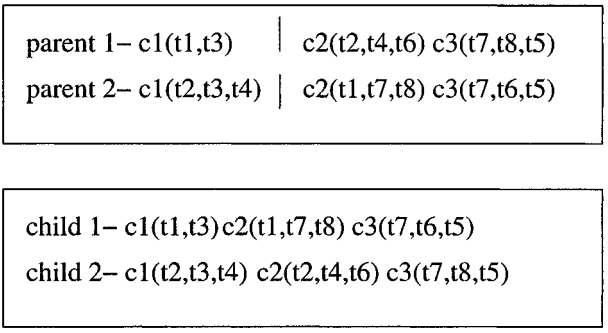


Fig. 2. One point crossover.

Two Point Crossover

In two point crossover, two crossover points are randomly chosen and the chromosome parts in between are exchanged between the parents, as shown in Figure 3. The information that is inherited depends on the crossover points. If the crossover points are far apart, more information is then inherited. Similar to one point crossover, this type of crossover is also commonly used in EAs.

Modified Uniform Crossover

A typical example of the uniform crossover is shown in Figure 4. The order of concentrators remains the same in all the chromosomes before and after crossover, but only some the terminals associated with each concentrator are inherited.

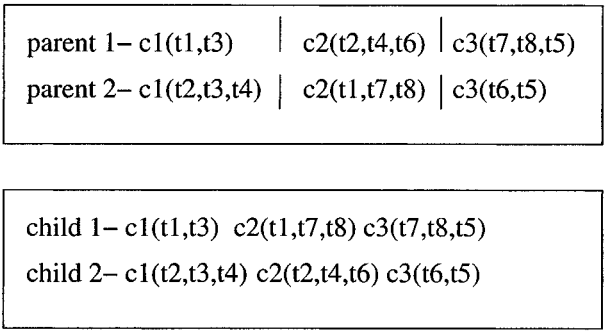


Fig. 3. Two point crossover.

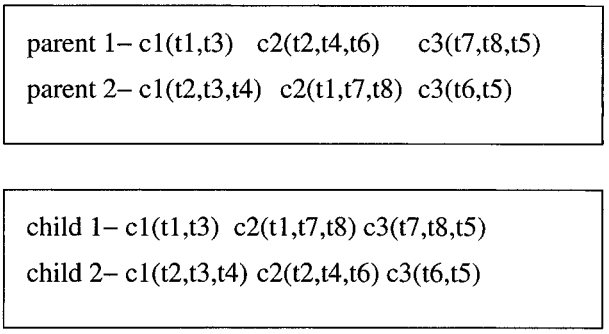


Fig. 4. Uniform crossover.

In Figure 4, for child1, the terminal set of c1 is inherited from parent1, for c2 it is inherited from parent2 and for c3 it is again inherited from parent1. In this example, the probability of inheriting a gene from a parent is set as 0.5. In the following experiments, the probability is calculated based on the available capacities of the concentrators. For example, if the available capacity of c1 is 60% in parent1 and is 30% in parent2, then the probability of selecting c1 from parent1 will be greater than from parent2. Such a uniform crossover is different from the classical one, and thus called modified uniform crossover. In this type of crossover it is possible that both terminal assignment constraints may be violated. The crossover may result in infeasible solutions. For example, terminals may be assigned to more than one concentrator, such as terminal t4 of child2 shown in Figure 4, which is assigned to both c1 and c2. Also, there may be some terminals that are not assigned to any concentrator, such as terminals t1, t7 and t8 in child2. In order to resolve the violation of constraints, repair should be done.

One node Crossover

The crossover operators introduced above are similar to classical crossover methods on the terminal-based representation. To exploit our representation better, two specific crossover methods based on concentrators are also designed. Figure 5 illustrates one of the methods, which is called one node crossover. Each concentrator is deemed as a node in this method. A random node point is chosen (such as c2 in the example shown in Figure 5) and the nodes together with their associated terminals in two parents are swapped to produce offspring. Repair is used to make the offspring feasible whenever necessary.

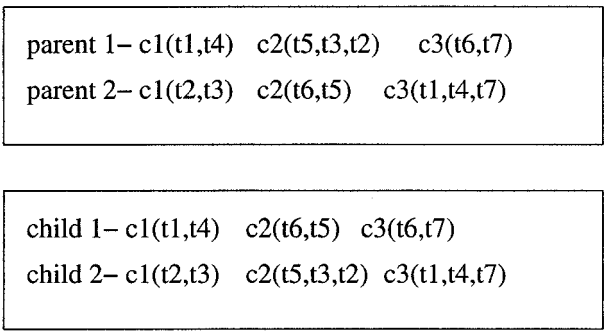


Fig. 5. One node crossover.

Best Node Crossover

In addition to one node crossover, another node-based crossover is proposed to exploit the best concentrator in the chromosomes. The best concentrator is chosen from each parent and then passed to the offspring. In Figure 6, concentrator c3 from parent1 and concentrator c1 from parent2 are transferred to both children. The offspring replace their parents only if the cost is less than or equivalent to that of the parents. This type of crossover can reduce the number of concentrators used.

One Group Crossover

This operator is inspired by the crossover used in GGAs as described in Section 2.2. Two random crossover points are generated separately and independently for two parents as shown in Figure 7. The crossover points may be different for two parents. Repair will be needed for infeasible children.

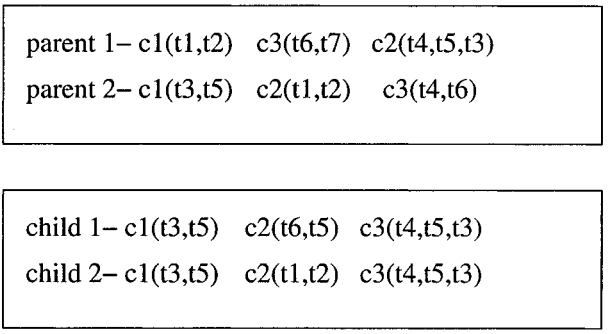


Fig. 6. Best node crossover.

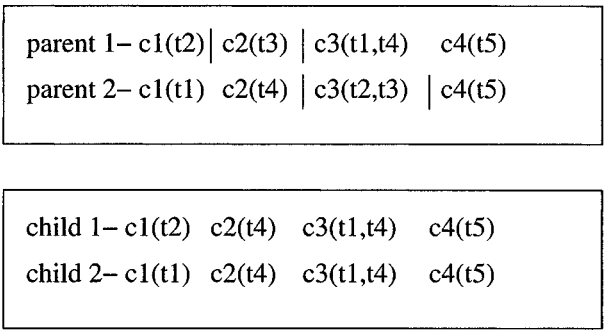


Fig. 7. One group crossover.

Best Group Crossover

This crossover is similar to the above one group crossover, but the best concentrator in the group will be retained in the offspring instead of being replaced and lost through crossover. Figure 8 illustrates this crossover. The concentrator c2 in parent1 is best utilized and hence is retained in child1. The concentrator c3 in parent1 is the best and hence is inserted into the chromosome of child2.

Repair

After crossover some terminals may be either presented in duplicates or completely missing and hence cause infeasible individuals. Stochastic repair is then used to make the individuals feasible. The repair process can be described by two steps:

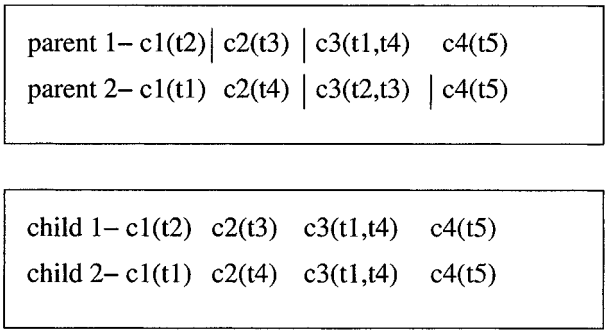


Fig. 8. Best group crossover.

1. Deletion of duplicate terminals - Each terminal is examined for duplicates and if there is any, a duplicate terminal in a less loaded concentrator is deleted;

2. Stochastic assignment of missing terminals - Missing terminals are assigned to less loaded concentrators which are randomly chosen.

Mutation

Mutation makes (usually small) alterations to one or more genes in a chromosome. It is considered as a method to recover lost genetic material during evolution. Here several mutation methods are used for the concentrator-based evolution.

Point Concentrator Swap

Two concentrators c1 and c2 are chosen stochastically and all the terminals associated are swapped between them. Because concentrators may have different capacities, swapping their terminals may reduce the cost but not the number of concentrators. This is shown in Figure 9.

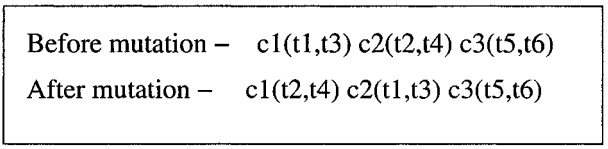


Fig. 9. Two point concentrator swap.

Two Point Terminal Swap

Here two concentrators are chosen at random and then two terminals are randomly chosen from the two concentrators, respectively. The selected terminals are then interchanged. An example of this mutation is shown in Figure 10.

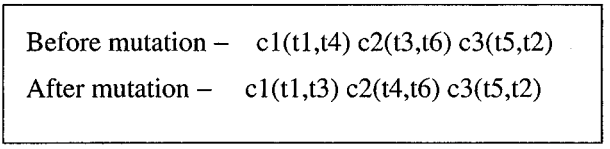


Fig. 10. Two point terminal swap.

In Figure 10, concentrators c1 and c2 are selected and in them terminals t3 and t4 are selected and then interchanged. The mutated individual is included in the population only if it is feasible and fitter than its parent.

Delete-Insert One Mutation

This mutation is designed to alter the concentrator of a terminal. A concentrator c1 is first chosen at random. Then a random terminal t1 is deleted from it and then inserted into the terminal set of another randomly chosen concentrator, c3 in this case. The mutated individual joins the population only if it is fitter than its parent and is feasible. This type of mutation is designed to reduce the number of concentrators used. For example, c1 in Figure 11 is no longer needed.

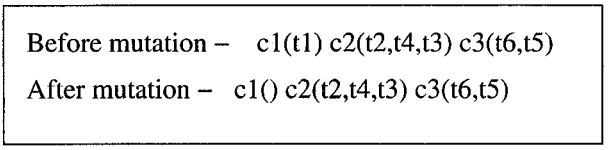


Fig. 11. Delete-insert one mutation.

In this mutation, a concentrator c1 is chosen at random and all the terminals in c1 are removed and inserted into another concentrator c2, which is also chosen at random, as shown in Figure 12. This type of mutation is designed to reduce the number of concentrators by shifting all of the terminals of a concentrator to other concentrators. In Figure 12, all the terminals of c1 are shifted to concentrator c2, however they may be reassigned to more than

one concentrator if the currently selected concentrator does not have sufficient capacity. After this mutation the individual joins the population only if it is fitter than its parent and is feasible.

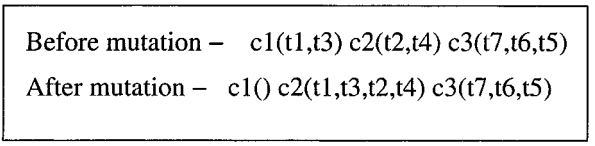


Fig. 12. Delete-insert all mutation.

Self Crossover Mutation

Two concentrators c1 and c2 are chosen at random. The terminal set of each concentrator is regarded as a small "individual" and the two sets are crossed using one point crossover.

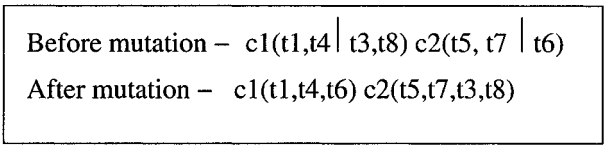


Fig. 13. Self crossover mutation.

One Group Mutation

A less loaded concentrator is chosen at random from a parent and all the terminals associated with the concentrator are deleted. The deleted terminals are then reassigned to other concentrators chosen randomly. In Figure 14 concentrator c2 is chosen at random and terminal t3 is deleted. The missing terminal t3 is added to the terminal list of concentrator c1. This type of mutation may result in fewer concentrators.

Multi-Group Mutation

This mutation is similar to the one group mutation except that more than one concentrator is involved. Several less loaded concentrators are chosen at random. In Figure 15 concentrators c2 and c4 are randomly chosen and terminals t4 and t2 are removed from their terminal lists. The removed terminals t4 and t2 are reassigned to a random concentrator c3. This type of mutation operator can also reduce the number of concentrators used.

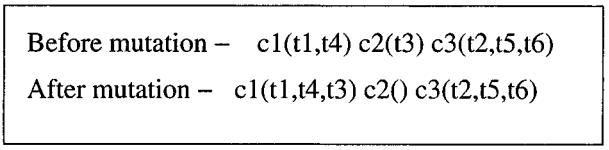


Fig. 14. One group mutation.

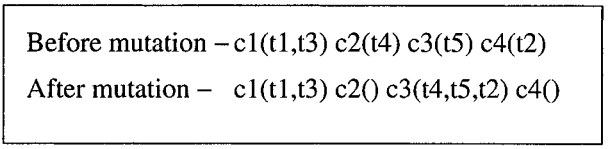


Fig. 15. Two group mutation.

3.3 Experimental Studies

In the previous sections, we introduced the concentrator-based representation and a number of search operators that can be used on the representation. In order to evaluate the proposed concentrator-based EA, a number of experiments were run with different experimental settings. In this section, both the concentrator-based representation and the corresponding operators will be tested for their performance.

Performance Test of the Concentrator-based Representation

The first experiment is used to examine the performance of the proposed concentrator-based representation. For the purpose of comparison, the terminal-based representation was also tested in the experiments. The initial populations in the experiments were generated in the way as described in Section 3.1. Tournament selection with uniform crossover and two-point interchange mutation was used for both representations in this comparison test. The EAs are terminated after the fitness value remains unchanged for 25 generations. Both concentrator-based and terminal-based representations are tested on different problems in which the number of terminals ranges from 100 to 1000. However, problems with more than 500 terminals were not considered in the terminal-based representation because the generation of their initial populations took too long. Table 1 gives a list of the experimental parameters used in the experiment. The experiment was run for 30 times and the results are shown in Table 2.

The results in Table 2 show that the concentrator-based representation generally found solutions better and much faster (with fewer generations) than the terminal-based representation, especially when the problem was large.

Table 1. Experimental setting

Population size:	100
Chromosome:	terminal based/concentrator based representation
Selection:	tournament selection
Crossover:	uniform crossover
Mutation:	two-point interchange
Termination Criterion:	the fitness value presents no change for 25 generations
Elitism:	yes
Number of runs:	30
Ratio of number of terminals to terminal number of concentrators:	2:1
Number of terminals:	100 to 500/100 to 1000
Number of concentrators:	50 to 250/50 to 500
Weight of terminals to: concentrators	1 to 6
Capacity of concentrators:	15 to 25

Table 2. Comparison between concentrator-based and terminal-based representations, where Size indicates the number of terminals, s.d. indicates standard deviations and N indicates the number of generations.

	Concentrator-based					Terminal-based				
Size	cost				N	cost				N
	best	mean	s.d.	worst		best	mean	s.d.	worst	
100	1254	1479	250	1720	250	932	1062	72	1212	435
200	1607	1806	116	2029	587	1565	1782	122	2051	1005
300	2018	2259	150	2568	1008	2038	2305	171	2756	1618
400	2375	2711	190	3152	1306	2676	3119	241	3843	2004
500	2962	3508	193	3843	1709	3194	3753	320	4665	2665
600	3274	3618	213	3959	2160					
700	3959	4399	246	4922	2306					
800	3920	4409	258	5029	2761					
900	4337	5179	342	6055	3224					
1000	4395	5295	378	5936	3624					

During the experiments, we found that the difference in cost between two representations was higher in the first generation as compared with the final generation. Though the concentrator-based representation produced a relatively uncompetitive population at the beginning, it obtained superior final results through evolution, except when the problem is very small, e.g., for terminal sizes 100 and 200. The concentrator-based representation achieved the results in fewer generations for all terminal sizes. The genetic operators worked more effectively on the concentrator-based representation than its counterpart.

The concentrator-based representation also showed good scalability. It easily generated and evolved populations for terminals up to 1000. This is in contrast to the terminal-based representation which became incapable of solving the problem when the number of terminals involved was more than 500. This incapability inevitably restricts the application of the terminal-based representation in real world communications networks, which usually involve a great number of network nodes. The number of generations required by the concentrator-based representation was approximately linearly increased with the increased number of terminals, as shown in Figure 16. The well-presented scalability of the concentrator-based representation shows that it is suitable for large-scale network applications.

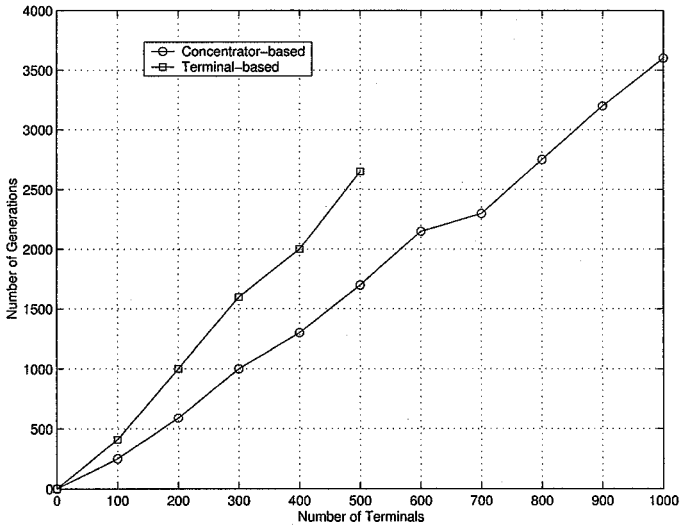


Fig. 16. Scalability of the concentrator-based representation. The generations required by the concentrator-based EA was linearly increased with the problem size. On the contrary, the terminal-based EA became incapable of solving the problem when terminals were more than 500.

Performance Tests of Search Operators

Various crossover and mutation operators designed for the concentrator-based representation were tested for their performance here. The EA guided by the operators should achieve the objective of minimizing the total cost between terminals and concentrators, and at the same time, the operators should keep the number of concentrators used at a minimum. In our experiments, the

number of terminals was set as 100 and the number of concentrators was 50. The other settings of the experiments were the same as those used in the previous tests (see Table 1). The experimental results over 30 runs are presented in Table 3.

Table 3. Performance tests of various search operators, where s.d. indicates standard deviation and N indicates the number of generations.

	Search operators	Number of conc.	Cost best	mean	worst	s.d.	N
1	One point crossover	20	4160	4567	4797	25622	50
2	Two point crossover	44	4380	4553	4728	8855	28
3	Uniform crossover	19	4498	4751	5052	23093	50
4	One node crossover	23	4145	4485	4779	23481	50
5	Best node crossover	43	4222	4503	4726	13887	50
6	One group crossover	44	4361	4563	4737	9863	25
7	Best group crossover	44	4251	4549	4720	16256	25
8	Delete-insert one mutation	39	681	702	744	259	296
9	Delete-insert all mutation	23	2877	3120	3408	13759	115
10	Two point concentrator swap	45	2479	2711	2984	19294	110
11	Two point terminal swap	50	1172	1420	1737	20588	254
12	Self crossover mutation	44	1357	1653	2171	31268	258
13	One group mutation	44	4380	4553	4728	8855	28
14	Multi-group mutation	44	4380	4553	4728	8855	28

¿From Table 3 we can see that, when the cost alone is considered, delete-insert one mutation was the best among all search operators. Two point terminal swap mutation, self crossover mutation, two point concentrator swap mutation and delete-insert all mutation followed delete-insert one mutation, but all of them required more generations. All the other operators performed similarly. When the number of concentrators used alone is considered, uniform crossover is the best, followed by one point crossover, one node crossover, delete-insert all mutation and delete-insert one mutation. The remaining operators had similar performance.

Generally speaking, crossover operators showed better performance in reducing the number of concentrators because most crossover operators are designed for this purpose. Moreover, the stochastic repair mechanism is also very effective in reducing the number of concentrators used. While the exchange of genes is more frequent (and hence more repair is required) in one point crossover and uniform crossover, only 40% of the total concentrators were finally utilized in the solutions found by these two crossover methods. Unlike one point crossover, two point crossover may not have frequent gene exchange and the repair mechanism only works on limited numbers of concentrators. The reduction of the concentrators used is therefore not so significant in two point crossover. Best node crossover is similar to one node crossover except

that the best utilized concentrator is chosen for crossover in both parents. Although repair may help reducing the concentrators used in both cases, it is unclear why the former is ineffective in the concentrator usage while the later is relatively more effective. This issue will be our future work. Compared with other crossover operators, the performance of group based crossover methods (e.g., one group crossover and best group crossover) was unsatisfactory. In most cases, group based crossover involves less exchange of genes than others. This may be the main reason why more concentrator used in group based crossover. Group based crossover also required much longer time to meet the termination criterion. All crossover operators showed insufficient effect on reducing the total cost.

In contrast to crossover, most mutation operators effectively reduced the total cost because the assignment of terminals to concentrators was continuously altered and only fitter individuals after mutation were allowed to join the population. Among these operators, delete-insert one and delete-insert all mutation performed best by maintaining better utilized concentrators and mutating less loaded concentrators. Two point concentrator swap, two point terminal swap and self crossover mutation were also good at reducing cost, but less effective than delete-insert one and delete-insert all mutation. This is because these operators do not take the concentrator load into account when swapping genes. Similar to group based crossover, one group mutation and multi-group mutation demonstrated unsatisfactory performance in both cost minimization and reduction of the number of concentrators used.

It is worth emphasizing that our study of genetic operators was carried out for each operator independently. We did not run EAs with two or more operators in the above experiments (Table 3). We expect EA's performance will improve further if we use two or more appropriate operators together.

4 Concentrator-based Hybrid Evolutionary Approaches

Hybrid EAs have been shown to be quite effective in solving a wide range of real world problems. How EAs and local search are combined is an extremely important issue that influences the final solution quality and the computational efficiency of the algorithm [10]. Hybridization of EA with local search gives rise to the concepts of Lamarckian evolution and Baldwin effect [10], which are the most often studied techniques in hybrid EAs.

In this section, both Lamarckian evolution and Baldwin effect are incorporated with the concentrator-based EA to form hybrid EAs for communications network design. Lamarckian evolution or Baldwin effect is applied to all the individuals in every generation. The two different forms of hybrid EAs are fully investigated on the terminal assignment problem, for both single-objective and multi-objective optimisation as introduced in Section 2.1. A series of experiments are designed to examine the performance of the hybrid EAs.

4.1 Lamarckian Evolution and Baldwin Effect

In Lamarckian evolution individuals improve during their lifetime through local search and the improvement is passed to the next generation. The individuals are selected based on improved fitness and are transferred to the next generation with the improvement incorporated in the genotype.

The Baldwin effect utilized in EAs was first investigated by Hinton and Nolan in [8]. Unlike Lamarckian evolution, the improvement does not change the genetic structure (genotype) of the individual that is transferred to the next generation. The individual is kept the same as before local search, but the selection is based on the improved fitness after local search. Baldwin effect follows natural evolution (Darwinian), i.e., learning improves the fitness and selection is based on fitness. The improvement is passed indirectly to the next generation through fitness in Baldwin effect.

While Lamarckian learning may disrupt the schema processing of a GA, Baldwin learning certainly aggravates the mapping problem of multiple genotypes to one phenotype. In a comparison of Baldwin and Lamarckian learning, [16] showed that utilizing either form of learning would be more effective than the classical GA without any local improvement procedure. They argued that, while Lamarckian learning is faster, it may be susceptible to premature convergence to a local optimum as compared to Baldwin learning [10].

4.2 Use of Memory

In Baldwin effect, if two individuals are different but map to the same local basin, the evolutionary approach will try to exploit both individuals. If these two individuals are crossed over and produce offspring in the same basin, computational effort will then be wasted on applying the local search to search the same basin again [10].

In Lamarckian evolution, these individuals are possibly identical and will reproduce clones of themselves if crossed over. The local improvement is therefore unnecessary as the children are the same as the parents. Slight mutation change may be useless since it may leave the individual in the same basin or in a later generation the EA may generate an individual that falls in a basin already explored. Therefore, the local improvement procedure may be reapplied to search the same basin while valuable computational cycles could be used to explore other regions in the search space. To solve this problem, random linkage, a search algorithm taken from global optimization, was designed [10] to prevent repeated searches by using an accept/reject function that determines whether a local search is appropriate.

In the work presented in this chapter, it is assumed that the offspring will converge to the same local basin after local search (though in practice it may not be the case), so these individuals are forbidden from crossing over with themselves. Consequently, the computational effort can be used to explore

other basins. Because this technique will check the fitness values of the offspring before crossover, it is similar to the use of memory. The memory is used with both Lamarckian evolution and Baldwin effect, and their performances are compared with those without memory in the following experiments. Tables 4 and 5 list the algorithmic descriptions of Lamarckian evolution and Baldwin effect with and without memory, respectively.

Table 4. Algorithmic descriptions of Lamarckian evolution with and without memory

Lamarckian evolution without memory

```
BEGIN
   Generate initial population P(0) randomly,
   i ← 0;
   REPEAT
      Select the parents from P(i) based on their fitness in P(i);
      Apply crossover to the parents and repair if necessary to make it feasible.
      Replace the parents only if the offspring is better;
      Apply mutation to the individuals and replace the population
            if the mutated individual is better and feasible;
      For each solution s0 from the population:
            REPEAT
               Perform local search to get a new solution s;
               If (f(s) < f(s0)) replace s0 by s;
            UNTIL terminal size
   UNTIL the population converges
END
```

Lamarckian evolution with memory

```
BEGIN
   Generate initial population P(0) randomly,
   i ← 0;
   REPEAT
      Select the parents from P(i) based on their fitness in P(i);
      Apply crossover to the parents only if their fitness are different and repair
            if necessary to make it feasible
            and replace the parents only if the offspring is better;
      For each solution s0 from the population;
        REPEAT
           Perform local search to get new a solution s
           If (f(s) < f(s0)) replace s0 by s;
        UNTIL terminal size
   UNTIL the population converges
END
```

Table 5. Algorithmic descriptions of Baldwin effect with and without memory

Baldwin effect without memory

```
BEGIN
   Generate initial population P(0) randomly,
   i ← 0;
   REPEAT
      Select the parents from P(i) based on their fitness(Baldwin) in P(i);
      Apply crossover to the parents and repair if necessary to make it feasible.
      Replace the parents only if the offspring is better;
      Apply mutation to the individuals and replace the population
            if the mutated individual is better and feasible;
      For each solution s0 from the population:
        REPEAT
           Perform local search to get a new solution s;
           Replace s0 by s;
        UNTIL terminal size
   UNTIL the population converges;
END
```

Baldwin effect with memory

```
BEGIN
   Generate initial population P(0) randomly,
   i ← 0;
   REPEAT
      Select the parents from P(i) based on their fitness(Baldwin) in P(i);
      Apply crossover to the parents only if their fitness are different and repair
            if necessary to make it feasible and
            replace the parents only if the offspring is better;
      For each solution s0 from the population;
        REPEAT
           Perform local search to get new a solution s
           Replace s0 by s;
        UNTIL terminal size
   UNTIL the population converges;
END
```

4.3 Experimental Studies

Concentrator-based hybrid EAs using Lamarckian evolution or Baldwin effect are evaluated and compared. Lamarckian evolution and Baldwin Effect are first combined with the various search operators introduced in Section 3.2 to solve the terminal assignment problem with a single-objective, then that with multi-objectives. The experimental setup is the same as the one used previously, as introduced by Table 1 in Section 3.3. Delete-insert one mutation

is used in the local search due to its effectiveness in reducing the cost as well as the number of concentrators.

Single Objective Optimization

Lamarckian evolution and Baldwin effect with and without memory are tested on the single-objective terminal assignment problem. Table 6 shows the total cost obtained by Lamarckian evolution and Baldwin Effect without memory and Table 7 shows the cost obtained with memory. All tests eventually used the same number of concentrators, which is 40, regardless of the use of memory. Because the use of memory influences only crossover, no mutation operators were used in the tests of Lamarckian and Baldwin learning with memory.

The experimental results listed in Tables 6 and 7 show that there is no significant difference in performance between Lamarckian and Baldwin evolution. When the local search is used without memory, the best results obtained are all around 680 for different combinations of search operators. The two-tail t-test on the mean cost also indicates that for a=0.5, none of the local search is significantly different from others. The hybrid EAs found the same basin for different combinations of search operators.

In the case of memory, the two tail t-test on the mean cost again shows no significant difference between Lamarckian and Baldwin evolution. However, the best cost obtained with memory can be lower than 680 when one point crossover, uniform crossover, one node crossover or best node crossover is used (the lowest is 461). This suggests that the use of memory aids the crossover operators to explore other basins and hence the computational effort can be saved from repeated exploration of the same basin. However, the standard deviation is quite high for those operators. It is worth noting that the performance of all the hybrid EAs outperformed the concentrator-based EA without local search as introduced in Section 3.

Multi-objective Optimisation

In real world communications networks, minimising cost and number of concentrators are both important and should be considered at the same time. It is therefore more sensible to deal with them as two independent objectives like in a multi-objective optimisation problem. To enable this, concentrator-based hybrid EAs with multiple objectives are studied.

In multi-objective optimisation, more than one objective should be optimised and these objectives are often in conflict with each other. Obtaining a global optimal solution for all the objectives is therefore not easy. Usually only a set of solutions that are non-dominated (known as Pareto optimal solutions) can be obtained. There are three main approaches to evolutionary multi-objective optimisation: the weighted sum approach, population-based non-Pareto approach and Pareto-based approach [6].

Table 6. Cost comparison between Lamarckian and Baldwin effect without memory in single-objective optimisation

Crossover	Mutation	Lamarckian			Baldwin			t-test
		best	mean	s.d.	best	mean	s.d.	
1	Del-ins one mut.	680	685	8	680	685	11	0.00
One point	Del-ins all mut.	680	685	8	680	685	15	0.00
	Two point conc.	680	685	18	680	684	9	0.27
	Two point term.	680	684	11	680	684	6	0.00
	Self crossover	680	684	10	680	684	5	0.00
2	Del-ins one mut.	680	686	18	680	684	16	0.45
Two point	Del-ins all mut.	680	685	17	680	683	8	0.58
	Two point conc.	680	684	9	680	684	12	0.00
	Two point term.	680	684	14	680	683	8	0.34
	Self crossover	680	684	8	680	683	10	0.42
3	Del-ins one mut.	680	686	18	680	684	7	0.56
Uniform	Del-ins all mut.	680	685	17	680	685	11	0.00
	Two point conc.	680	684	9	680	683	10	-0.40
	Two point term.	680	684	14	680	683	6	0.36
	Self crossover	680	684	8	680	685	11	-0.40
4	Del-ins one mut.	681	685	11	680	685	5	0.00
One node	Del-ins all mut.	680	684	7	680	685	7	-0.55
	Two point conc.	680	685	8	680	685	6	0.00
	Two point term.	680	685	7	680	685	10	0.00
	Self crossover	680	685	9	680	685	7	0.00
5	Del-ins one mut.	680	686	15	680	684	9	0.62
Best node	Del-ins all mut.	680	686	20	680	683	4	0.80
	Two point conc.	680	684	9	680	684	6	0.00
	Two point term.	680	685	16	680	683	4	0.66
	Self crossover	680	684	11	680	684	9	0.00
6	Del-ins one mut.	680	684	7	680	682	5	1.26
One group	Del-ins all mut.	680	684	10	680	684	8	0.00
	Two point conc.	680	684	11	680	684	8	0.00
	Two point term.	680	684	10	680	683	4	0.50
	Self crossover	680	683	15	680	683	4	1.04
7	Del-ins one mut.	680	686	19	680	683	6	0.81
Best group	Del-ins all mut.	680	685	12	680	683	4	0.86
	Two point conc.	680	685	15	680	683	6	0.67
	Two point term.	680	685	13	680	684	9	0.34
	Self crossover	680	686	22	680	684	8	0.46

When hybrid EAs are used for the multi-objective terminal assignment problem, the weighted sum approach is used. It is similar to the single-objective optimisation except that the fitness function explicitly deals with two objectives: one is to minimise the total cost and the other is to minimise the number of concentrators used. The mathematical formulation for

Table 7. Comparison between Lamarckian evolution and Baldwin effect with memory in single-objective optimisation

	Lamarckian			Baldwin			t-test
Type	best	mean	s.d.	best	mean	s.d.	
a) One point crossover	542	741	19021	461	715	15418	0.01
b) Two point crossover	680	684	11	461	683	6	0.43
c) Uniform crossover	583	705	4643	551	726	6577	-0.02
d) One node crossover	661	687	284	662	683	101	0.07
e) Best node crossover	668	684	35	665	685	38	-0.10
f) One group crossover	680	685	17	680	683	4	0.62
g) Best group crossover	680	686	17	680	684	7	0.59

this problem was shown in Section 2.1. However, there are some weaknesses in the weighted sum approach [9]:

1. It can provide only one Pareto solution from one run;
2. It has been shown that the weighted sum approach is unable to deal with a multi-objective optimisation problem with a concave Pareto front [5].

If the weights for different objectives are changing during optimisation, the optimiser may go through all points on the Pareto front. If the searched non-dominated solutions are archived, the whole Pareto front can be achieved. This has been shown to be working well for both convex and concave Pareto fronts. Whether the weighted sum approach is able to converge to a Pareto-optimal solution depends on the stability of the Pareto solution corresponding to the given weight combination. Without considering the time consumption, the whole Pareto front can be obtained by running the optimiser as long as possible [9].

Investigation of Varied Weights

To examine the weight effect on optimisation, varying weights between 0.1 and 0.9 are set for both objectives of the terminal assignment problem. In the experiments, EAs with one point crossover and delete-insert one mutation are used, and all the other experimental settings are the same as those used for the performance tests of various search operators, as introduced by Table 1 in Section 3.3. Table 8 summarises the experimental results, including the number of concentrators used, the total cost obtained and the number of generations required by each EA. Figure 17 shows the relationship between the values obtained for both objectives, i.e., cost vs the number of concentrators.

In Table 8, Weight One indicates the weight assigned to the first objective (cost) and Weight Two is that assigned to the number of concentrators. As Weight One increases, the cost decreases as expected. The decrease in cost became less obvious while weight one is higher than 0.4. The EA reaches a relatively reasonable performance for both objectives when their weights are

around 0.5. In the following experiments of hybrid EAs for multi-objective optimisation, we choose 0.5 as the weights for both objectives.

Table 8. Results obtained by varied weights in the weighted sum approach to multi-objective optimisation

	Weight one	Weight two	Number of concentrators	Cost best	mean	worst	s.d.	Generations
1	0.10	0.90	20	1623	2101	2528	50012	62
2	0.20	0.80	21	1029	1256	1466	10207	50
3	0.30	0.70	23	888	1008	1126	3538	52
4	0.40	0.60	24	795	846	918	1110	53
5	0.50	0.50	27	740	770	813	236	59
6	0.60	0.40	29	713	733	760	170	50
7	0.70	0.30	30	697	714	745	144	51
8	0.80	0.20	33	689	696	712	27	52
9	0.90	0.10	35	684	690	697	10	50

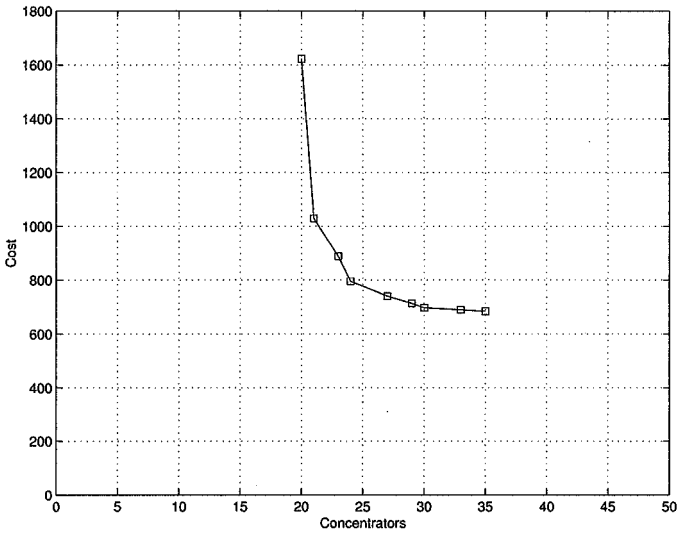


Fig. 17. Relationship between obtained cost and number of concentrators.

Comparison Between Lamarckian and Baldwin Effect in Multi-objective Optimization

In multi-objective optimization, both Lamarckian evolution and Baldwin effect are tested with various combinations of search operators. For simplicity, the use of memory is not considered in these tests. The weights for the two objectives are set as 0.5. Table 9 shows the comparison results.

In the Lamarckian-style hybrid EA, the best cost obtained was 719 when one point crossover was used with delete-insert all mutation and two point crossover was used with two point concentrator swap. The corresponding number of concentrators used was 26 in both cases. In Baldwin effect, the best cost obtain was 720 and the corresponding number of concentrators used was also 26, when two point crossover and two point concentrator swap mutation are used together. The experimental results again demonstrate that there is no significant difference between these two local search methods.

When comparing the results obtained for single-objective and multi-objective optimization, hybrid EA in single-objective optimization sometimes obtained a lower cost than in multi-objective optimization, such as the Lamarckian evolution with one point crossover plus two point terminal swap mutation, and Baldwin effect with uniform crossover plus self crossover mutation and with best node crossover plus delete-insert all mutation. The cost achieved for the single objective case is around 685, compared with the cost around 760 achieved for the multi-objective case. The number of concentrators used, however, is much lower in the multi-objective case, which is around 26, compared with 40 obtained in the single-objective case. If taking both objectives into consideration, the multi-objective optimization performed better in satisfying two objectives simultaneously than the single objective optimization. It is worth noting that in either case, hybrid EAs with local search outperformed EAs without local search as given in Section 3.

5 Conclusions and Future Work

Communications network design is essential to the development and implementation of widely used packet switch networks and fiber optical networks. Optimal communications network design is challenging since it needs to satisfy multiple constraints and to minimize one or more objectives at the same time. EAs have been shown to perform well for the terminal assignment problem. Their performance can be further enhanced by a new concentrator-based chromosome representation and by hybridization with local search.

This chapter proposes a novel concentrator-based representation that utilizes the group character of terminals and concentrators to overcome the limitations of the traditional terminal-based representation. A series of new search operators including crossover and mutation are designed for the concentrator-based representation. The concentrator-based EAs have been shown to outperform other terminal-based EAs. Our computational study also demonstrates

Table 9. Comparison between Lamarckian evolution and Baldwin effect in multi-objective optimisation

Xover	Mutation	Lamarckian				Baldwin				t-test
		conc.	best	mean	s.d.	conc.	best	mean	s.d.	
One point	Del-ins one mut.	27	740	770	236	27	732	765	418	0.06
	Del-ins all mut.	26	719	757	431	26	723	756	495	0.01
	Two point conc.	26	731	757	394	26	721	756	323	0.01
	Two point term.	26	740	767	211	26	737	764	424	0.04
	Self crossover	26	724	753	10	26	725	759	325	-0.08
Two point	Del-ins one mut.	26	745	780	592	26	734	770	543	0.07
	Del-ins all mut.	26	727	761	342	26	721	752	349	0.10
	Two point conc.	26	719	753	229	26	720	756	575	-0.03
	Two point term.	26	739	776	453	26	729	768	824	0.05
	Self crossover	26	728	753	242	26	727	753	788	0.00
Unif.	Del-ins one mut.	27	733	768	513	26	736	774	771	-0.04
	Del-ins all mut.	26	724	753	260	26	726	757	461	-0.04
	Two point conc.	26	733	760	312	26	729	751	246	0.12
	Two point term.	26	731	764	473	26	736	766	281	-0.02
	Self crossover	26	723	754	511	26	725	750	162	0.04
One node	Del-ins one mut.	27	734	767	617	26	728	761	374	0.08
	Del-ins. all mut.	26	723	754	517	26	723	750	232	0.04
	Two point conc.	26	730	756	429	26	725	747	281	0.10
	Two point term.	27	732	761	353	27	727	257	261	0.05
	Self crossover	26	724	750	262	27	725	758	555	-0.07
Best node	Del-ins one mut.	27	738	767	617	27	724	754	221	0.11
	Del-ins all mut.	26	724	763	399	26	726	751	113	0.16
	Two point conc.	26	725	757	499	26	722	750	233	0.07
	Two point term.	26	739	770	413	26	721	757	311	0.14
	Self crossover	26	723	749	259	26	731	751	229	-0.03
One group	Del-ins one mut.	26	740	772	384	26	741	767	281	0.06
	Del-ins all mut.	26	726	762	308	25	731	762	267	0.00
	Two point conc.	26	725	755	251	26	727	753	282	0.03
	Two point term.	26	733	771	494	26	735	761	221	0.10
	Self crossover	26	727	752	293	26	722	751	274	0.02
Best group	Del-ins one mut.	26	744	776	381	26	740	764	204	0.15
	Del-ins all mut.	25	721	768	616	26	730	755	198	0.11
	Two point conc.	26	729	754	436	26	732	755	264	-0.01
	Two point term.	26	766	685	236	27	720	328	766	0.00
	Self crossover	26	733	762	212	26	727	750	146	0.25

the good scalability of the concentrator-based EAs, which can still work well with the number of terminals up to 1000.

Hybrid EAs integrating Lamarckian evolution or Baldwin effect with or without memory have been designed to tackle both the single-objective and multi-objective formulations of the terminal assignment problem. Our experimental results reveal that Lamarckian evolution and Baldwin effect performed

similarly in most cases for the terminal assignment problem. However, the hybrid EAs obviously outperformed the EAs without local search.

It is worth noting that the proposed concentrator-based hybrid EAs are not limited to the terminal assignment problem. They can also be applied to other real world applications, such as bin packing and cutting stock problems. Further study of the concentrator-based hybrid EAs in these applications will be carried out. Although the work presented here includes a comprehensive investigation of the performance of various search operators for the concentrator-based representation, the most proper combination of these operators for the concentrator-based hybrid EA still needs further study. In particular, we are interested in analysing those group based crossover and mutation, which showed unsatisfactory performance in the experiments. Another work we want to investigate is the use of memory in hybrid EAs. Our experiments show that there is no significant difference between the EAs with and without memory. This is somewhat anti-intuitive and needs to be investigated further.

References

1. Abuali, F., Schoenefeld, D. and Wainwright, R. (1994). Terminal assignment in a Communications Network Using Genetic Algorithms. Proceedings of the 22nd Annual ACM Computer Science Conference, pages 74-81. ACM Press.
2. Aggarwal, K.K., Chopra, Y.C. and Bajwa, J.S. (1982). Reliability evaluation by network decomposition. IEEE Transactions on Reliability, R-31:355-358.
3. Atiqullah, M.M. and Rao, S.S. (1993). Reliability optimization of communication networks using simulated annealing. Microelectronics and Reliability, 33:1303-1319.
4. Falkenauer, E. (1996). A hybrid grouping genetic algorithm for bin packing. Journal of Heuristics, 2(1):5-30.
5. Fleming, P.J. (1985). Computer aided control systems using a multi-objective optimization approach. Proceedings of IEE Control'85 Conference, pages 174-179.
6. Fonseca, C.M. and Flemming, P.J. (2000). Multiobjective optimisation. In Back, T., Fogel, D.B. and Michalewicz, Z., editors, Evolutionary computation, volume 2, pages 25-27. Institute of Physics Publishing, Bristol.
7. Glover, F., Lee, M. and Ryan, J. (1991). Least-cost network topology design for a new service: and application of a tabu search. Annals of Operations Research, 33:351-362.
8. Hinton, G.E. and Nolan, S.J. (1987). How learning can guide evolution. Complex Systems, 1:495-502.
9. Jin, Y., Olhofer, M. and Sendhoff, B. (2001). Dynamic Weighted Aggregation for Evolutionary Multi-Objective Optimization: Why Does It Work and How? Proceedings of the Genetic and Evolutionary Computation Conference GECCO, pages 1042-1049. Morgan Kaufmann.
10. Joines, A.J. and Kay, M.G. (2002). Utilizing Hybrid Genetic Algorithms. In Sarker, R., Mohammadian, M and Yao, X., editors, Evolutionary Optimisation. Kluwer Academic Publishers. pp.199-228.
11. Kershenbhaum, A. (1993). Telecommunications Network Design Algorithms. McGraw-Hill.
12. Khuri, S. and Chiu, T. (1997). Heuristic Algorithms for the Terminal Assignment Problem. Proceedings of the 1997 ACM Symposium on Applied Computing, pages 247-251. ACM Press.
13. Koh, S.J. and Lee, C.Y. (1995). A tabu search for the survivable fiber optic communication network design. Computers and Industrial Engineering, 28:689-700.
14. Kumar, A., Pathak, R.M., Gupta, Y.P. and Parsaei, H.R. (1995). A genetic algorithm for distributed system topology design. Computers and Industrial Engineering, 28:659-670.
15. Pierre, S., Hyppolite, M.A., Bourjolly, J.M. and Dioume, O. (1995). Topological design of computer communication networks using simulated annealing. Engineering Applications of Artificial Intelligence, 8:61-69.
16. Whitley, D., Gordon, V.S. and Mathias, K. (1994). Lamarckian evolution, the Baldwin effect and function optimization. In Davidor, Y., Schwefel, H.-P. and Mnner R., editors, Lecture Notes in Computer Science, 866:6–15, Springer-Verlag.

Effective Exploration & Exploitation of the Solution Space via Memetic Algorithms for the Circuit Partition Problem

Shawki Areibi[1]

School of Engineering, University of Guelph sareibi@uoguelph.ca

1 Introduction

Genetic Algorithms **(GA's)** are a class of evolutionary techniques that seek improved performance by sampling areas of the parameter space that have a high probability for leading to good solutions [11]. The evolution program is a probabilistic algorithm which maintains a population of individuals (chromosomes). Each chromosome represents a potential solution within the landscape of the problem at hand. These individuals undergo transformations based on operators to create new populations (solutions). Many evolution programs can be formulated to solve different problems. These programs may differ in the data structures, parameter tuning, specific genetic operators but share some common principles (i) population of individuals (ii) genetic operators to transform individuals into new (possibly better) solutions. The power of GA's comes from the fact that the technique is robust, and can deal successfully with a wide range of problem areas, including those which are difficult for other methods to solve. GA's are not guaranteed to find the global optimum solution to a problem, but they are generally good at finding "acceptably good" solutions to problems. In other words, GA's are considered to be competitive if: the solution space to be searched is large (exploration) and the fitness function is noisy (landscape is not smooth nor unimodal).

Genetic Algorithms are not well suited for fine-tuning structures and incorporation of local improvement has become essential for Genetic Algorithms to compete with other meta-heuristic techniques. Memetic Algorithms [1] apply a separate local search process to refine individuals by hill climbing.

1.1 Motivation and Contributions

Efficient optimization algorithms used to solve hard problems usually employ a hybrid of at least two techniques to find a near optimal solution to the problem being solved. The main motivation for hybridization in optimization practice

is the achievement of increased efficiency (i.e adequate solution quality in minimum time or maximum quality in specified time). From an optimization point of view, Memetic Algorithms combine global and local search by using Evolutionary Algorithms (EA) to perform exploration while the local search method performs exploitation.

The main contributions of this book chapter are (i) investigation of parameter tuning of Genetic Algorithms to solve the circuit partitioning problem effectively, (ii) investigating the balance between exploration and exploitation of the solution space.

1.2 Chapter Organization

The book chapter is organized as follows: Section 2 introduces very briefly the VLSI circuit partitioning problem and terminology used throughout the chapter. The concept of evolutionary computation and Genetic Algorithms will be introduced in Section 3. Section 4 introduces the need behind Memetic Algorithms to further explore the solution space effectively. Results are introduced in Section 5. The chapter concludes with some comments on the issue of effective space exploration and exploitation and possible future work.

2 Background

The last decade has brought explosive growth in the technology for manufacturing integrated circuits. Integrated circuits with several million transistors are now commonplace. This manufacturing capability, combined with the economic benefits of large electronic systems, is forcing a revolution in the design of these systems and providing a challenge to those people interested in integrated system design. Since modern circuits are too complex for an individual designer or a group of designers to comprehend completely, managing this tremendous complexity and automating the design process have become crucial issues.

A large subset of problems in VLSI CAD is computationally intensive, and future CAD tools will require even more accuracy and computational capabilities from these tools. In the combinatorial sense, the layout problem is a constrained optimization problem. We are given a circuit (usually a module-wire connection-list called a *netlist*) which is a description of switching elements and their connecting wires. We seek an assignment of geometric coordinates of the circuit components (in the plane or in one of a few planar layers) that satisfies the requirements of the fabrication technology (sufficient spacing between wires, restricted number of wiring layers, and so on) and that minimizes certain cost criteria. The most common way of breaking up the layout problem into subproblems is first to do *logic partitioning* where a large circuit is divided into a collection of smaller modules according to some criteria, then to perform component *placement*, and then to determine the

approximate course of the wires in a *global routing* phase. This phase may be followed by a *topological-compaction* phase that reduces the area requirement of the layout, after which a *detailed-routing* phase determines the exact course of the wires without changing the layout area.

2.1 Circuit Partitioning

Circuit partitioning is the task of dividing a circuit into smaller parts. It is an important aspect of layout for several reasons. Partitioning can be used directly to divide a circuit into portions that are implemented on separate physical components, such as printed circuit boards or chips. Here, the objective is to partition the circuit into parts such that the sizes of the components are within prescribed ranges and the complexity of connections (nets cut) between the components is minimized. Figure 1 presents a circuit that is partitioned into two blocks (partitions) with a single cut introduced. The inputs/outputs of the circuit represent the terminals (I/O pads) of the circuit. All gates/cells are interconnected by using nets (hyperedges).

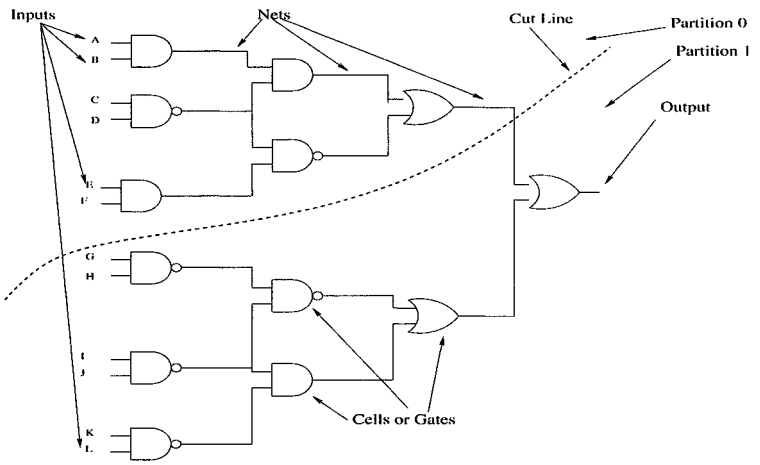


Fig. 1. Circuit Partitioning & Terminology

2.2 Benchmarks

The quality of solutions obtained for the circuit partitioning problem are based on a set of hypergraphs that are part of widely used ACM/SIGDA [12] circuit partitioning benchmarks suite. The characteristics of these hypergraphs are shown in Table 1. The second column of the table shows the number of cells within the circuit. The third column presents the number of nets connecting

the cells within the benchmarks. The total number of pins (i.e connections) within the circuit is summarized in column four. The last two columns summarize the statistics of the circuit (i.e connectivity).

Table 1. Benchmarks Used as Test Cases

Circuit	Cells	Nets	Pins	Cell Degree			Net Size		
				MAX	$\overline{x}$	σ	MAX	$\overline{x}$	σ
Fract	125	147	462	7	3.1	1.6	17	3.1	2.2
Prim1	833	902	2908	9	3.4	1.2	18	3.2	2.5
Struct	1888	1920	5471	4	2.8	0.6	17	2.8	1.9
Ind1	2271	2192	7743	10	3.4	1.1	318	3.5	9.0
Prim2	2907	3029	18407	9	3.7	1.5	37	3.7	3.8
Bio	6417	5742	21040	6	3.2	1.0	861	3.6	20.8
Ind2	12142	13419	48158	12	3.8	1.8	585	3.5	10.9
Ind3	15057	21808	65416	12	4.3	1.4	325	2.9	3.2
Avq.s	21854	22124	76231	7	3.4	1.4	4042	3.4	53.3
Avq.l	25144	25384	82751	7	3.2	1.2	4042	3.2	49.8
Ibm05	29347	28446	126308	9	4.3	2.3	17	4.4	4.2
ibm07	45926	48117	175639	98	3.8	2.4	25	3.6	3.0
ibm10	69429	75196	297567	137	4.2	3.2	41	3.9	3.5
ibm13	84199	99666	357075	180	4.2	3.3	24	3.5	3.0

2.3 Heuristic Techniques for Circuit Partitioning

Heuristic algorithms for combinatorial optimization problems in general and circuit partitioning in particular can be classified as being *constructive* or *iterative*. Constructive algorithms determine a partitioning from the graph describing the circuit or system, whereas iterative methods aim at improving the quality of an existing partitioning solution. Constructive partitioning approaches are mainly based on clustering[3, 6], spectral or eigenvector methods[5], mathematical programming or network flow computations. To date, iterative improvement techniques that make local changes to an initial partition are still the most successful partitioning algorithms in practice. The advantage of these heuristics is that they are quite robust. In fact, they can deal with netlists as well as arbitrary vertex weights, edge costs, and balance criteria.

Constructive Based Techniques (GRASP)

GRASP is a greedy randomized adaptive search procedure that has been successful in solving many combinatorial optimization problems efficiently [8,

4]. Each iteration consists of a construction phase and a local optimization phase. The construction phase intelligently constructs an initial solution via an adaptive randomized greedy function. Further improvement in the solution produced by the construction phase may be possible by using either a simple local improvement phase or a more sophisticated procedure in the form of Tabu Search or Simulated Annealing. The construction phase is iterative, greedy and adaptive in nature. It is *iterative* because the initial solution is built by considering one element at a time. The choice of the next element to be added is determined by ordering all elements in a list. The list of the best candidates is called the restricted candidate list (RCL). It is *greedy* because the addition of each element is guided by a greedy function. The construction phase is *randomized* by allowing the selection of the next element added to the solution to be any element in the RCL. Finally, it is *adaptive* because the element chosen at any iteration in a construction is a function of those previously chosen.

Iterative Improvement

Kernighan and Lin (KL) [10] described a successful iterative heuristic procedure for graph partitioning which became the basis for most module interchange-based improvement partitioning heuristics used in general. Their approach starts with an initial bisection and then involves the exchange of pairs of vertices across the cut of the bisection to improve the cut-size. The algorithm determines the vertex pair whose exchange results in the largest decrease of the cut-size *or* in the smallest increase, if no decrease is possible. A pass in the Kernighan and Lin algorithm attempts to exchange all vertices on both sides of the bisection. At the end of a pass the vertices that yield the best cut-size are the only vertices to be exchanged. Fiduccia and Mattheyses (FM) [7] modified the Kernighan and Lin algorithm by suggesting to move one cell at a time instead of exchanging pairs of vertices, and also introduced the concept of preserving balance in the size of blocks. The FM method reduces the time per pass to linear in the size of the netlist (i.e O(p), where p is the total number of pins) by adapting a single-cell move structure, and a gain bucket data structure that allows constant-time selection of the highest-gain cell and fast gain updates after each move.

Figure 2(a) shows the swap/move of modules between blocks that may lead to a reduction of nets cut. Each module is initially labeled to be free **"F"** to move, but once moved during a pass it is relabeled to be locked **"L"**. The gain of moving a specific module from one partition to another is maintained by using the bucket gain data structure (shown in Figure 2(b)). At the end of a pass only those modules that contribute to the highest gain (i.e reduction in cut size) are allowed to move to their new destination (as illustrated in Figure 2(c)).

Figure 3 shows the basic Fiduccia-Mattheyses (FM) algorithm used for circuit partitioning[7].

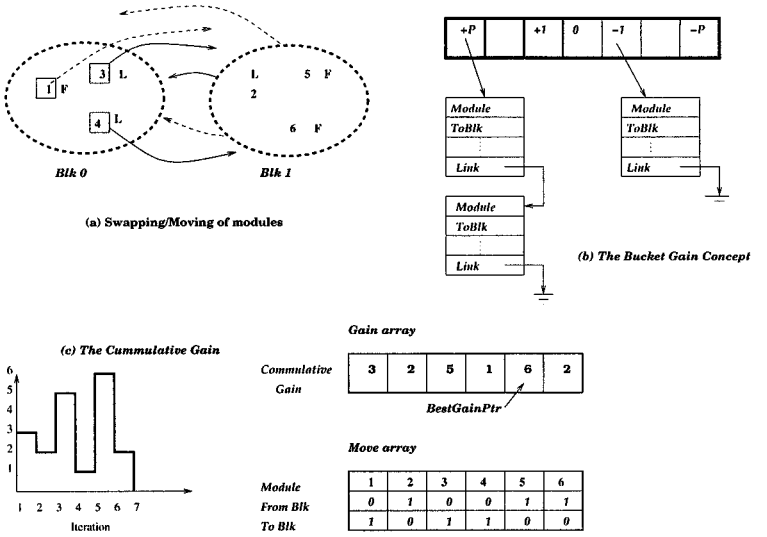


Fig. 2. Basic techniques for Interchange Methods

```
current_solution ← initial_solution
current_cost ← evaluate(current_solution)
Repeat
        initialize partition
        While (can_move(modules))
              choose cell with highest gain
              update gains of all cells
              if (current_gain > previous_gain)
                    bestgain = current_gain
        end while
        move nodes pointed to by bestgain_ptr
        if (no improvement)
              ++noimp_counter
Until((pass > MaxPass) OR
              (noimp > MaxNoImp))
```

Fig. 3. Fiduccia Mattheyses Algorithm

Sanchis [13] uses the above technique for multiple way network partitioning. Under such a scheme, we should consider all possible moves of each free cell from its home block to any of the other blocks, at each iteration during a pass the best move should be chosen. As usual, passes should be performed until no improvement in cutset size is obtained. This strategy seems to offer some hope of improving the partition in a homogeneous way, by adapting the level gain concept to multiple blocks.

Table 2 presents the results obtained using Sanchis local search technique for two-way and multi-way partitioning. The results are the average of fifty runs. The CPU time increases dramatically as the number of partitions increase in size from 2-way to 4-way and ultimately to 8-way partitioning. In general, node interchange methods are greedy or local in nature and get easily trapped in local minima. More important, it has been shown that interchange methods fail to converge to "optimal" or "near optimal" partitions unless they initially begin from "good" partitions. Sechen [14] shows that over 100 trials or different runs (each run beginning with a randomly generated initial partition) are required to guarantee that the best solution would be within twenty percent of the optimum solution. In order for interchange methods to converge to "near optimal" solutions they have to initially begin from "good" starting points [2].

Table 2. Multi-Way Partitions Based on Local Search

Circuit	2 Blocks		4 Blocks		6 Blocks		8 Blocks	
	Cuts	CPU	Cuts	CPU	Cuts	CPU	Cuts	CPU
Fract	11	0.3	28	0.3	48	0.4	56	0.5
Prim1	58	2.3	148	2.7	171	3.3	189	4.0
Struct	46	5.8	195	6.4	264	8.4	312	10.5
Ind1	30	7.2	245	8.3	364	12.5	374	16.6
Prim2	230	12.4	636	13.3	773	19.1	804	28.0
Bio	91	28.4	532	45.8	726	71.9	806	105.9
Ind2	507	70.4	1759	143.1	2162	272.2	2141	394.4
Ind3	396	63.5	1675	118.4	2636	190.2	2862	280.7
Avq.s	453	126.2	2151	309.9	2436	499.5	2641	674.7
Avq.l	460	178.1	2594	321.8	2728	594.5	3027	857.1
Ibm05	2451	329.4	8922	1618	9629	3719	9894	6059
ibm07	1350	518.3	13527	4437	15922	11820	17011	23185
ibm10	1972	1068	22331	12855	26544	40252	27835	79470
ibm13	1560	1365	26710	16456	31949	53715	34171	105000

3 Genetic Algorithms

As an optimization technique, Genetic Algorithms simultaneously examine and manipulate a set of possible solutions. Figure 4 illustrates a Genetic Algorithm implementation for circuit partitioning.

```
        A PURE GENETIC ALGORITHM
1. Represent Problem Using Group Number Encoding
2.(a) set pop_size, max_gen, gen=0;
  (b) set cross_rate, mutate_rate;
3. Initialize Population.
4. While max_gen ≥ gen
       Evaluate Fitness (Cuts)
       For (i=1 to pop_size)
         Select (mate1,mate2)
         if (rnd(0,1) ≤ cross_rate)
           child = Crossover(mate1,mate2);
         if (rnd(0,1) ≤ mutate_rate)
           child = Mutation();
         Repair child if necessary
       End For
       Add offsprings to New Generation.
       gen = gen + 1
   End While
5. Return best chromosomes (Partitions).
```

Fig. 4. A Genetic Algorithm for Circuit Partitioning

The GA starts with several alternative solutions to the optimization problem, which are considered as individuals in a *population.* These solutions are coded as binary strings, called *chromosomes.* Figure 5 shows a group number encoding scheme to represent the partitioning problem where the j^{th} integer $i_j \in \{1, \ldots, k\}$ indicates the group number assigned to object j.

The initial population is constructed randomly. These individuals are *evaluated,* using the partitioning-specific fitness function. The GA then uses these individuals to produce a new *generation* of hopefully better solutions. In each generation, two of the individuals are selected probabilistically as *parents,* with the selection probability proportional to their fitness. *Crossover* is performed on these individuals to generate two new individuals, called *offspring,* by exchanging parts of their structure. Thus each offspring inherits a combination of features from both parents. The next step is *mutation* where an incremental change is made to each member of the population, with a small probability. This ensures that the GA can explore new features that may not be in the population yet. It makes the entire search space reachable, despite the finite

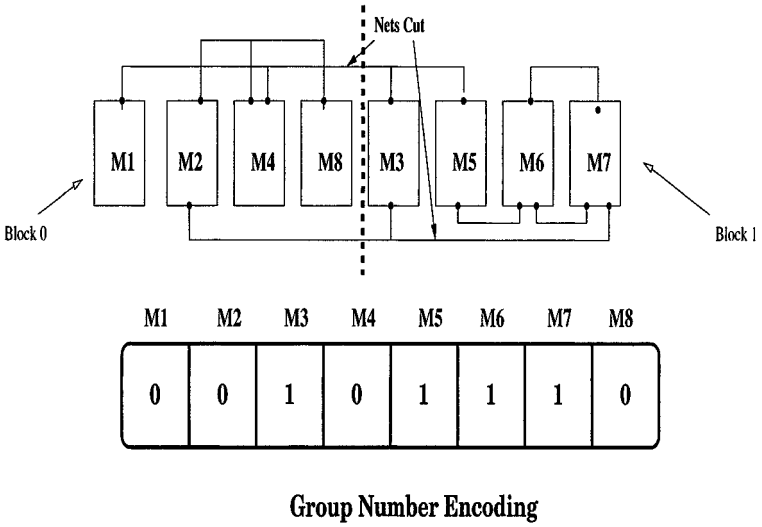


Fig. 5. Chromosome Representation for Circuit Partitioning

population size. However an offspring may contain less than k groups; moreover, an offspring of two parents, both representing feasible solutions may be infeasible, since the constraint of having equal number of modules in each partition is not met. In this case either special *repair heuristics* are used to modify chromosomes to become feasible, or *penalty functions* that penalize infeasible solutions, are used to eliminate the problem.

3.1 Crossover & Mutation

Figure 6 shows the crossover/mutation operators used for the circuit partitioning problem. Operators in the reproduction module, mimic the biological evolution process, by using unary (mutation type) and higher order (crossover type) transformation to create new individuals. *Mutation* as shown in Figure 6(a) is simply the introduction of a random element, that creates new individuals by a small change in a single individual. When mutation is applied to a bit string, it sweeps down the list of bits, replacing each by a randomly selected bit, if a probability test is passed. On the other hand, *crossover* recombines the genetic material in two parent chromosomes to make two children. It is the structured yet random way that information from a pair of strings is combined to form an offspring. Crossover begins by randomly choosing a cut point K where $1 \leq K \leq L$, and L is the string length. The parent strings are both bisected so that the left-most partition contains K string elements, and the rightmost partition contains $L - K$ elements. The child string is formed by copying the rightmost partition from parent P_1 and then the left-most

partition from parent P_2. Figure 6(b) shows an example of applying the standard crossover operator (sometimes called one-point crossover) to the group number encoding scheme. Increasing the number of crossover points is known to be multi-point crossover as seen in Figure 6(c).

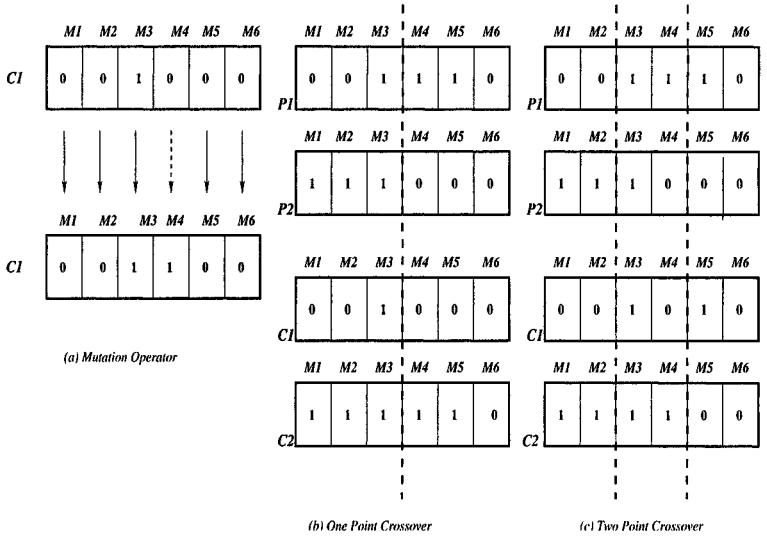


Fig. 6. Mutation & Crossover Operators

Figure 7 and Figure 8 show the affect of mutation rate on the quality of solutions obtained. Figure 9 and Figure 10 highlight the importance of tuning

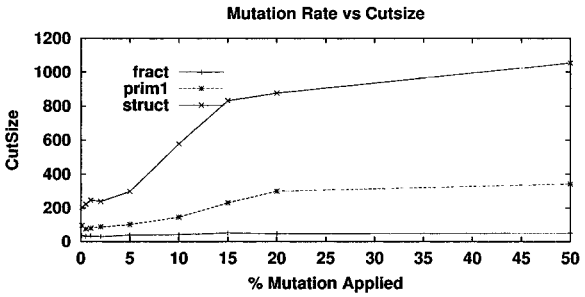


Fig. 7. Mutation Rate (Small Circuits)

the crossover rate and its affect on the solution quality. Figures 11, 12, 13 show the affect of crossover points. It is clear from the figures that multi-point

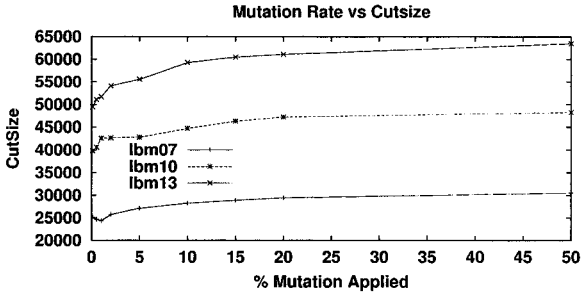


Fig. 8. Mutation Rate (Very Large Circuits)

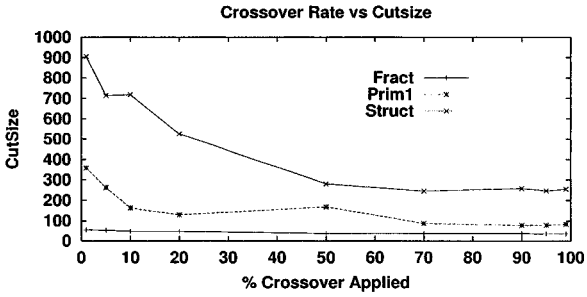


Fig. 9. Crossover Rate (Small Circuits)

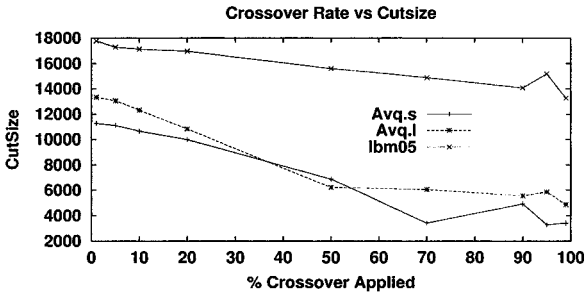


Fig. 10. Crossover Rate (Large Circuits)

crossover performs much better than one-point crossover technique. A 3-point and 4-point crossover works best for our circuit partitioning problem.

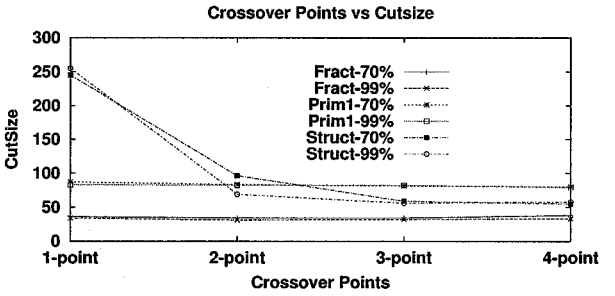


Fig. 11. Crossover Points (Small Circuits)

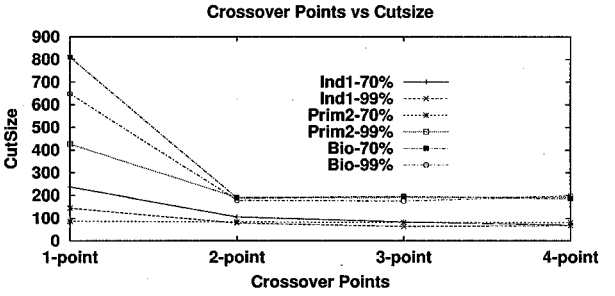


Fig. 12. Crossover Points (Medium Circuits)

3.2 Population/Generation Size

The size of the population is one of the most important choices in implementing any Genetic Algorithm and is considered to be critical for several applications. If the population size is too small then this may lead to early convergence and if it is too large this may lead to huge computation time (i.e waste of computational resources). Figure 14 shows the affect of the population size on the quality of solutions obtained for large circuits. The population in any Genetic Algorithm implementation evolves for a prespecified total number of generations under the application of evolutionary rules. The generation size is crucial in any Genetic Algorithm implementation. As the number of generations increase the quality of solutions improve, but the computation

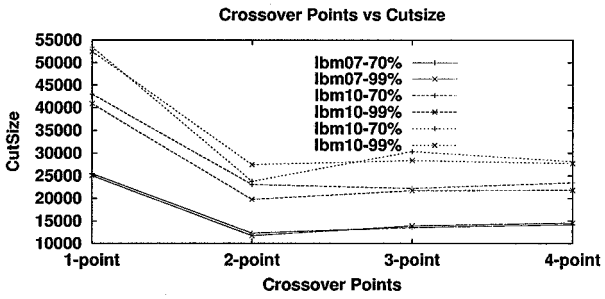


Fig. 13. Crossover Points (Very Large Circuits)

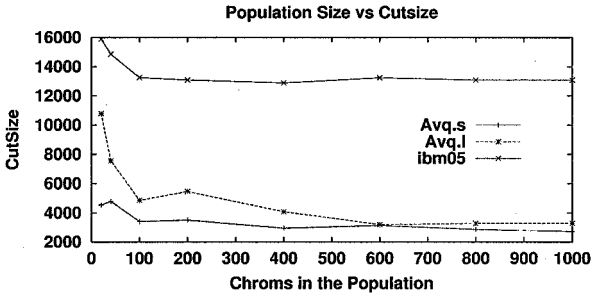


Fig. 14. Population Size (Large Benchmarks)

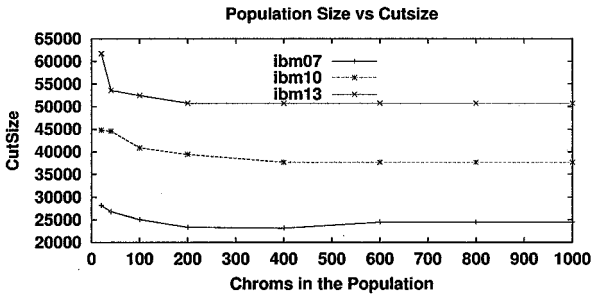


Fig. 15. Population Size (Very Large Benchmarks)

time involved increases dramatically. Figure 16 and Figure 17 show the affect of generation size on the solution quality obtained based on large circuits and very large circuits respectively.

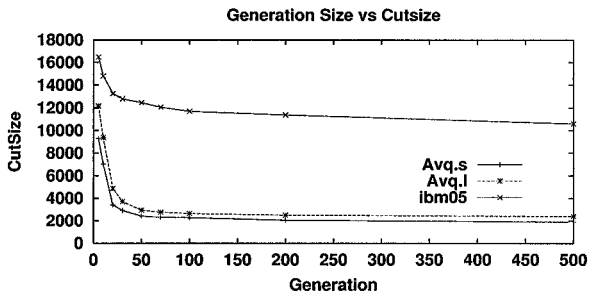


Fig. 16. Affect of Generation Size for Large Benchmarks

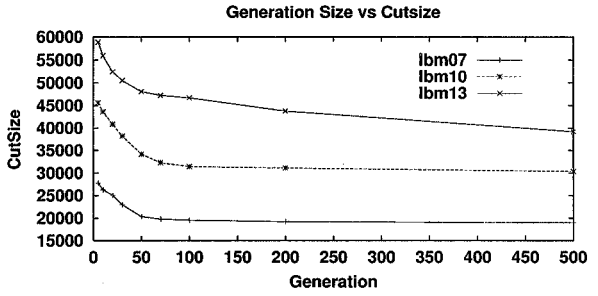


Fig. 17. Affect of Generation Size for Very Large Benchmarks

3.3 Selection Techniques

Strings are selected for mating based on their fitness, those with greater fitness are awarded more offspring than those with lesser fitness. Parent selection techniques that are used, vary from stochastic to deterministic methods. The probability that a string i is selected for mating is p_i, "the ratio of the fitness of string i to the sum of all string fitness values", $p_i = \frac{fitness_i}{\sum_j fitness_j}$. The ratio of individual fitness to the fitness sum denotes a ranking of that string in the population. The Roulette Wheel Selection method (Gsm1) is one of the stochastic selection techniques that is widely used. The ratio p_i is used to construct a weighted roulette wheel, with each string occupying an area on

the wheel in proportions to this ratio. The wheel is then employed to determine the string that participates in the reproduction. A random number generator is invoked to determine the location of the spin on the roulette wheel. In Deterministic Selection methods, reproduction trials (selection) are allocated according to the rank of the individual strings in the population rather than by individual fitness relative to the population average. Several selection methods have been implemented as seen in Figure 18 and 19. The technique referred to as *Gsm0* is a deterministic technique where parents are picked uniformly one after the other from the population. *Gsm1* is the stochastic roullette wheel technique. In *Gsm2* the population is sorted according to their fitness each trial the best two in the list are chosen for mating. *Gsm3* is similar to *Gsm2* except that the first half of the sorted list would take higher chances for mating than the rest of the population at the end of the list. *Gsm4* and *Gsm5* are based on a ranking technique. The last two approaches *Gsm6* and *Gsm7* are based on Tournament with replacement and without replacement respectivley. It is clear from Figures 18 and 19 that Tournament Selection with replacement gives the best solution quality compared to other selection techniques.

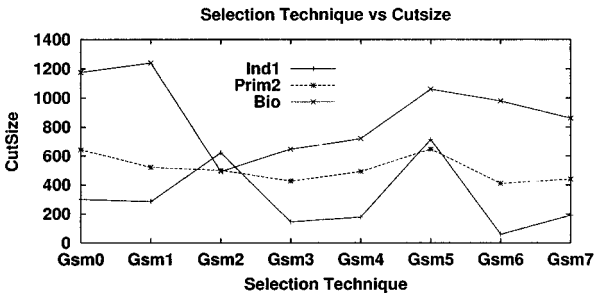


Fig. 18. Selection vs Cutsize (Medium Circuits)

3.4 Replacement Strategy

Generation replacement techniques are used to select a member of the old population and replace it with the new offspring. The quality of solutions obtained depends on the replacement scheme used. Some of the replacement schemes used are based on: (i) deleting the old population and replacing it with new offsprings (R-ap), (ii) both old and new populations are sorted and the newly created population is constructed from the top half of each (R-hp), (iii) replacing parent with the child if newly created member is more fit (R-pc) (iv) replacing the most inferior members (R-mi) in a population by new offsprings. Figure 20 and 21 show the performance of each replacement

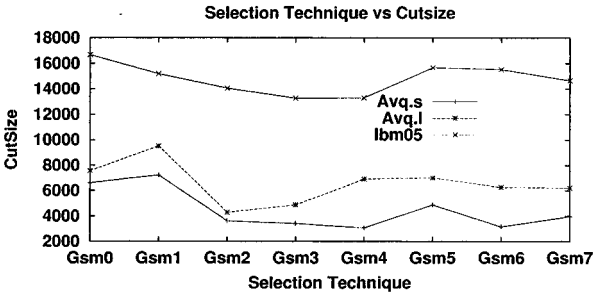


Fig. 19. Selection vs Cutsize (Large Circuits)

technique for large circuits and very large circuits respectivley. It is evident from the Figures that (R-ap) and (R-pc) perform poorly with respect to (R-hp) and (R-mi). Variations to (R-hp) scheme use an incremental replacement approach, where at each step the new chromosome replaces one randomly selected from those which currently have a *below-average* fitness. The quality of solutions improve using (R-hp) and (R-mi) replacement schemes due to the fact that they maintain a large diversity in the population. Our generation replacement technique utilized in both the pure Genetic Algorithm and Memetic Algorithm for circuit partitioning are based on replacing the most inferior member (R-mi) in a population by new offsprings.

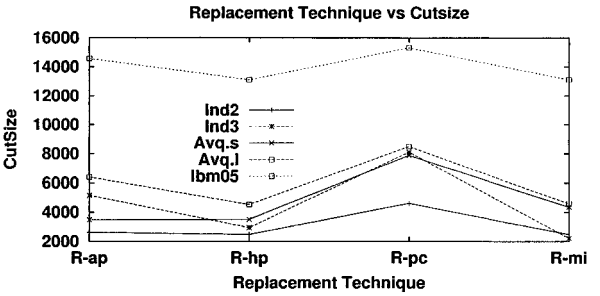


Fig. 20. Replacement Strategy vs Cutsize (Large Circuits)

3.5 Computational Results for GA

Table 3 shows the solution quality for multi-way partitioning and CPU time involved. It is interesting to note that the Genetic Algorithm solution quality compared to Local Search is better for small, medium and large circuits for

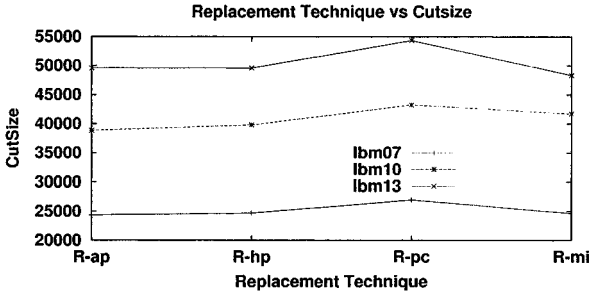


Fig. 21. Replacement Strategy vs Cutsize (Very Large Circuits)

2-way and multi-way partitions. As the size of the circuit increases, the performance of GA deteriorates (as can be seen for benchmarks ibm07, ibm10 and ibm13). On the other hand the complexity of Genetic Algorithm in terms of CPU time is linear as the number of blocks increases. For example, comparing Table 2 and Table 3 for benchmark ibm13, the GA technique cuts the CPU time by almost 50%.

Table 3. Genetic Algorithm Solution Quality for Multi-Way Partitioning

Circuit	2 Blocks		4 Blocks		6 Blocks		8 Blocks	
	Cuts	CPU	Cuts	CPU	Cuts	CPU	Cuts	CPU
Fract	18	19	39	24	49	28	52	38
Prim1	80	126	145	156	136	182	159	234
Struct	52	277	161	344	231	402	255	532
Ind1	70	326	111	408	154	493	159	640
Prim2	186	460	325	581	409	690	557	892
Bio	176	881	266	1122	328	1340	367	1757
Ind2	272	2103	1010	2778	1038	3881	1590	4857
Ind3	491	3106	1337	4645	2130	5753	2341	7801
Avq.s	464	3911	986	4831	1111	7110	1425	9821
Avq.l	465	3999	1002	6336	1093	8066	1426	11272
Ibm05	9248	5585	11890	8158	13026	10918	13704	14690
ibm07	12529	11414	18183	16901	20496	21357	20499	27626
ibm10	20624	22652	29108	30507	31900	37503	32983	47272
ibm13	25876	32610	38186	41371	41452	49693	43139	61771

Comparing results obtained by the Genetic Algorithm with those based on Local Search we can conclude the following. (i) GA's are not guaranteed to find the global optimum solution to a problem, but they are generally good

at finding "acceptably good" solutions to problems, (ii) Where specialized techniques exist for solving particular problems, they are likely to out-perform GA's in both speed and accuracy of the final result. Another drawback of Genetic Algorithms is that they are not well suited to perform finely tuned search, but on the other hand they are good at exploring the solution space since they search from a set of designs and not from a single design. Genetic Algorithms are not well suited for fine-tuning structures which are close to optimal solutions [9]. Incorporation of local improvement operators into the recombination step of a Genetic Algorithm is essential if a competitive Genetic Algorithm is desired.

4 Memetic Algorithms

Memetic algorithms (MAs) are evolutionary algorithms (EAs) that apply a separate local search process to refine individuals (i.e improve their fitness by hill-climbing). Under different contexts and situations, MAs are also known as hybrid EAs, genetic local searchers. Combining global and local search is a strategy used by many successful global optimization approaches, and MAs have in fact been recognized as a powerful algorithmic paradigm for evolutionary computing. In particular, the relative advantage of MAs over GA is quite consistent on complex search spaces. Figure 22 shows one possible implementation of a Memetic algorithm based on the Genetic Algorithm introduced earlier in Section 3. We use a simple variation of the Fiduccia and Mattheyses (FM) heuristic [13]. The original FM heuristic has several passes after which the heuristic terminates as presented in Section 2. In the local optimization phase, a single pass is allowed, furthermore a restriction on the number of modules to be moved is set to a certain value. It is to be noted that if local optimization is not strong enough to overcome the inherent disruption of the crossover, more strong local optimization is needed.

4.1 Computational Results for MA

Table 4 shows the results obtained from the Memetic Algorithm. The first column in the table *MA-ii* is the direct application of local search on each chromosome in the population at only the initial stage. The secon column *MA-gi* is the direct application of local search on each chromosome in the population in every generation. It is clear that *MA-gi* performs better fine tuning and exploitation than *MA-ii* which only attempts to fine tune the search at an early stage. *MA-hi* is in affect the combination of *MA-ii* with *MA-gi* such that after an early exploitation of the landscape the system attempts to explore and exploit the solution space simultaneously. The results in the table indicate that the combination does not have a drastic affect on the final solution quality even though an improvement of 2-3% is achieved. The fourth column in the table *MA-ci* is the direct application of GRASP

```
        MEMETIC ALGORITHM
1. Encode Solution Space
2.(a) set pop_size, max_gen, gen=0;
  (b) set cross_rate, mutate_rate;
3. Initialize Population Randomly.
** Utilize GRASP to Construct Initial Population (MA-ci)
** Apply Local Search to Initial Population (MA-ii)
4. While(Gen < Gensize)
    Apply Generic GA
**      Apply FM Local Search to Population (MA-gi)
EndWhile /* end of a run */
** Apply Final Local Search to Best Chromosome (MA-fi)
```

Fig. 22. The Memetic Algorithm

to effectively construct good intial solutions for the Genetic Algorithm. The system achieves an improvement of 65% over *MA-ii* and 51% over *MA-gi* for the largest benchmark (ibm13). Experimental results indicate that less than 25% of the population should be injected with good initial solutions for *MA-ci* to perform well. The last column in the table *MA-ci-gi* is a combined *MA-ci* and *MA-gi* approach where good intial solutions are injected into the initial population followed by a balanced exploration (via crossover, mutation) and exploitation (via a single pass of local search) stage. It is quite evident that this Memetic Algorithm approach achieves the best overall results compared to the previously mentioned methods (i.e MA-ii, MA-gi and MA-hi). The overall improvement obtained (over MA-hi) for the largest circuits are: 61% for ibm07, 50% for ibm10 and over 66% for the largest benchmark ibm13.

5 Results & Analysis

In this section we will summarize the results obtained using (i) Local Search (ii) Genetic Algorithms (iii) Memetic Algorithm. Table 5 presents the results obtained by the three techniques mentioned above for four way partitioning. As can be seen in Table 5 the Memetic Algorithm obtains on average better solutions (cuts) than the Local Search technique. As the benchmarks increase in size the quality of solutions obtained using the local search technique deterioates. A comparison between the pure Genetic Algorithm and the Memetic Algorithm reveals the importance of embeding local search within GA to improve its performance. The affect of exploitation shows very clearly for the large benchmarks (ibm07, ibm10 and ibm13).

Table 4. Comparison of Several Memetic Algorithm Implementations

Circuit	MA-ii		MA-gi		MA-hi		MA-ci		MA-ci-gi	
	Cuts	CPU	Cuts	CPU	Cuts	CPU	Cuts	CPU	Cuts	CPU
Fract	47	24	37	24	41	24	35	24	35	24
Prim1	131	157	145	157	123	159	104	158	103	159
Struct	165	344	160	345	165	348	128	348	127	351
Ind1	100	409	97	412	97	414	91	413	90	416
Prim2	303	585	317	588	294	588	265	617	265	621
Bio	267	1120	249	1139	266	1139	234	1155	233	1147
Ind2	1035	2757	710	2841	710	2821	589	2861	587	2832
Ind3	1320	4627	1286	4702	1272	4672	1217	4847	1185	4837
Avq.s	1003	4777	936	4953	963	4804	885	5086	882	5019
Avq.l	999	6351	986	6457	979	6366	968	6386	965	6319
Ibm05	12502	8137	8424	8377	8384	8173	6236	8969	5158	8948
ibm07	18368	16939	12108	17138	12065	17218	8190	17863	6485	18096
ibm10	28765	30569	20239	30820	20296	31238	12307	33421	10119	34322
ibm13	35502	41186	25180	42066	24345	42650	12237	44220	8152	45438

Table 5. Comparison between LS, GA and MA

Circuit	Local Search		Genetic Algorithms		Memetic Algorithms		Improvement	
	Cuts	CPU	Cuts	CPU	Cuts	CPU	LS	GA
Fract	28	0.3	39	24	35	24	-20%	+10%
Prim1	148	2.7	145	156	103	159	+30%	+29%
Struct	195	6.4	161	344	127	351	+34%	+21%
Ind1	245	8.3	111	408	90	416	+63%	+18%
Prim2	636	13.3	325	581	265	621	+58%	+18%
Bio	532	45.8	266	1122	233	1147	+56%	+12%
Ind2	1759	143	1010	2778	587	2832	+66%	+41%
Ind3	1675	118	1337	4645	1185	4837	+29%	+11%
Avq.s	2151	309	986	4831	882	5019	+59%	+10%
Avq.l	2594	321	1002	6336	965	6319	+62%	+4%
Ibm05	8922	1618	11890	8158	5158	8948	+42%	+56%
ibm07	13527	4437	18183	16901	6485	18096	+52%	+64%
ibm10	22331	12855	29108	30507	10119	34322	+54%	+65%
ibm13	26710	16456	38186	41371	8152	45438	+69%	+78%

6 Conclusions

Memetic Algorithms (MAs) are Evolutionary Algorithms (EAs) that apply some sort of local search to further improve the fitness of individuals in the population. This paper provides a forum for identifying and exploring the key issues that affect the design and application of Memetic Algorithms. Several approaches of integrating Evolutionary Computation models with local search techniques (i.e Memetic Algorithms) for efficiently solving underlying VLSI circuit partitioning problem were presented. A Constructive heuristic technique in the form of GRASP was utilized to inject the initial population with good initial solutions to diversify the search and exploit the solution space. Furthermore, the local search technique was able to enhance the convergence rate of the Evolutionary Algorithm by finely tuning the search on the immediate area of the landscape being considered. Future work involves adaptive techniques to fine-tune parameter of the Genetic Algorithm and Local Search when combined to form a Memetic Algorithm. Balancing exploration and exploitation is yet another issue that needs to be addressed more carefully.

References

1. S. Areibi, M. Moussa, and H. Abdullah. A Comparison of Genetic/Memetic Algorithms and Other Heuristic Search Techniques. In *International Conference on Artificial Intelligence*, pages 660–666, Las Vegas, Nevada, June 2001.
2. S. Areibi. An Integrated Genetic Algorithm With Dynamic Hill Climbing for VLSI Circuit Partitioning. In *GECCO 2000*, pages 97–102, Las Vegas, Nevada, July 2000. IEEE.
3. S. Areibi and A. Vannelli. An Efficient Clustering Technique for Circuit Partitioning. In *IEEE ISCAS*, pages 671–674, San Diego, California, 1996.
4. S. Areibi and A. Vannelli. A GRASP Clustering Technique for Circuit Partitioning. 35:711–724, 1997.
5. P.K. Chan, D.F. Schlag, and J.Y. Zien. Spectral K-way Ratio-Cut Partitioning and Clustering. *IEEE Transactions on Computer Aided Design*, 13(9):1088–1096, 1994.
6. S. Dutt and W. Deng. VLSI Circuit Partitioning by Cluster-Removal Using Iterative Improvement Techniques. In *IEEE International Conference on CAD*, pages 194–200. ACM/IEEE, 1996.
7. C.M. Fiduccia and R.M. Mattheyses. A Linear-Time Heuristic for Improving Network Partitions. In *Proceedings of 19th DAC*, pages 175–181, Las Vegas, Nevada, June 1982. ACM/IEEE.
8. T. Feo, M. Resende, and S. Smith. A Greedy Randomized Adaptive Search Procedure for The Maximum Ind ependent Set. *Operations Research*, 1994. Journal of Operations Research.
9. D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Publishing Company, Inc, Reading, Massachusetts, 1989.
10. B.W. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell System Technical Journal*, 49(2):291–307, February 1970.

11. Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlog, Berlin, Heidelberg, 1992.
12. K. Roberts and B. Preas. Physical Design Workshop 1987. Technical report, MCNC, Marriott's Hilton Head Resort,South Carolina, April 1987.
13. L.A. Sanchis. Multiple-Way Network Partitioning. *IEEE Transactions on Computers*, 38(1):62–81, January 1989.
14. C. Sechen and D. Chen. An improved Objective Function for Min-Cut Circuit Partitioning. In *Proceedings of ICCAD*, pages 502–505, San Jose, California, 1988.

Part III

Methodological Aspects of Memetic Algorithms

Towards Robust Memetic Algorithms

Natalio Krasnogor[1]

Automated Scheduling, Optimization and Planning Group
School of Computer Science &IT
University of Nottingham
Nottingham, NG81BB, UK. Natalio.Krasnogor@Nottingham.ac.uk

Summary. This chapter reports the results of Multimeme algorithms that employ adaptive helpers. A Multimeme Algorithm resorts to a variety of local search neighborhoods for its local search stage allowing for a more robust global search. Each neighborhood is explored by an adaptive helper that allows non-improving moves that render the Memetic algorithm even more robust to deceptive local optima. We will report results on the use of a single adaptive helper Memetic algorithm for the Traveling Salesman Problem (TSP) and on adaptive helpers within a Multimeme algorithm for the TSP and Protein Structure Prediction Problem (PSP).

1 Introduction

Memetic algorithms are evolutionary algorithms that include, as part of the "standard" evolutionary cycle of crossover-mutation-selection, a local search stage. They have been extensively studied and used on a wide range of problems. Multimeme evolutionary algorithms were introduced by Krasnogor and Smith [28] and applied to OneMax,NK-Landscapes, TSP and two bioinformatic problems, Protein Structure Prediction and Protein Structure Alignment[23], [5] and [11],[24]. The distinction between Memetic and Multimeme Algorithms is that the former uses only one (usually complex) local search while the later employs a *set* of (usually simple) local searchers. Multimeme algorithms self-adaptively select from this set which heuristic to use for different stages of the search process. This kind of algorithm exploits features from Evolutionary Algorithms and Variable Neighborhood Search (by virtue of its multi-operator local search). In [23] we proposed two alternative ways an MA can achieve a more robust search:

- the MA can use several local searchers that explore the search space from complementary perspectives (i.e., the MA is a Multimeme algorithm) or
- the MA can somehow reduce its "greediness" by either not using elitist replacement strategies or by exploiting operators that can lead to deteri-

orated points from which progress can be achieved at a later stage of the search.

The rationale for these two criteria are studied in detail in [23].

In this paper we will show one possible realization of these two points for the implementation of Multimeme algorithms[28]. The paper is divided in 3 parts. In the first part we will introduce a formalism that will help us motivate the design of our algorithms. In the second part we describe a Monte Carlo helper that by adapting the temperature at which it performs its search strategically oscillates between periods of exploitation of solutions and phases of exploration. The adaptive helper effectively reduces the greediness of the MA and in turn helps to sustain population diversity for longer periods of time. Part 3 will see the integration of the two design issues mentioned above by employing several local searchers, where each one of them is realized as an adaptive helper.

2 Search in a Multidimensional Landscape

This section describes the concept of local optima and basins of attractions for a multidimensional fitness landscape. This is done to motivate the particular choices we made to address the integration of several local searchers and the reduction of the "greediness" of our MAs (while preserving the benefits of local search and maintaining population diversity).

2.1 Local Optima, Basins of Attraction and the Dynamics of Search

In [2] a formalism is introduced to describe the topology of multidimensional fitness landscapes in the context of molecular dynamics. We will briefly describe the main ideas of this approach, adapted to memetic search on discrete spaces. Later we will use this formalism to illustrate the utility of the adaptive helpers that are going to be introduced in this paper.

Suppose that a memetic algorithm is trying to optimize a multidimensional function Φ that takes as argument objects from a discrete space S and maps them into a scalar field, i.e. $\Re$. We can assume that Φ will have several local optima. These local optima can be described as a discrete set, M, indexed by α. This set can be obtained if we fix an operator o by direct minimization from a point $s \in S$ to a point $\alpha \in S$ which is a local optimum for Φ is found. Let's call this mapping $M^o(s)|S \mapsto \{\alpha\}$. Each α has an associated basin of attraction $R(\alpha)$. The basin of attraction for α is composed of all those points in S for which α is its nearest local optimum (by means of the operator o). That is, $S = \bigcup R(\alpha)$ for every α a local optimum.

Having the set of local optima for Φ is not very informative, because we lack the information on how those basins are interconnected, which is the internal

constitution of each basin, the size of the basins, which fitness barriers separate them, etc. One way around this is to extend the mapping M^o by redefining R as $R^\epsilon(\alpha') = \bigcup R(\alpha)$. That is, $R^\epsilon(\alpha')$ is the union of all the $R(\alpha)$ basins that are connected by fitness barriers lower than ϵ. In this sense $R^\epsilon(\alpha')$ is a super basin that groups together all those basins $R(\alpha)$ such that a path from $v \in R(\alpha_1)$ to $u \in R(\alpha_2)$ never crosses a barrier higher than ϵ (this is for any $R(\alpha_1), R(\alpha_2) \subset R^\epsilon(\alpha')$).

In turn α' is defined as the lowest local minimum in $R^\epsilon(\alpha')$, that is, $\alpha' = min\{\alpha | \alpha \in R(\alpha), R(\alpha) \subset R^\epsilon(\alpha')\}$. By considering different values of ϵ (i.e. fitness barriers of different heights) we can obtain a more detailed description of the multidimensional landscape (under the view of operator o). A hypothetical multidimensional fitness landscape represented by the iterative mapping M^o_ϵ for various ϵ's is depicted in figure 1. This graph was called "Disconnectivity Graph", DG for short, in [2]. In the figure, five iterations of the map can be seen for five different ϵ values . Each value corresponds to a horizontal level (marked as -2,-1,0,1 or 2 in the picture) with the root of the tree, node A, the largest basin calculated with the highest ϵ for a fitness barrier. Basin A represents all the basins of attractions of local optima that are connected by walks in the landscape that never cross barriers higher then ϵ_2 (the ϵ corresponding to level 2). The higher the value for ϵ_i (for some level i in the graph) the broader the features the graph associated with the map will be able to show. To gain higher details of the multidimensional fitness landscape associated with Φ, the map can be iterated with smaller and smaller ϵ's. In one extreme, the highest possible barrier between any two optima is the difference between the global optimum and the worst local optimum. In this case, the graph will have a unique vertex representing all the possible basins. On the other hand, when ϵ takes the smallest possible value, i.e. the difference between the two closest local optima value, the graph will display information about every single barrier. This, in turn, makes the concept of super basins and long range topological features associated with Φ disappear. For the disconnectivity graph to be of use it needs to include a range of ϵ values within the extreme cases described above.

2.2 Dynamics of Search

In general, a memetic algorithm will start searching with a population representing several of the vertices of the DG. The population will eventually collapse to a certain basin of attraction represented by a particular α in agreement with theoretical analysis like those done in [23]. For example, our MA can start in vertex A, move down to B and C and after a number of generations down to D. If the MA were to explore the unnamed basin connected to D by C, it will need to bypass a fitness barrier given by ϵ_0(associated to level 0 to the right of the graph). Equivalently, if the memetic search were trapped in basin B, and we hope that it will reach the global optima in H, then we need to provide the MA with a mechanism to jump across the fitness barrier

with an ϵ_2. That is, move from vertex A move down to E and eventually move to F,G and then H.

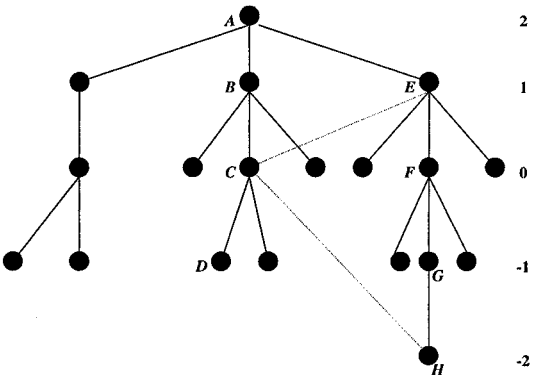


Fig. 1. *Schematic representation of a landscape's attraction basins.*

As we can see from this example it will be desirable to provide the MA with a mechanism that can detect when the MA is trapped in a certain vertex of DG and jump over fitness barriers. We will introduce next one possible mechanism to accomplish this. While our approach is simple and proved effective we are not claiming here that this is the only, or the best, way to enforce a dynamic that is able to cross barriers in the landscape explored. Other studies relating operators, problems and instances to the barriers to be jumped can be worth doing in order to further specialize the method we are proposing.

3 An Adaptive Helper for TSP and PSP

In this paper we review the hybridization scheme for a Memetic Algorithm (MA) based on an *adaptive* helper that uses statistics from the GA's population. This adaptive helpers were introduced in [27] and [23]. We extend the results of those studies with new experiments and in a later section we will integrate those helpers within a Multimeme algorithm.

The MA is composed of two optimization processes, a Genetic Algorithm and a helper which is a Monte Carlo method (MC). In contrast with other GA-Monte Carlo hybridized memetic algorithms, in this work the MC stage serves two purposes. First, when the population is diverse, it acts like a local

search procedure and second, when the population converges, its goal is to diversify the search. To achieve this, the MC is adaptive based on observations from the underlying GA behavior; the GA controls the long-term optimization process. We present results on the application of this approach to the TSP problem. These results are going to be extended in subsequent sections for Protein Structure Prediction Problem. Moreover, we will integrate the adaptive helpers with Multimeme Algorithms and present results. The adaptive helper we introduce here, by accepting moves in the search space that deteriorate the value of the objective function, reduces the overall greediness of the MA. Additionally, it helps keeping the diversity of the population at higher values and prevents premature convergence.

3.1 Introduction and Previous Related Works

In [27] and [23] we reviewed several applications of evolutionary algorithms (in particular MAs) for the TSP. We refer the interested reader to [19] and [21] for a large collection of papers and instances.

The helper used later employs a Metropolis criterion to accept the candidate solution. Thus, we will briefly discuss related papers that came from the Simulated Annealing (SA) literature. Those papers are related to our approach in the sense that they are not standard annealing schemes but rather "multi-agent" annealing heuristics. We say multi-agent because they keep a population of solutions that cooperate through information exchange during the optimization process. In [1] Aarts et al. propose an architecture where the optimizing individuals are arranged in a hierarchical structure of increasing temperature. Every so often, individuals receive solutions from their topological neighbor that uses the highest temperature. They evaluate the solution received with the Boltzmann criteria and decide to keep their own or to jump to the neighbors' solution. The authors also propose an alternative architecture where every agent works with the same starting solution across the hierarchy of temperatures. As soon as an agent accepts a new solution, every other agent update its own with the newly created point. Another multi-agent simulated annealing is that of Lee and Lee [17] where the individuals compare their solutions and all jump to the one with the better cost. Two memetic algorithms for the TSP that used simulated annealing as an aid to the search are those presented in [18] and [4]. In the first paper the authors introduce an MA that uses a single step of simulated annealing as its mutation stage and selection was itself implemented as a Boltzmann criterion. The annealing schedule was the same for all the individuals. Boseniuk et al. presented a simple implementation of an MA with $D = 4$ (as per [23]) where the local search phase is SA. In K.D. Boese's Ph.D. thesis [3], extensive theoretical and experimental analyses of optimal infinite and finite time adaptive annealing schedules were done for a model of iterative global search by means of multi-agents. The instances used were small (i.e. 6 to 8 cities for the TSP) but the results were conclusive: The optimal annealing schedules he arrives at were

those which oscillate between temperature values of 0 and infinity or where the temperature was allowed to rise. Those results hold true for a number of combinatorial optimization problems. In that work we can read:

> Our analysis of small instances ... suggests that the preocupation with optimality of s_M in the literature has incorrectly led to the assumption that cooling strategies are best.

and in a subsequent chapter the author continues:

> ... we have computed the optimal annealing temperature schedules for small combinatorial problems; these schedules can resemble multi-start, with alternating periods of greedy descent and randomization (corresponding to annealing at zero and infinite temperatures)...

We can learn two lessons from the literature on Multi-Agent annealing schemes and MAs for TSP:

- In the memetic algorithm literature, keeping population diversity while using local search together with a GA is always an issue to be addressed, either implicitly or explicitly. Usually this takes the form of complex operators or sophisticated book-keeping and/or guiding strategies.
- In the multi-agent literature, different annealing schemes were proposed together with different ways of sharing either solutions, annealing schedules or temperatures. Several inter individual coupling mechanisms were investigated.

3.2 The Adaptive Memetic Algorithm

Our purpose here is to show the potential for both search and diversity in our approach. In this MA, the temperature reflects the state of the global search. As explained before, when fitnesses across the population converge we will assume that the MA is trapped in a certain vertex of the disconnectivity graph. The temperature is going to rise (by design) leading to a more explorative global search that will allow the MA to jump over fitness barriers and eventually move to a different vertex of DG. Once the fitnesses in the population spread, the temperature will anneal, exploiting the solutions held by each individual. In a given generation, all members of the population use the same temperature.

The reader should note that it is not the goal of this paper to develop a specialized TSP solver. We have used very naive and generic genetic operators (i.e. crossover or mutation). The local search move that was employed is depicted below:

```
MA:
Begin
  Initialize population Parents;
  Repeat Until ( Finalization_criteria_met ) Do
    For indip := first in Parents To last in Parents Do
      Local_Search(indip);
    endDo
    mating_pool := Select_mating(Parents);
    offsprings := Cross(mating_pool);
    Mutate(offsprings);
    Parents := Select(Parents + offsprings);
  endDo
End.
```

```
Local_Search(indip):
Begin
  /* This is a Maximizing process */
  prevFitness = fitness(indip);
  Modify(indip);
  nFitness = fitness(indip);
  If (prevFitness < nFitness) Then
    Accept configuration;
  endIf
  Else
  deltaE = prevFitness − nFitness;
  threshold = e^(−k* deltaE / temperature);
  If (random(0,1) < threshold) Then
    Accept configuration;
    /* even if worse than the previous one */
  endIf
  Else
  Reject changes;
End.
```

We describe next a general Adaptive Memetic Algorithm where the goal is to maximize the fitness[1]. In the basic algorithm used (shown above to the left), the `Select(...)` procedure is a $(\mu + \lambda)$ or a (μ, λ) selection strategy, representing two extremes of selection pressure, with the *plus*-strategy having the highest pressure and the *comma*-strategy the lowest. We tried these two scenarios because we want to explore not only final tour length but also population diversity. We wanted to compare how well our adaptive memetic algorithm performs under these two extremes.

[1] For the TSP the length of a tour was multiplied by -1.

The local search/diversification procedure in the pseudo-code above sets the temperature to $\frac{1}{|maxFitness - avgFitness|}$. It then applies to each member of the population the `ApplyMove(...)` operator (see pseudo-code above). Note that `Modify(...)` can be any local search move (e.g. a 2swap, city insertion,k-exchange, etc.). The adaptation of the local search to either an exploitation or exploration behavior is governed by the temperature parameter. As it was mentioned before, the entire population shares the same temperature. This temperature determines the extent to which decreasing fitness moves will be allowed. As the spread of fitnesses within the population converges the temperature rises. As a consequence, each individual in the population will be more likely to be changed, exploring the search space. The extent by which a worsening move will be accepted is a function of both the individual fitness (i.e. its location in the search space) and the global state of the population (i.e. measured by the temperature). Eventually, the fitnesses will spread, lowering the population temperature. We prevent the modification by local-search of the best individual, hence the overall best fitness in each generation is always maintained.

3.3 Experimental Method and Results

For our experiments we used a population of 50 individuals. Crossover, mutation and local search were applied with probability 0.8, 0.05 and 1.0 respectively. We have chosen two instances from TSPLIB[21] to test our approach, eil76.tsp and lin318.tsp. These instances are of no particular difficulty and of relatively small size. We run 30 simulations under two different selection strategies, a $(50, 50)$ and a $(50 + 50)$ strategy. We test our algorithms against four other algorithms, all of them sharing either of the selection strategies:

(1) A standard Genetic algorithm (GA) with no local search of any kind, which constitutes the basis for constructing all the other algorithms tested (see section 3.3 for details). (2) A Hill Climber Memetic Algorithm (HC) which uses as local search the `two_exchange(...)` move but only accepts improvements. (3) A Boltzmann Hill Climber memetic algorithm (BHC) which uses the same decision procedures as the adaptive memetic algorithm but with a fixed temperature. The temperature was set to be the average (*a posteriori* temperature employed by the our proposed adaptive MA in one of its runs. (4) A Linear annealing memetic algorithm (LMA) which uses the Boltzmann criteria to accept/reject moves. In this case the temperature was set at the beginning of the run to a value that was linearly annealed during the run.

Each algorithm was run for 2000 generations, except the GA which was given 6000 generations to compensate for the use of extra fitness evaluations by the local searcher[2]. To compare the quality of our MA against the other four alternatives we look at two measures, the quality of the best individual

[2] In fact, the GA employed more fitness evaluations than all the other MAs.

at the end of the run and the diversity of the population at that time. The quality was equivalent to the tour length and the diversity the number of different fitnesses found in the population divided by the population size. We performed an ANOVA test on the averages of these measurements over the 30 runs for the 5 algorithms. A total of 300 runs were analyzed [3]. More details for replicating the experiments can be found in [23].

Tables 1 and 2 summarize the results obtained.

From table 1 (left) we can see that the proposed MA achieves better final tour length than the standard GA, the GA with a Hill climber (HC), the GA with a Boltzmann Hill Climber(BHC) and the linear annealed MA (LMA). These anova results are of statistical significance with a p-value of 0.01. The diversity results in table 2 show that the adaptive approach is capable of maintaining the diversity of the population on higher values than the other four algorithms. The differences are of statistical significance.

As mentioned in the introduction, the use of local search within a GA usually causes a premature convergence in the search space, hence maintaining a diverse population is crucial[4]. It can be seen from the lower/upper diagonal of tables 1 and 2 that in most cases, when an algorithm beats another algorithm in one table, it beats (or is at least equivalent to) the same one in the other table as well.

We performed similar tests on a different mating and selection strategy. While the previous experiments used a tournament size of 2 and a $(50, 50)$ or $(50 + 50)$ strategy, this new experiment employed a $(50, 200)$ and a $(50 + 200)$ strategy with a tournament size of 4, effectively producing a very high selection pressure during both reproduction and survival phases. Tables 3 and 4 present the results obtained for instance eil76.tsp and lin318.tsp respectively. The only notable difference with the previous experiments is the instance lin318 results for the $(50 + 200)$ case. The selection pressure was probably too high and hence no difference (with statistical significance) was found among the 4 MAs and the GA.

In order to examine further the conduct of our approach under different operators and representations we changed the encoding from the one described above to a permutation encoding. We also used a PMX crossover keeping all the other parts of the 5 algorithms unmodified. Again, 30 runs of each algorithm under the two selection schemes were executed for 2000 (6000 in the GA case) generations.The results obtained were consistent with those shown above. The adaptive MA is better in both final tour length and diversity of the final population with a statistical significant difference (not shown here).

[3] 30 runs per each one of the 5 algorithms per each one of the two selection strategies.

[4] This is of particular importance on MA applied to dynamic optimization and multi-objective optimization.

Table 1. *Summary of anova analysis for tour length under the* $(50, 50)-$*strategy (left) and the* $(50 + 50)-$*strategy (right): + denotes that the algorithm that names the row achieves a longer tour than the one that names the column, - denotes that the algorithm that names the row achieves a shorter tour than the one that names the column, - or + with * denotes statistical significance up to at least a p-value of 0.05*

(50,50)-Strategy						(50+50)-Strategy				
Algorithms	GA	HC	BHC	LMA	MA	GA	HC	BHC	LMA	MA
GA		-	+	+	+*		-	+*	+*	+*
HC	+		+	+	+*	+		+*	+*	+*
BHC	-	-		+	+*	-*	-*		-*	-
LMA	-	-	-		+*	-*	-*	+*		+*
MA	-*	-*	-*	-*		-*	-*	+	-*	

Table 2. *Summary of anova analysis for population diversity under the* $(50, 50)-$*strategy (left) and* $(50 + 50)-$*strategy (right)*

(50,50)-Strategy						(50+50)-Strategy				
Algorithms	GA	HC	BHC	LMA	MA	GA	HC	BHC	LMA	MA
GA		+	-*	+	-*		=	-*	=	-*
HC	+		-*	-	-*	=		-*	=	-*
BHC	+*	+*		+*	-*	+*	+*		+*	+*
LMA	-	-	-*		-*	=	=	-*		-*
MA	+*	+*	+*	+*		+*	+*	-*	+*	

Table 3. *Summary of anova analysis for tour length under the* $(50, 200)-$*strategy (left) and the* $(50 + 200)-$*strategy (right) with tournament size of 4. (Instance Eil76.tsp)*

(50,200)-Strategy						(50+200)-Strategy				
Algorithms	GA	HC	BHC	LMA	MA	GA	HC	BHC	LMA	MA
GA		+	+*	+*	+*		-	+*	-	+*
HC	-		+	+	+*	+		+*	-	+*
BHC	-*	-		+	+*	-*	-*		-*	=
LMA	-*	-	-		+*	+	+	+*		+*
MA	-*	-*	-*	-*		-*	-*	=	-*	

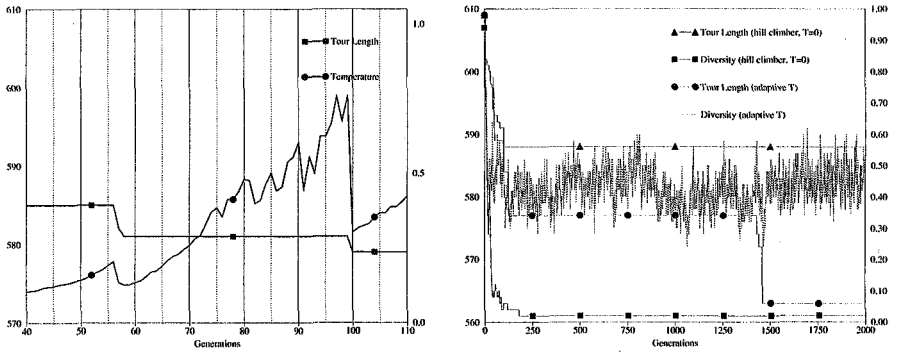
Table 4. *Summary of anova analysis for tour length under the* $(50, 200)-$*strategy (left) and the* $(50 + 200)-$*strategy (right) with tournament size of 4. (Instance lin318.tsp)*

(50,200)-Strategy						(50+200)-Strategy				
Algorithms	GA	HC	BHC	LMA	MA	GA	HC	BHC	LMA	MA
GA		+	+*	+*	+*		-	+	+	+
HC	-		+*	+*	+*	+		+	+	+
BHC	-*	-*		-	+	-	-		-	-
LMA	-*	-*	+		+*	-	-	+		=
MA	-*	-*	-	-*		-	-	+	=	

3.4 Adaptive MA Behavior

Figure 2(a) we show a plot of the evolution of the fitness (tour length) of the best individual in the population and the population temperature as a function of time (generations) for a randomly selected run. Only the time window between generations 40 to 110 is shown. It is easy to visually inspect the graph to note that all of the major fitness transitions are preceded by a peak in temperature and a subsequent fall in its value. Once a new best fitness

is established in the population the temperature starts to rise. At a certain point in time a new "discovery" is made by the MA and the temperature cools down. The behavior of our MA closely follows that of Boese's optimal schedules[3]; this is done in an adaptive fashion and for TSP instances that are one and two order of magnitude bigger than the ones he used.



(a) (b)

Fig. 2. *Eil76.tsp: Tour length and Temperature (a), Tour Length and Diversity (b) as a function of generations.*

In figure 2(b) we plot the average fitness of the population and diversity for both the Adaptive and Hill Climber (T=0) MA. The diversity achieved a much higher level in the adaptive searcher while it was rapidly lost in the hill climber. Furthermore, the final tour length was much better with the adaptive approach.

We suspect that the oscillations in the temperature approximately follow a power law, which (in the words of Coveney and Highfield[8]) represents

"The fingerprint of self-organized criticality"

We calculated the Fourier transformation of the temperature time series and we plot in figure 3 a $log - log$ plot of amplitudes versus frequencies. Even though the time series is noisy it is possible to see that big amplitudes correspond with low frequencies following a linear trend in the graph. In terms of the search process this means that we can expect to have, for example, a change in temperature of magnitude 10, 10 times more frequently than a change of magnitude 100.

4 Multimeme Algorithms with Adaptive Helpers

In a Multimeme Algorithm an individual is composed of its genetic material (that represents the solution to the problem being solved, e.g. a candidate

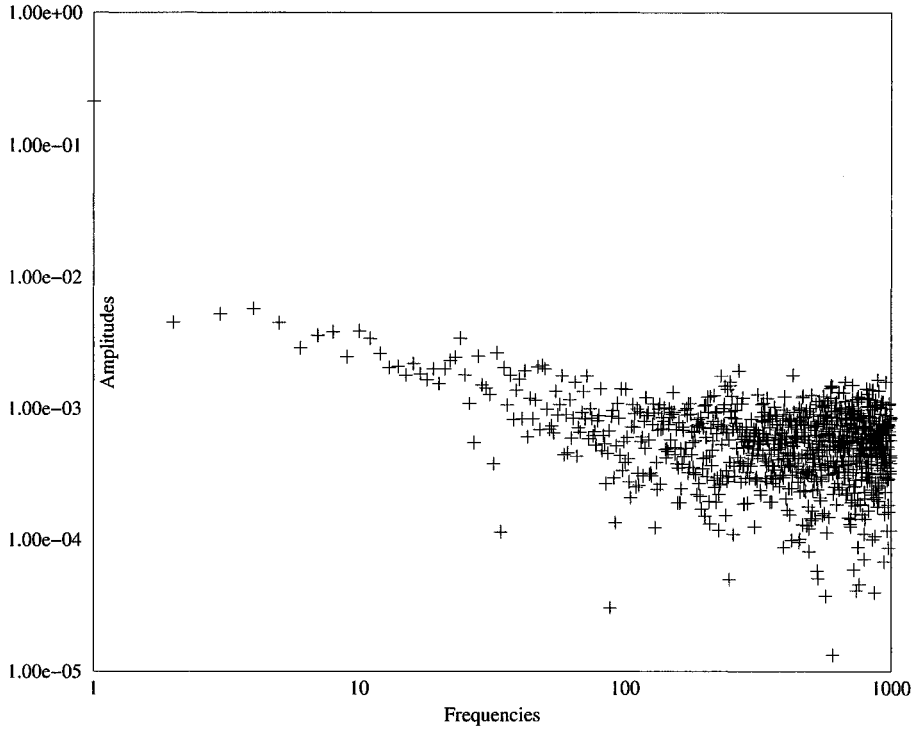


Fig. 3. *Fourier Analysis of the temperature time series.*

protein structure or a TSP tour in this paper) and its memetic material (that defines the kind of local searcher to use, e.g. alternative operators to improve a protein structure or a TSP tour). The mechanisms of genetic exchange and variation are the usual crossover and mutation operators but tailored to the specific problem one wants to solve. Memetic transmission is effected using the so called Simple Inheritance Mechanism (SIM)[28] where a meme (local searcher) L, at time $t-1$ that is carried by parent j (or k), will be transmitted to the offspring i if that meme is shared by all the parents. If they have different memes, L is associated to the fittest parent. Otherwise, when fitnesses($F(\cdot)$) are comparable and memes different, a random selection is made. The rationale is to propagate local searchers (i.e. memes) that are associated with fit individuals, as those individuals were probably improved by their respective memes. During mutation, the meme of an individual can also be overridden and a local searcher assigned at random (uniformly from the set of all available local searchers) based on the value of the innovation rate parameter. This is done to introduce novelty in the local search phase of the MMA.

In previous sections we showed how evolutionary search can be made more robust by the use of adaptive helpers that are capable of selecting (for short periods of time) worst points of the search space. Furthermore, these adaptive helpers are able to maintain population diversity and allow the search process to jump over deep minima and navigate wide neutral plateaus.

In this part of the work we apply SIM to learn which is the best **adaptive** meme to employ during different stages of the search, effectively integrating the adaptive Monte Carlo helpers with the Multimeme algorithm. The case studies are the TSP and the PSP. By combining both approaches, a more competent[22] memetic algorithm is achieved.

As was explained before, adaptive helpers were allowed to jump over fitness barriers as those depicted in Figure 1. The inclusion by a multimeme algorithm of several helpers can further aid that process. If the multimeme algorithm is trapped, let us say, in vertex E of Figure 1, then it can follow either F,G and finally H, or (by virtue of its several neighborhoods search) jump straight to vertex C, without using the higher basins A and B, and from C going down towards H (shown in the picture with dotted lines).

While a multimeme algorithm based on SIM enables one to view the search space through the looking glass of several local searchers, it does not facilitate jumping over basins that are common to all of these searchers. On the other hand, an adaptive helper can jump over basins (effectively backtracking in the fitness landscape) but, as the theoretical analysis of [23] exemplified for the case of MAs for the TSP, sometimes even achieving a local optimum can take exponential time. Experimental evidence of a very long convergence to a local optimum was also found for the PSP[20].

By extending the capabilities of a multimeme algorithm (which searches through various neighborhoods) with adaptive helpers (that are capable of jumping over basins of attraction or navigate through wide neutral plateaus) we produce a more robust metaheuristic that benefits from the characteristics of both approaches.

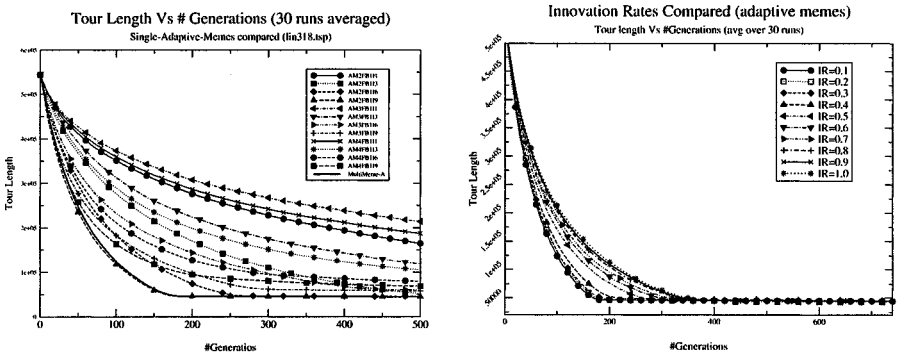
4.1 TSP, Experiment Description

As in our previous investigation[13] (where only static helpers were studied) we used 24 different memes; each meme defines the acceptance strategy, the underlying basic move and the number of iterations to use during the local search stage. There were two acceptance strategies, namely *first-improvement* and *best-improvement*. For static helpers as those used in [13] where only neighbor solutions with improved fitness are considered, the standard semantic is assumed for a *first-improvement* or a *best-improvement* strategy. However, these need to be changed in our present case to reflect the fact that under the adaptive scenario chosen, deteriorating moves are allowed. In both cases, where no *first-improvement* or *best-improvement* was found (no better tour or better protein structure was reached from the starting solution), a lower quality tour/protein structure was constructed and accepted (or rejected) based

on the Boltzmann criterion. The temperature was set to reflect the state of the population fitnesses spread as explained previously and also reported in [27] and [23].

Lets consider first the TSP case. Three basic moves were employed, namely $2-exchange$, $3-exchange$ and $4-exchange$. The final property of a meme was the number of times the acceptance strategy was going to be iterated employing the basic move, in our experiment the options where $1, 3, 6$ or 9 iterations. For all the experiments run, the probability of mutation was 0.4, the probability of crossover was 0.6 and the innovation rate was set to 0.125. The crossover used was DPX, and the mutation operator was the double-bridge move. The underlying GA was a generational GA with a $(50, 200)$ strategy using a tournament size of 4. The architecture of the MA was accordingly to [23] $D = 4$. That is, local search was executed independently of mutation and crossover in a separate stage. The probability of local search (expressing the meme) was 1. The encoding used was a permutation encoding.

We first ran a set of experiments (one for each of the 24 memes), each consisting of 30 trials, where the whole population used the same meme. That is, each individual was statically linked to a unique helper. The goal of this experiment was to obtain a ranking of memes for the different instances.



(a) (b)

Fig. 4. *In (a) relative performance of a memetic algorithm using adaptive memes can be seen as a function of the generation number for instance lin318.tsp. The curve for the associated multimeme algorithm is also shown and it converges toward the best meme. In (b) comparison of different innovation rates for the multimeme adaptive approach.*

A t-test and an ANOVA analysis of the average over 30 runs for the best tour in each experiment shows that the final values of best fitness are (with 95% confidence level) different.

In the table 5 and 7 we show the rankings obtained for a memetic algorithm that uses one of the adaptive memes on instances lin105.tsp and lin318.tsp.

Table 5. *Summary of anova analysis for tour length on instance lin105.tsp. The meaning of symbols is as in previous tables. Memes are adaptive helpers.*

Alg.	AM2I1	AM2I3	AM2I6	AM2I9	AM3I1	AM3I3	AM3I6	AM3I9	AM4I1	AM4I3	AM4I6	AM4I9
AM2I1	.	+*	+*	+*	-*	-*	-	-	-*	-*	-*	-*
AM2I3	-*	.	+	+	-*	-*	-*	-*	-*	-*	-*	-*
AM2I6	-*	-	.	+	-*	-*	-*	-*	-*	-*	-*	-*
AM2I9	-*	-	-	.	-*	-*	-*	-*	-*	-*	-*	-*
AM3I1	+*	+*	+*	+*	.	+*	+*	+*	-*	+	+*	+*
AM3I3	+*	+*	+*	+*	-*	.	+*	+*	-*	-*	+	+
AM3I6	+	+*	+*	+*	-*	-*	.	+	-*	-*	-*	-*
AM3I9	+	+*	+*	+*	-*	-*	-	.	-*	-*	-*	-*
AM4I1	+*	+*	+*	+*	+*	+*	+*	+*	.	+*	+*	+*
AM4I3	+*	+*	+*	+*	-	+*	+*	+*	-*	.	+*	+*
AM4I6	+*	+*	+*	+*	-*	-	+*	+*	-*	-*	.	+
AM4I9	+*	+*	+*	+*	-*	-	+*	+*	-*	-*	-	.

Table 6. *Summary of anova analysis for tour length on instance lin318.tsp. Memes are adaptive helpers*

Alg.	AM2I1	AM2I3	AM2I6	AM2I9	AM3I1	AM3I3	AM3I6	AM3I9	AM4I1	AM4I3	AM4I6	AM4I9	AMulM
AM2I1	.	+*	+*	+*	-*	+*	+*	+*	-*	+*	+*	-*	+*
AM2I3	-*	.	+*	+*	-*	-*	-*	-*	-*	-*	-*	-*	+*
AM2I6	-*	-*	.	+*	-*	-*	-*	-*	-*	-*	-*	-*	+*
AM2I9	-*	-*	-*	.	-*	-*	-*	-*	-*	-*	-*	-*	-
AM3I1	+*	+*	+*	+*	.	+*	+*	+*	-*	+*	+*	+*	+*
AM3I3	-*	+*	+*	+*	-*	.	+*	+*	-*	+*	+*	+*	+*
AM3I6	-*	+*	+*	+*	-*	-*	.	+*	-*	-*	-*	-*	+*
AM3I9	-*	+*	+*	+*	-*	-*	-*	.	-*	-*	-*	-*	+*
AM4I1	+*	+*	+*	+*	+*	+*	+*	+*	.	+*	+*	+*	+*
AM4I3	-*	+*	+*	+*	-*	-*	+*	+*	-*	.	+*	+*	+*
AM4I6	-*	+*	+*	+*	-*	-*	+*	+*	-*	-*	.	-*	+*
AM4I9	+*	+*	+*	+*	-*	-	+*	+*	-*	-*	+*	.	+*
AMulM.	-*	-*	-*	+	-*	-*	-*	-*	-*	-*	-*	-*	.

Table 7. *p-values of an anova analysis are shown. The final tour length of an adaptive multimeme algorithm(left column) and a static multimeme algorithm(top row) were compared for different innovation rates. When a - (+) appears in a cell then the algorithm naming the row is better(worse) than the algorithm naming the column and the associated p-value is shown with a precision of 4 decimals.*

Alg.	S-IR=0.1	S-IR=0.2	S-IR=0.3	S-IR=0.4	S-IR=0.5	S-IR=0.6	S-IR=0.7	S-IR=0.8	S-IR=0.9	S-IR=1.0
A-IR=0.1	0.0000(-)	0.0000(-)	0.0000(-)	0.0000(-)	0.0000(-)	0.0000(-)	0.0000(-)	0.0000(-)	0.0000(-)	0.0000(-)
A-IR=0.2	0.0000(-)	0.0000(-)	0.0000(-)	0.0000(-)	0.0000(-)	0.0000(-)	0.0000(-)	0.0000(-)	0.0000(-)	0.0000(-)
A-IR=0.3	0.0000(-)	0.0000(-)	0.0000(-)	0.0000(-)	0.0000(-)	0.0000(-)	0.0000(-)	0.0000(-)	0.0000(-)	0.0000(-)
A-IR=0.4	0.0038(-)	0.0014(-)	0.0000(-)	0.0000(-)	0.0153(-)	0.0244(-)	0.0001(-)	0.0088(-)	0.0003(-)	0.0071(-)
A-IR=0.5	0.0000(-)	0.0000(-)	0.0000(-)	0.0000(-)	0.0001(-)	0.0001(-)	0.0000(-)	0.0000(-)	0.0000(-)	0.0000(-)
A-IR=0.6	0.0002(-)	0.0000(-)	0.0000(-)	0.0000(-)	0.0010(-)	0.0018(-)	0.0000(-)	0.0005(-)	0.0000(-)	0.0004(-)
A-IR=0.7	0.0000(-)	0.0000(-)	0.0000(-)	0.0000(-)	0.0000(-)	0.0000(-)	0.0000(-)	0.0000(-)	0.0000(-)	0.0000(-)
A-IR=0.8	0.0008(-)	0.0003(-)	0.0000(-)	0.0000(-)	0.0075(-)	0.0078(-)	0.0000(-)	0.0033(-)	0.0001(-)	0.0024(-)
A-IR=0.9	0.0001(-)	0.0000(-)	0.0000(-)	0.0000(-)	0.0012(-)	0.0013(-)	0.0000(-)	0.0005(-)	0.0000(-)	0.0003(-)
A-IR=1.0	0.0001(-)	0.0000(-)	0.0000(-)	0.0000(-)	0.0008(-)	0.0008(-)	0.0000(-)	0.0003(-)	0.0000(-)	0.0002(-)

Table 7 demonstrates that the adaptive versions of the single meme algorithms are better than the static counterparts.

In Figure 4(a) the performance of the multimeme adaptive algorithm is plotted against the performance of the single meme MAs. As was reported in an earlier work [13] where the best static meme was correctly learned, the Multimeme Algorithm can also successfully track the best adaptive meme for this instance. In table 7 we can see that the Multimeme Algorithm is superior to all the single meme algorithms, except for the one that uses only the best meme for this instance (in this case $A - M2FB1I9$). However the difference between the best possible single meme algorithm and the Multimeme algorithm is not of statistical significance[5].

To further investigate our algorithms behaviour we conducted experiments for each of the possible innovation rates in the range $[0.1, 1.0]$. The results of plotting the tour length obtained for every generation is given in Figure 4(b).

As it is evident from figure 4(b), low innovation rates produce better performance than higher ones. In particular, the innovation rate value of 1.0 produces the worst performance. As an innovation rate of that value indicates a random selection of the local search neighborhood at each generation, we can conclude that it is indeed profitable to exploit SIM and learn to use only the right neighborhoods. More extensive analysis of this and other experiments can be found in [23],[13].

The evolutionary activity of the memetic system (as defined in [13]) is depicted in figure 5.

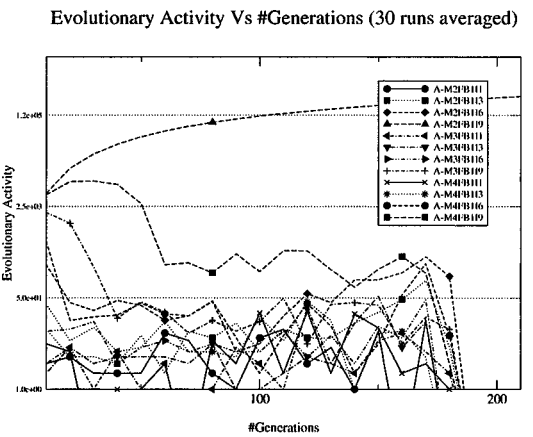


Fig. 5. *Evolutionary Activity of the memetic algorithm for instance lin318.tsp using adaptive memes. Zoom on the initial 200 generations.*

[5] And obviously it is very hard to predict *prior* to running experiments which is the best single meme for a given instance.

The most conspicuous curves are those associated with memes $A - M2FB1I9, A - M2FB1I6, A - M3FB1I9$ and $A - M4FB1I9$. These memes are amongst the most highly ranked in table 7.

4.2 Adaptive Multimeme Algorithm For The Protein Structure Prediction Problem

The construction of highly effective algorithms for solving structure prediction on simplified models, e.g. the HP model, is essential if we hope to target the structure prediction of real life proteins that cannot be solved by homology or threading methods. Several of the most successful methodologies employed during the last two Critical Assessments of Structure Prediction, CASP3 and CASP4, employed one or more simplified models for sampling and optimizing structures embedded in different lattices and measuring them with simplified energy potentials [23],[7]. It is evident then that any improvement on the optimization of lattice-based structures will be welcomed by the scientific community.

In the next section we will integrate the adaptive version of the protein structure memes into a multimeme algorithm for predicting protein structure. For a more extensive investigation on the use of the algorithms proposed here the reader can refer to [11] and [23].

4.3 The Experiments

As we did for the TSP, we want to elucidate the behavior of SIM under the presence of adaptive helpers, this time for the PFP. In these experiments we executed 10 runs for each innovation rate value (IR). The underlying GA is implemented as in [23], where the replacement strategy was a $(500, 1000)$ strategy with tournament selection of size 2. Crossover probability was 0.8 and that of mutation 0.3. Crossover was standard two-point crossover and mutation was one point mutation. Each run was executed for 200 generations which assured convergence. Solutions were encoded using internal coordinates in the relative encoding. We conducted experiments for each of the possible innovation rates in the set $\{0.0, 1.0E-5, 1.0E-4, 1.0E-3, 0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$.

In table 9 we can see the number of times the optimum was reached for instance I15 of length 24 and with a maximum number of bonds of 17 (minimum energy of -17), together with the average time needed to achieve those values.

Table 8 presents the number of optima reached with the different innovation rates together with the mean first hitting time of those optima.

In table 11 the mean first hitting times and number of optima reached with each adaptive meme is tabulated. Comparing with table 9 a marked increase in robustness is observed with the multimeme reaching optimal values much

Table 8. *Different innovation rates and the relation between the number of times the multimeme algorithm reached optima relative to the number of runs executed. Also, in the third column, the mean first hitting time is computed. Memes are static helpers. Instance 15 in [8]*

Innovation Rate	#Optima / #Runs	Mean First Hitting Time
0.0	4/10	20.25
0.00001	4/10	20.5
0.0001	5/10	20.0
0.001	6/10	21.16
0.01	3/10	17.66
0.1	6/10	28.83
0.2	7/10	23.71
0.3	4/10	18.0
0.4	3/10	20.0
0.5	4/10	22.5
0.6	4/10	24.5
0.7	5/10	33.8
0.8	4/10	55.0
0.9	3/10	20.33
1.0	3/10	109.0

Table 9. *Memes are adaptive helpers in this table.*

Innovation Rate	#Optima / #Runs	Mean First Hitting Time
0.0	8/10	19.5
0.00001	7/10	19.14
0.0001	8/10	19.12
0.001	6/10	16.16
0.01	8/10	16.87
0.1	6/10	18.83
0.2	6/10	22.66
0.3	8/10	45.5
0.4	7/10	43.71
0.5	9/10	26.44
0.6	6/10	24.33
0.7	4/10	31.25
0.8	8/10	32.12
0.9	4/10	33.75
1.0	6/10	36.5

Table 10. *Relation between the number of times the single meme algorithm reached optima relative to the number of runs executed for different memes. Also, in the third column, the mean first hitting time is computed. The last row presents the associated values for the multimeme algorithm. Memes are static helpers. Instance 15 in [8]*

Static Memes	#Optima / #Runs	Mean First Hitting Time
GA (no memes)	0/10	-
Macro Mutation (r=4)	5/10	85.0
Macro Mutation (r=8)	2/10	100.0
Macro Mutation (r=16)	1/10	100.0
Reflect (r=4)	3/10	27.3
Reflect (r=8)	2/10	63.5
Reflect (r=16)	1/10	100.0
Stretch (r=4)	0/10	-
Stretch (r=8)	0/10	-
Stretch (r=16)	0/10	-
Pivot	5/10	67.4
MultiMeme(IR=0.2)	7/10	23.71

Table 11. *Memes are adaptive helpers in this table.*

Static Memes	#Optima / #Runs	Mean First Hitting Time
GA (no memes)	0/10	-
Macro Mutation (r=4)	2/10	27.5
Macro Mutation (r=8)	3/10	53.3
Macro Mutation (r=16)	2/10	43.0
Reflect (r=4)	3/10	20.6
Reflect (r=8)	1/10	79.0
Reflect (r=16)	1/10	45.0
Stretch (r=4)	0/10	-
Stretch (r=8)	0/10	-
Stretch (r=16)	0/10	-
Pivot	5/10	27.0
MultiMeme (IR=0.01)	8/10	16.87

more frequently than the single meme algorithm. Furthermore, the hitting times are also improved.

There are two main differences between Tables 10 (static memes) and 11 (adaptive memes). One of them is that for six of the memes the number of optima hits remains the same, while there are two improvements and two deteriorations. However, if we concentrate on the mean first hitting times the adaptive memes need roughly half the time needed by the static memes to achieve their respective optima.

Comparing Table 9 with the equivalent Table 8 for a multimeme static algorithm, we find the one employing adaptive memes is a better option than

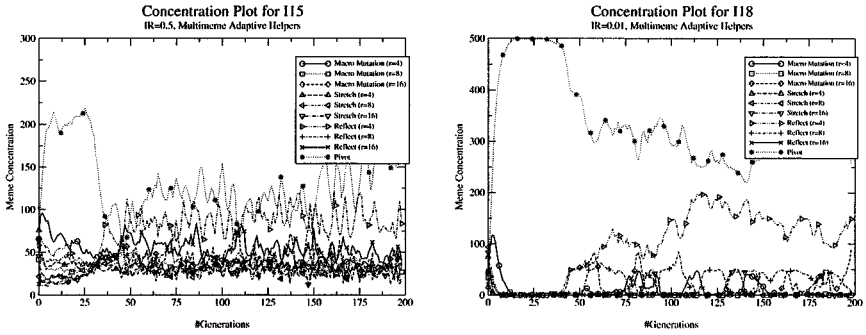
a multimeme algorithm with static memes. Moreover, both versions of the multimeme are, in turn, better than single meme algorithms, even ignoring the fact that it is difficult to know a priori which will be the best meme to use for a given instance. Similar results were found with other instances (in particular instance 18).

We perform an ANOVA analysis to find out whether the differences on mean hitting times were different with statistic significance. We found that the ANOVA identifies as statistically different, and indeed worse (longer average hitting times), the IR of $\{0.3, 0.8, 1.0\}$. While the others were ranked as well the associated confidence level was below 90%.

To understand which memes the multimeme algorithm selects during the search and at what times, Figure 6(a) shows the concentration of memes as a function of time. In this case concentration is plotted for an $IR = 0.5$, with an associated hitting ratio of $\frac{9}{10}$ and average hitting time of 26.44 generations.

In this case, the prevailing meme is the pivot move(e.g. a rigid rotation), followed very closely by the reflect with $r = 4$ during the whole run. We can see that when the concentration of the reflect meme peaks, the concentration of the pivot decreases. At the beginning of the run and for the considerably long period of 25 generations the macromutation with $r = 4$ is prominent. Important concentration peaks for reflect $r = 16$ are also found. Figure 6(b) shows the concentration graph for this instance and an innovation rate of 0.01

.



(a) (b)

Fig. 6. *In (a) meme concentration as a function of time for I15 and IR=0.5. Memes are adaptive. In (b) (Instance I18 in [8]):Adaptive Memes Concentration Graph for Instance I18 and an innovation rate of 0.01.*

Before generation 47, where the first important average fitness deterioration by the dominant adaptive helper (pivot meme) occurs, both reflect with $r = 4$ and macro mutation with $r = 16$ start to increase their presence in the population. Reflect helpers with $r = 4$ and $r = 8$ and the pivot meme are going to be the subsequent dominant memes.

5 Conclusion

In this paper we motivated the used of adaptive helpers and the inclusion of several local search neighborhoods in the set of available operators for a Memetic algorithm. Using two small instances of the TSP we showed how an adaptive local search phase is more beneficial to an MA than a static local search phase. We integrated the adaptive helpers with a Multimeme algorithm and we showed the resulting adaptive Multimeme algorithm to be more robust for both TSP and PSP Problem instances. In this paper we showed that the use and scheduling of several local search operators, realized by the various memes, can aid to perform a better exploration and exploitation of the search space. Moreover, the inclusion of adaptive helpers that can jump "backwards" in the optimization space further increases the search capabilities of the multimeme algorithm. That is, the multimeme algorithm with *adaptive* helpers produces more robust results than both the multimeme algorithm with *static* helpers and the memetic (single meme) algorithm with adaptive or static helpers.

An avenue of research not explored here but certainly worth pursuing is that of studying the appropriate scaling of the temperature parameter of the adaptive helpers according to the specific neighborhood that each one explores. That is, in general, we can expect that different move operators will need different temperatures to perform the adaptation and the cross barrier jumps. In the experiments presented here the temperature is the same for all the memes. It should be clear that different memes might be better suited to different scaling of the temperature as each one of them sees a different fitness landscape.

Experiments where the SIM schedules a mix of static and adaptive in the same run will be the object of future research where its ability to select the best meme from this enlarged set will be tested. Another extension of this work is the utilization of memes based on fuzzy sets concepts like those described in [12]. As suggested in [22] and [23], investigations where the memes are fully co-evolved alongside the problem's solutions are under way.

The two design principles advanced in this paper are:

- The use of several local searchers that explore the search space from complementary perspectivesm and,
- The reduction of the MA "greediness" by either not using elitist replacement strategies or by exploiting operators that can lead to worsening points from which progress can be achieved at a later stage of the search.

These two design issues (that are supported also by theoretical considerations[23]) should be taken into account for any new problem if no clear "winner" heuristic is known for that problem and robustness is paramount. In this case, Memetic algorithms, in particular Multimeme memetic algorithms, can provide both solution quality and robustness.

6 Acknowledgments

Natalio Krasnogor thanks J.E. Smith, W.E.Hart, and S. Gustafson for many insightful discussions.

References

1. E. Aarts, F. de Bont, E. Habers, and P. van Laarhoven. Parallel implementations of the statistical cooling algorithm. *Integration, the VLSI Journal*, 4:209–238, 1986.
2. O. Becker and M. Karplus. The topology of multidimensional potential energy surfaces: Theory and application to peptide structure and kinetics. *J. Chemical Physics*, 106(4):1495–1517, 97.
3. K. Boese. *Models For Iterative Global Optimization.* Ph.D. Thesis, UCLA Computer Science Department, 1996.
4. T. Boseniuk and W. Ebeling. Boltzmann, darwin and haeckel strategies in optimization problems. In H. Schwefel and R. Manner, editors, *Parallel Problem Solving From Nature*, pages 430–444. Springer-Verlag, 1991.
5. B. Carr, W. Hart, N. Krasnogor, E. Burke, J. Hirst, and J. Smith. Alignment of protein structures with a memetic evolutionary algorithm. In *GECCO-2002: Proceedings of the Genetic and Evolutionary Computation Conference.* Morgan Kaufman, 2002.
6. P. Coveney and R. Highfield. *Frontiers of Complexity, the search for order in a chaotic world.* faber and faber (ff), 1995.
7. A. Kolinski, M. Betancourt, D. Kihara, P. Rotkiewicz, and J. Skolnick. Generalized comparative modeling (genecomp): A combination of sequence comparison, threading, and lattice modeling for protein structure prediction and refinement. *PROTEINS: Structure, Function, and Genetics*, 44:133–149, 2001.
8. N. Krasnogor. Two dimensional triangular lattice instances for the hp model. In *http://dirac.chem.nott.ac.uk/ natk/Public/HP-PDB/2dtrihp.html.*
9. N. Krasnogor. Co-evolution of genes and memes in memetic algorithms. In A. Wu, editor, *Proceedings of the 1999 Genetic And Evolutionary Computation Conference Workshop Program*, 1999.
10. N. Krasnogor. http://www.cs.nott.ac.uk/ ~nxk/papers.html. In *Studies on the Theory and Design Space of Memetic Algorithms.* Ph.D. Thesis, University of the West of England, Bristol, United Kingdom., 2002.
11. N. Krasnogor, B. Blackburne, E. Burke, and J. Hirst. Multimeme algorithms for protein structure prediction. In *Proceedings of the Parallel Problem Solving from Nature VII. Lecture notes in computer science*, 2002.
12. N. Krasnogor and D. Pelta. Fuzzy memes in multimeme algorithms: a fuzzy-evolutionary hybrid. In J. Verdegay, editor, *Book chapter in "Fuzzy Sets based Heuristics for Optimization"*. Physica Verlag, 2002.
13. N. Krasnogor and J. Smith. Memetic algorithms: Syntactic model and taxonomy. submitted to The Journal of Heuristics. Available from the authors.
14. N. Krasnogor and J. Smith. A memetic algorithm with self-adaptive local search: Tsp as a case study. In *Proceedings of the 2000 Genetic and Evolutionary Computation Conference.* Morgan Kaufmann, 2000.

15. N. Krasnogor and J. Smith. Emergence of profitable search strategies based on a simple inheritance mechanism. In *Proceedings of the 2001 Genetic and Evolutionary Computation Conference*. Morgan Kaufmann, 2001.
16. N. Krasnogor. Self-Generating Metaheuristics in Bioinformatics: The Proteins Structure Comparison Case Journal of Genetic Programming and Evolvable Machines (to appear, May 2004), 5:2, 2004
17. S. Lee and K. Lee. Asynchronus communication of multiple markov chains in parallel simulated annealing. *Proceedings of International Conference on Parallel Processing*, 3:169–176, 1992.
18. S. Mahfoud and D. Goldberg. Parallel recombinative simulated annealing. *Parallel Computing*, 21:1–28, 1995.
19. P. Moscato. http://www.densis.fee.unicamp.br/~moscato/memetic_home.html.
20. H. Nakamura, T. Sasaki, and M. Sasai. Strange kinetics and complex energy landscapes in a lattice model of protein folding. *Chemical Physics Letters*, 347:247–254, 2001.
21. G. Reinelt. Tsplib (http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/tsplib95/tsplib.html). In *mirror site: gopher://softlib.rice.edu/11/softlib/tsplib.*
22. F. Rothlauf, D. Goldberg, and A. Heinz. Bad codings and the utility of well-designed genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*. Morgan-Kaufmann, 2000.
23. Y. Xia, E. Huang, M. Levitt, and R. Samudrala. Ab initio construction of protein tertiary structures using a hierarchical approach. *Journal of Molecular Biology*, 300:171–185, 2000.

NK-Fitness Landscapes and Memetic Algorithms with Greedy Operators and k-opt Local Search

Peter Merz

University of Kaiserslautern
Department of Computer Science
P.O. Box 3049, D-67653 Kaiserslautern, Germany
peter.merz@ieee.org

Summary. Memetic algorithms (MAs) with greedy initialization and recombination operators have been successfully applied to several combinatorial optimization problems, including the traveling salesman problem and the graph bipartitioning problem. In this contribution, a k-opt local search heuristic and a greedy heuristic for NK-landscapes are proposed for use in memetic algorithms. The latter is used for the initialization of the population and in a greedy recombination operator. Memetic algorithms with k-opt local search and three different variation operators, including the newly proposed greedy recombination operator, are compared on three types of NK-landscapes. In accordance with the landscape analysis, the MAs with recombination perform better than the MAs with mutation for landscapes with low epistasis. Moreover, the MAs are shown to be superior to previously proposed MAs using 1-opt local search.

1 Introduction

The NK-model of fitness landscapes has been introduced by Kauffman [1, 2] to study gene interaction in biological evolution. In the NK-model, the fitness is the average value of the fitness contributions of the loci in the genome. For each locus, the fitness contribution is a function of the gene value (allele) at the locus and the values of K other interacting genes. Although this is a very simplified model, it allows to produce families of fitness landscapes with interesting properties.

Besides its biological implications, the model is interesting for researchers in the field of evolutionary computation, since the NK-landscape model provides combinatorial optimization problems with tunable difficulty.

In this paper, effective memetic algorithms [3, 4, 5] for NK-landscapes are presented. New greedy and k-opt local search heuristics for NK-landscapes are proposed which can be easily embedded into memetic algorithms. The properties of NK-landscapes are discussed and a fitness distance correlation analysis

is performed for the newly introduced heuristic algorithms. It is shown that based on the results of the analysis, the performance algorithms can be predicted: For low epistasis – low values of K in the model – recombination based algorithms are able to exploit the structure of the search space effectively. With increasing epistasis, the landscapes become quickly unstructured, limiting the usefulness of recombination. For high epistasis, mutation based algorithms become favorable over recombination based evolutionary algorithms.

In computer experiments, the effectiveness of sophisticated MAs based on the proposed greedy and k-opt local search heuristics is demonstrated. These algorithms offer (near) optimum solutions in short time even for high dimensional landscapes.

The paper is organized as follows. In section 2, greedy and local search heuristics for the NK-model are introduced. The fitness landscape of three types of NK-models is discussed in section 3. In section 4, results from experiments with memetic algorithms using k-opt local search and three different variation mechanisms are presented. Section 5 concludes the paper and outlines areas of future research.

2 Heuristics for the *NK*-Model

Since NK-Landscapes have been studied mainly in the context of simulated biological evolution, little attention has been payed to the development of simple non-evolutionary heuristics. However, besides hill climbing/local search techniques, constructive heuristics such as greedy algorithms can be applied to problems of the NK-model.

In the following, a solution vector x is assumed to be a binary vector of length N, i.e. $x = (x_1, \ldots, x_N)$ with the fitness function

$$f(x) = \frac{1}{N} \sum_{i=1}^{N} f_i(x_i, x_{i_1}, \ldots, x_{i_K}), \tag{1}$$

where the fitness contribution f_i of locus i depends on the value of gene x_i and the values of K other genes $x_{i_1}, \ldots, x_{i_K}$. The function $f_i : \{0,1\}^{K+1} \to \mathbb{R}$ assigns a uniformly distributed random number between 0 and 1 to each of its 2^{K+1} inputs. Other random search landscapes have been proposed in [6, 7] which are highly tunable, but will not be investigated in this work.

The NK-model is similar to the unconstrained binary programming problem (BQP) [8]. In fact, the BQP can be regarded as a special case of NK-fitness landscapes with

$$f_(x) = \sum_{j=1}^{n} f_i(x) \quad \text{with} \quad f_i(x) = \sum_{j=1}^{n} q_{ij}\, x_i\, x_j, \tag{2}$$

where $Q = (q_{ij})$ is a $n \times n$ matrix. While for NK-landscapes $k(i) = K$ is constant for all i, in the BQP $k(i)$ is defined as the number of non-zero

entries in the i-th column of matrix Q. The mean $\overline{k}$ of the $k(i)$ is given by $\overline{k} = n \cdot \text{dens}(Q)$. Due to the strong resemblance of the two problems, heuristics developed for one problem can be applied after small modifications to the other. The heuristics described in the following are similar to the greedy and local search heuristics for the BQP in [9].

2.1 Greedy Algorithms

A point in a *NK*-landscape can be constructed in N steps by assigning in each step a gene value to a gene at a given locus. If the choice of a gene value follows a greedy rule, such an approach can be classified as a greedy heuristic for *NK*-landscapes.

The greedy heuristic proposed in this paper works as follows. A solution is built in N steps by choosing a gene which is still not assigned a value, and a gene value to assign to the gene. The choice is made by maximizing a gain function $g(i,v) : \{1,\ldots,N\} \times \{0,1\} \to \mathbb{R}$ with $g(i,v)$ denoting the gain attained by setting the value of the i-th gene to v. The gain function $g(i,v)$ is defined as the difference between the fitness of a partial solution y with gene i set to v and the fitness of a partial solution x with gene i unspecified: $g(i,v) = f^p(y) - f^p(x)$ with

$$y_j = \begin{cases} v & \text{, if } \quad i = j \\ x_j & \text{, otherwise.} \end{cases}$$

The fitness f^p of a partial solution is defined as the average fitness of all solutions matching the template defined by the partial solution: Assume the partial solution x is $x = (1,0,*,0,*,1)$ with $*$ denoting the *don't care* symbol (the gene has no value). Then, the fitness f^p of x is the average fitness of the four solutions $(1,0,\underline{0},0,\underline{0},1)$, $(1,0,\underline{0},0,\underline{1},1)$, $(1,0,\underline{1},0,\underline{0},1)$, and $(1,0,\underline{1},0,\underline{1},1)$.

Assuming the fitness contribution of site i denoted $f_i(x_i, x_{i_1}, \ldots, x_{i_K})$, depends on the site i itself and K neighbors $i_1,\ldots,i_K$, then the neighborhood $N_i = \{i, i_1, \ldots, i_K\}$ defines the set of genes/loci which contribute to the fitness at site i. The set of loci/genes which depend on the value of gene k is thus defined as $D_k = \{i \mid k \in N_i\}$. Hence, the gain function becomes

$$g(i,v) = f^p(y) - f^p(x) = \sum_{i \in D_k} f_i^p(x_i,\ldots,v,\ldots) - f_i^p(x_i,\ldots,x_k,\ldots). \quad (3)$$

Initially, the partial fitness contribution of locus i is the average over all 2^{K+1} possible values of f_i. Hence, the greedy heuristic based on partial fitness calculations requires more than $n \cdot 2^{K+1}$ additions and is therefore only practically useful for landscapes with small values of K. On the other hand, with increasing K, the solutions produced by the greedy heuristic approach the average fitness of the points in the landscape since for high epistasis the values of f_i^p differ significantly from the values of f_i in the final solution.

The greedy heuristic is randomized by (1) choosing a small fraction ($N/20$) of the genes randomly, and (2) by selecting randomly with a probability proportional to the gains from $\{\arg\max_i g(i,0), \arg\max_i g(i,1)\}$.

2.2 Local Search

The application of local search techniques to *NK*-landscapes is straightforward: Neighboring solutions can be reached by flipping one or more bits simultaneously in the genome. However, instead of calculating the fitness for each neighboring solution anew, it is more efficient to calculate the gain achieved by moving to the new solution. In this context the gain is referred to as the fitness difference between the new and the old solution.

The gain associated with the flipping of a single gene k in the genome x leading to a solution y with

$$y_i = \begin{cases} 1 - x_i \text{ ,if } & i = k \\ x_i & \text{,otherwise} \end{cases}$$

is the fitness difference of the new solution y and the old solution x:

$$g_k(x) = f(y) - f(x) = \sum_{i \in D_k} f_i(x_i, \ldots, 1 - x_k, \ldots) - f_i(x_i, \ldots, x_k, \ldots). \quad (4)$$

A local search for the *NK*-model can be implemented by maintaining a gain vector $g = (g_1 \ldots, g_N)$ instead of calculating all gains anew in each iteration. After flipping gene k, generally not all of the gains have to be updated. A gain g_i only changes if there is a $j \in D_i$ with $k \in N_j$ or in words the gain of flipping gene i changes if there is a fitness distribution function that depends on the value of gene k and i.

1-opt Local Search

A simple local search based on a 1-opt neighborhood can be realized straightforwardly. The neighborhood is searched by flipping a single bit in the current solution. The gain vector can now be used to find an improving flip in reduced computation time. However, after flipping the gene value, some elements of the gain vector have to be updated accordingly.

Variable k-opt Local Search

The basic scheme described above can be extended to derive more powerful local search algorithms. For example, a *2-opt* local search can be realized by flipping two genes to reach a solution in the neighborhood of the current solution. More generally, a k-opt local search can be realized by flipping k genes simultaneously. Since the neighborhood size of a k-opt local search grows

exponentially with k, mechanisms are required to perform a k-opt local search in reasonable time. This can be achieved be considering a small fraction of the k-opt neighborhood similarly to the heuristics by Lin and Kernighan for the TSP [10] and the GBP [11]. The k-opt local search for *NK*-landscapes proposed here is based on the ideas of Lin and Kernighan: in each iteration, a variable number of genes is flipped, depending on a gain criterion. To find the most profitable k-opt move, a sequence of up to n solutions is generated by stepwise flipping genes with the highest associated gain. Every gene is flipped no more than once to guarantee that all solutions in the sequence are different. The solution in the sequence with the highest gain is accepted as the new current solution. This solution may differ in 1 up to n genes depending on the position in the sequence. The pseudo code for the approach is provided in Figure 1. To reduce the running time of the algorithm, the value for the

```
procedure Local-Search-k-opt(x ∈ X): X;
  begin
    calculate gains g_i for all i in {1,...,N};
    repeat
      x_prev := x, G_max := 0, G := 0, steps = 0, C := {1,...,N};
      repeat
        find j with g_j = max_{i∈C} g_i;
        G := G + g_j;
        x_j := 1 − x_j;
        if G > G_max then
          G_max := G;
          x_best := x;
        endif
        update gains g_i for all i;
        C := C\{j};
        steps := steps + 1;
      until steps > maxsteps or C = ∅;
      if G_max > 0 then
        x := x_best;
      else
        x := x_prev;
      endif
    until G_max ≤ 0;
    return x;
  end;
```

Fig. 1. k-opt Local Search for *NK* Landscapes

maximum k can be bound to a value smaller than N. Furthermore, the inner repeat loop may be terminated if there was no new x_{best} for more than m solutions.

3 The Fitness Landscape of the *NK*-Model

The *NK*-model of Kauffman [2, 12] defines a family of fitness landscapes which can be tuned by two parameters: N and K. While N determines the dimension of the search space, K specifies the degree of epistatic interactions of the genes constituting a genome. Each point in the fitness landscape is represented by a bit string of length N and can be viewed as a vertex in the N-dimensional hypercube.

With this model, the "ruggedness" of a fitness landscape can be tuned by changing the value of K and thus the number of interacting genes per locus. Low values of K indicate low epistasis and high values of K represent high epistasis. The two extremes are considered in more detail in the following.

Properties of $K = 0$ Landscapes

The $K = 0$ landscapes have the following properties [2]:

- There is only one 1-opt local/global optimum
- The landscape is smooth; neighboring points (*1-opt* neighbors) in the search space are highly correlated. The fitness of 1-opt neighbors can differ by no more than $\frac{1}{N}$.
- The number of fitter neighbors decreases by one in each iteration of a 1-opt local search.
- The average number of iterations to reach the optimum is $\frac{N}{2}$ and thus in $O(N)$.

For the highest value of K, the properties of the fitness landscapes become quite different.

Properties of $K = N - 1$ Landscapes

If $K = N - 1$, the fitness contribution of a gene depends on the values of all other genes, which results in a highly uncorrelated, rugged fitness landscape. These landscapes have the following properties [2]:

- The expected number of 1-opt local optima is $\frac{2^N}{N+1}$
- The expected fraction of fitter 1-opt neighbors dwindles by $\frac{1}{2}$ after each iteration of a 1-opt local search
- The expected number of improvement steps to reach a 1-opt local optimum is in $O(\log N)$
- The expected number of solutions to examine for reaching a 1-opt local optimum is proportional to N
- The ratio of accepted to tried moves scales as $\log N/N$
- Starting from an arbitrary solution, only a small fraction of local optima ($\leq N^{\log_2(N-1)/2}$) can be reached by a 1-opt local search.

- Only from a small fraction of starting solutions ($2^{(\log_2 N)^2/2}$), the global optimum can be reached by a 1-opt local search.

Furthermore, Kauffman [2] has shown that for increasing N, the fitness values of the local optima decrease towards $\frac{1}{2}$. He calls this phenomenon *a complexity catastrophe.*

Random vs. Adjacent Neighbor Model

Besides the values for the parameters N and K, the choice of the neighbor model is important for *NK*-landscapes, too. Kauffman [2] distinguishes two variants, the *random neighbor model* and the *adjacent neighbor model.* In the former, the genes which contribute to the fitness at locus i are chosen at random. In other words, the neighbors i_1 through i_K are randomly selected among the N. In the latter, the i_1 through i_k are the nearest loci to the gene at locus i.

The landscape properties described above are independent of the neighbor model. However, Weinberger [13] has shown that the computational complexity of both models differs. He was able to show that the *NK* decision problem with adjacent neighbors is solvable in $O(2^K N)$ steps and is thus in $\mathcal{P}$ and that the *NK* decision problem with random neighbors is $\mathcal{NP}$-complete for $K \geq 3$.

3.1 Autocorrelation Analysis

To measure of the ruggedness of a fitness landscape, Weinberger [14] suggests the use of (auto)correlation functions. The *autocorrelation function* $\rho(d)$ [15, 14] reflects the fitness correlation of points at distance d in the search space. Weinberger [16] derived formulas for the autocorrelation function of *NK*-landscapes. He found that the autocorrelation function $\rho(d)$ depends on the neighbor model of the landscape. In the random neighbor model, the autocorrelation function becomes

$$\rho(d) = \left(1 - \frac{d}{N}\right)\left(1 - \frac{K}{N-1}\right)^d, \tag{5}$$

and for the adjacent neighbor model, ρ becomes

$$\rho(d) = 1 - \frac{d(K+1)}{N} + \frac{d}{N\binom{N-1}{d-1}} \sum_{l=1}^{K} (K+1-l)\binom{N-l-1}{d-2}, \tag{6}$$

with d denoting the hamming distance between bit vectors.

Alternatively, Weinberger suggested to perform random walks to investigate the correlation structure of a landscape. The *random walk correlation function* $r(s)$ [14, 17, 18] of a time series $\{f(x_t)\}$ defines the correlation of two points s steps away along a random walk through the fitness landscape.

The random walk correlation function for the *NK*-model has been calculated by Fontana *et al.* [19]:

$$r(s) \approx \left(1 - \frac{K+1}{N}\right)^s \tag{7}$$

for the adjacent and random neighbor model.

If the time series is *isotropic*, *Gaussian* and *Markovian* [14], then the corresponding landscape is called AR(1) landscape, and the random walk correlation function is of the form $r(s) = r(1)^s = e^{-s/\ell}$ with ℓ being the correlation length of the landscape. Hence, the correlation length ℓ [18] of the landscape is defined as

$$\ell = -\frac{1}{\ln(|r(1)|)} = -\frac{1}{\ln(|\rho(1)|)} \tag{8}$$

for $r(1), \rho(1) \neq 0$. The correlation length directly reflects the ruggedness of a landscape: the lower the value for ℓ, the more rugged the landscape. In the *NK*-model, the correlation length is for adjacent and random neighbors

$$\ell \approx \frac{N}{K+1}. \tag{9}$$

It is not surprising that the correlation length decreases with increasing K.

The formula show that the *NK*-model allows to produce landscapes with arbitrary ruggedness. The correlation length can be set to 1 by choosing $K = N - 1$ leading to a totally random landscape with uncorrelated neighboring points. Choosing the other extreme $K = 0$, the correlation length grows to its maximum value: N, resulting in a smooth, single peaked landscape.

3.2 Fitness Distance Correlation Analysis

The fitness distance correlation (FDC) coefficient is known to be an important measure in the context of fitness landscapes, proposed in [20] as a measure for problem difficulty for genetic algorithms. The FDC coefficient ϱ is defined as

$$\varrho(f, d_{\text{opt}}) = \frac{\text{Cov}(f, d_{\text{opt}})}{\sigma(f)\,\sigma(d_{\text{opt}})} \approx \frac{1}{\sigma(f)\sigma(d)} \frac{1}{m} \sum_{i=1}^{m} (f_i - \overline{f})(d_i - \overline{d}), \tag{10}$$

given a set of points $x_1, x_2, \ldots, x_m$ with $f_i = f(x_i)$ denoting the objective value, $d_i = d_{opt}(x_i)$ denoting the shortest distance to a global optimum solution, and $\sigma(f)$ and $\sigma(d)$ denoting the standard deviation of f and d, respectively.

In his studies of *NK*-landscapes, Kauffman [2] investigated the correlation of fitness and distance to the optimum of local optimum solutions with respect to 1-opt local search. In this work, the analysis is extended by investigating fitness distance correlation with respect to the greedy heuristic and k-opt local search. Experiments were conducted for three selected instances with N fixed

to 1024, K in $\{2, 4, 11\}$ and a random neighbor model. Since the optimum solutions for these instances are not known, the best solutions found with the MAs described below in long runs (14400 s on a Pentium II 300 MHz PC) are used instead. These solutions are likely to be the global optima or at least close to the global optima with respect to fitness and distance.

In the first experiment, the distribution of greedy solutions in the search space is investigated. The results of the analysis are summarized in Table 1. In the first column, the name of the instance is displayed, and in the second

Table 1. Results of the Fitness Distance Analysis of Greedy Solutions.

Instance	N	K	$\min d_{opt}$	$\overline{d}_{opt}$	$\overline{d}_{gr}$	N_{gr}	ϱ
C2-1024	1024	2	130	220.62 (0.22)	195.03	2500	-0.62
D4-1024	1024	4	264	372.29 (0.36)	377.38	2500	-0.24
B11-1024	1024	11	458	515.74 (0.50)	469.35	2500	-0.01

and third column the parameters N and K are given. In columns four through eight, the minimum distance of the greedy solutions to the expected global optimum ($\min d_{opt}$), the average distance of greedy solutions to the global optimum ($\overline{d}_{opt}$), the average distance between the greedy solutions ($\overline{d}_{gr}$), the number of distinct greedy solutions (N_{gr}) out of 2500, and the fitness distance correlation coefficient (ϱ) are provided, respectively. Additionally, the normalized average distance, i.e. the average distance of the local optima to the global optimum divided by the maximum distance in the search space N is shown in column five in parentheses.

For small K, the greedy solutions are close to each other and close to the best known solution. There is a correlation between fitness and distance to the best known solution as the value ρ indicates. About three quarters of the gene values are equal in all greedy solutions for $K = 2$ and thus the solutions are contained in a small fraction of the search space. With increasing K, average distance between the greedy solutions quickly converges to the average distance ($N/2$) of the solutions in the search space. Surprisingly, already at $K = 11$ there is no correlation between greedy solutions and they have random distribution in the search space as expected for large values of K.

In the second experiment, the correlation of fitness and distance to the best known solution of k-opt solutions was investigated. The results are shown in Table 2. Again, in the first column, the name of the instance is displayed, and in the second and third column the parameters N and K are given. In columns four through eight, the minimum distance of the locally optimal solutions to the expected global optimum ($\min d_{opt}$), the average distance of the local optima to the global optimum ($\overline{d}_{opt}$), the average distance between the local optima ($\overline{d}_{loc}$), the number of distinct local optima (N_{k-opt}) out of 2500, and the fitness distance correlation coefficient (ϱ) are provided, respectively. Additionally, the normalized average distance, i.e. the average distance of the

Table 2. Fitness Distance Correlation Analysis of k-opt Solutions.

Instance	N	K	$\min d_{opt}$	$\overline{d}_{opt}$	$\overline{d}_{loc}$	N_{k-opt}	ϱ
C2-1024	1024	2	191	301.47 (0.29)	346.16	2500	-0.65
D4-1024	1024	4	347	440.57 (0.43)	470.36	2500	-0.33
B11-1024	1024	11	459	511.88 (0.50)	511.72	2500	0.02

local optima to the global optimum divided by the maximum distance in the search space N is shown in column five in parentheses. Similarly as for the greedy heuristic, the average distance between the local optima and the average distance to the best known solution increases quickly with increasing K. At $K = 11$ there is no correlation between fitness and distance, and the distribution is similar to a uniform distribution of random points in the search space. There is slightly higher correlation in case of k-opt in comparison to 1-opt in case of the $K = 2, 4$ landscapes. However, greedy solutions have even a shorter minimum and average distance to the best known solution than k-opt solutions. In addition to Tables 1 and 2, fitness distance plots for the three instances are shown in Figure 2. On the left, the scatter plots for 2500 greedy solutions are provided, and on the right the scatter plots for 2500 k-opt solutions are displayed. For $K = 2$, the orientation of the points towards the origin is obvious. The cloud of points "moves" with increasing K quickly to the middle of the plane losing the orientation to the origin and thus to the optimum. These results correspond to the findings of Kauffman [2] for 1-opt local search. He further observed that for instances of the adjacent neighbor model the correlation of fitness and distances decreases not as rapidly as for the random neighbor model with increasing K.

From the perspective of performance prediction of MAs, the analysis provides some useful information. For small K (< 5) , recombination-based memetic algorithms are expected to have a good performance since with recombination the fitness distance correlation of the local optima can be exploited: With increasing fitness, the local optima are closer together, and their distance to the optimum becomes smaller. Furthermore, the locally optimal solutions are found in a small region of the search space in which the global optimum has a more or less central position. The greedy heuristic is very well suited for these instances with low epistasis and it is therefore promising to include the heuristic in the initialization phase of the population as well as in the recombination step. For larger K, the effectiveness of recombination decreases and eventually mutation based MAs are better suited.

3.3 Alternative Distance Measures

The fitness distance correlation analysis requires a feasible distance measure for the search space. In case of bit-strings, the hamming distance appears to

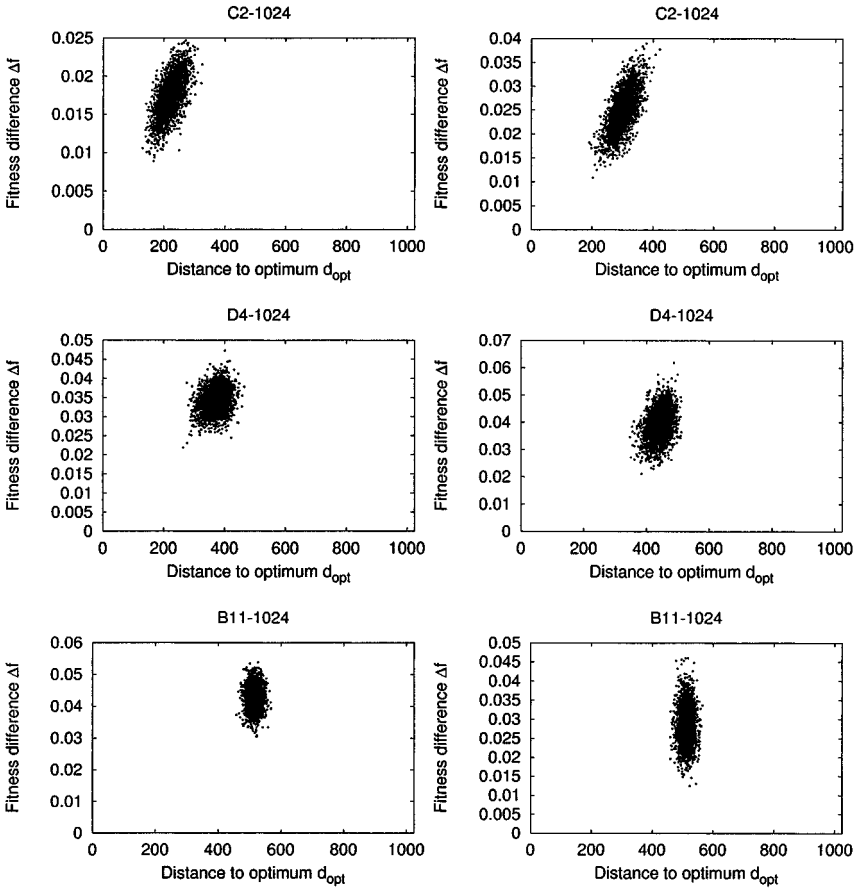


Fig. 2. Fitness-Distance Plots of Greedy Solutions (left) and k-opt Solutions (right)

be a natural choice. However, the hamming distance does not reflect exactly how a k-opt local search "sees" the landscape. Alternatively, an edit distance may be considered which counts the changes required for a k-opt local search to convert one solution to the other. However, a problem arises with such an approach, since the k-opt local search is not capable of converting all solutions into all other solutions. Besides the fact that only better solutions are produced by a k-opt, per definition not all better solutions are found by the local search. The hamming distance is a lower bound of the number of steps (flips) required for a k-opt local search to convert a solution to another assuming that it can. Essentially, a 1-opt local search and a k-opt local search are based on single flips, only the acceptance criterion is different in k-opt. Note, that the k-opt local search discussed in this paper considers only a

sequence of order one flips, not, for example, all pairs of order two flips (as would be in a true 2-opt local search). Finally, the FDC analysis may provide hints how the evolutionary part of a MA "sees" the landscape. Here, the hamming distance appears still to be a suitable choice, since properties like respectfulness and assortedness [21, 22] can be described with this distance measure.

4 Memetic Algorithms for *NK* Landscapes

Memetic algorithms have been applied with great success to several combinatorial optimization problems. In this paper, we focus on a class of memetic algorithms that uses a simple evolutionary framework with a single panmictic population instead of spatially structured populations [23], or tree-structured populations [24]. Furthermore, we concentrate on using a single local search strategy in contrast to the self-adaptation of the local search strategy [25]. The framework is thus rather simple and derived from other evolutionary algorithms, with the only difference that after initialization and after recombination or mutation, a local search procedure is applied to assure that all individuals in the population are local optima. This simple framework has been successfully used in studies for several combinatorial problems, including the graph bipartitioning problem [26], the quadratic assignment problem [27], the traveling salesman problem [28], and the binary quadratic programming problem [29].

The application of MAs to *NK*-landscapes is straightforward. Since problems of the *NK*-model are binary-coded, all GA variation operators such as k-point crossover and bit-flip mutation for bit strings can be used in a MA. As shown in [30], genetic algorithms do not scale well with problem size N. They perform much worse than memetic algorithms for a problem size $N \geq 512$. Therefore, we concentrate in the following on the hardest landscapes from the studies in [30] with $N = 1024$ and varying K.

4.1 Population Initialization and Local Search

The population can be initialized by randomly generating bit strings and by subsequently applying local search. For low values of K, the use of the randomized greedy heuristic described above can be used alternatively in combination with local search. Suitable local search algorithms are 1-opt local search and k-opt local search as described above.

4.2 Evolutionary Variation Operators

Due to the binary coding of the problem, all operators on binary strings can be applied in an evolutionary algorithm and therefore in a memetic algorithm, such as single point or two-point crossover, uniform crossover and bit flip mutation operators.

Recombination

A variant of uniform crossover (UX) that is used in the CHC algorithm of Eshelman [31] is an alternative to the crossover operators noted above. The operator creates (with high probability) offspring that have a maximum Hamming distance to the parents which is half of the distance between the parents themselves. The operator is called denoted *HUX* in the following.

Alternatively, the greedy construction scheme can be used in recombination to produce offspring. A *greedy recombination operator* denoted *GX* is therefore devised that works by first inheriting all the gene values that are common to the two parents to retain respectful recombination [22]. Then the remaining loci are set making greedy choices as in the greedy heuristic described above. This operator is especially effective for problems with low epistasis.

Mutation

Simple bit flip mutation is not useful in a memetic algorithm, since the flipping of a single bit will be reversed by a subsequently performed local search with a high probability. Hence more than one bit must be flipped simultaneously in the parent solution. If p bits are flipped by the mutation operator, the Hamming distance of the resulting offspring and the original parent solution becomes p. The value of p should be chosen to minimize the probability that the subsequent local search rediscovers the unmutated solution.

4.3 Selection and Restarts

Selection for reproduction is performed on a purely random basis without bias to fitter individuals, while selection for survival is achieved by choosing the best individuals from the pool of parents and children. Thus, replacement in our algorithm is similar to the selection in the $(\mu + \lambda)$-ES [32]. Additionally, duplicates will be replaced by other solutions, so that each phenotype exists only once in the new population.

In order to circumvent the problem of premature convergence, cataclysmic mutations [31] are performed when the population has converged. The mutation operator is applied to all but the best individual in the population, where p is determined by a third of the average Hamming distance between the individuals in the initial population. This value for p exhibited good performance in several experiments.

5 Performance Evaluation

We studied the performance of the memetic algorithms described above in several experiments. The results are discussed in the following starting with

an evaluation of the components, namely the greedy and local search heuristics. All experiments were performed on a Pentium II PC (300 MHz). The algorithms were implemented in C++.

5.1 Variable k-opt Local Search Variants

Running time and solution quality of the k-opt local search highly depend on the termination criterion of the inner loop, in other words, the maximum number of steps (search depth) considered in each iteration.

In order to investigate the influence of the search depth termination criterion we tested three variants of the local search procedure in Fig. 1. The full k-opt variant is exactly as shown in the figure, with *maxsteps* set to N. In the fast variant, the inner loop is terminated if there was no new x_{best} for more than $m = 40$ steps and the number of *maxsteps* was set to $N/2$. Finally, a simple tabu search variant was considered. In this variant the inner loop is terminated as soon as a better solution has been found. It is essentially a tabu search with a memory of N solutions and no aspiration criterion. The results of the comparison is displayed in Table 3. In the table, the percentage

Table 3. Comparison of k-opt Local Search Variants

Instance	Fast k-opt		Full k-opt		Tabu k-opt	
C2-1024	3.322%	1.0	3.263%	3.1	3.708%	20.4
D4-1024	4.918%	1.0	4.810%	2.7	5.614%	10.3
B11-1024	3.697%	1.0	3.571%	2.7	4.427%	2.5

deviation from the best known solution as well as the relative performance in respect to the fast variant are provided (larger values denote higher run times). As the figures suggest, the full variant is approximately 3 up to 5 times slower than the fast variant with only slightly better average objective values. Hence, the extra running time for the full variant appears not to be justified. The tabu search variant is much slower (up to 20 times) than the fast variant and also clearly inferior in average solution quality. Therefore, the fast variant is used in all remaining experiments.

An interesting issue is how the dynamics of a k-opt local search change if the landscapes become more rugged: The number of K and the search depth of the k-opt local search may be related. To investigate this issue the local search variants were compared in respect to the average number of steps per iteration and the number of iterations required to find a local optimum. The findings are summarized in Fig. 3 and Fig. 4. In the left plot of Fig. 3, the average number of flips performed in each iteration of the fast variant are displayed. As can be seen, the number of flips is initially very high and slightly less than $N/2$ for $K = 2$. Not surprisingly, the number is much lower for $K = 11$ due to the rapidly decreasing (auto-)correlation function of the landscape. As

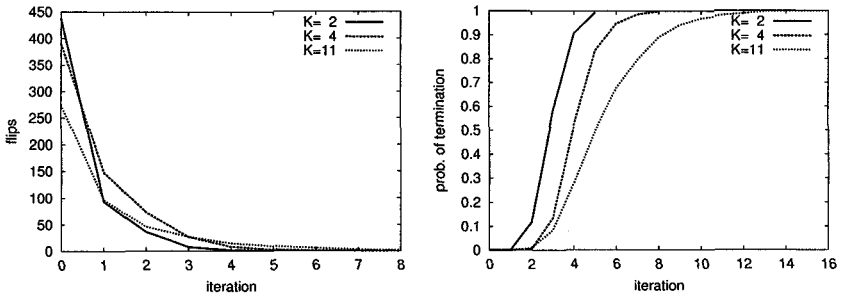


Fig. 3. k-opt Local Search Statistics for the Fast k-opt Variant

shown in the right of the figure, the number of iterations to reach a local optimum is very low (below 14 iterations), and increases with K. In the tabu

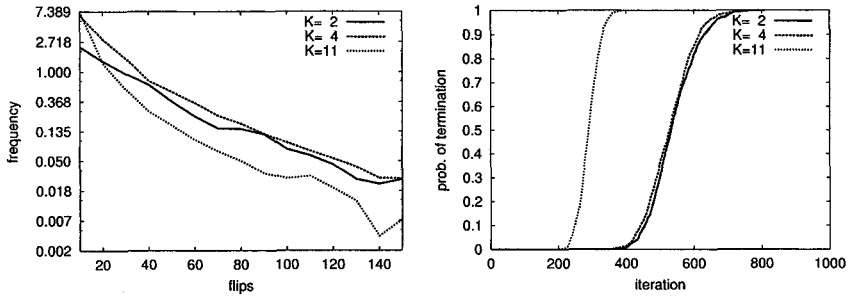


Fig. 4. k-opt Local Search Statistics for the Tabu k-opt Variant

search variant the expected number of iterations is much higher since flips are performed immediately, when an improving flip is found. The probability of termination is provided in the right plot of Fig. 4. Up to 800 iterations are required for $N = 1024$. The plot in the left hand side of the figure shows the frequency of k-flips depending on k on a logarithmic scale. Again, the frequencies of the $K = 11$ landscape are lower than those of the other two landscapes with $K = 2$ and $K = 4$, and the frequencies decrease exponentially. These results indicate that the optimum number of k (the search depth) in a k-opt local search should be dynamically chosen and not to be fixed in advance. In MAs where this parameter is adapted, it should be ensured that the parameter can be adjusted fast enough to meet the requirements at the current state of the search.

5.2 Greedy and Local Search

To investigate the relative performance of the greedy heuristic and the k-opt local search, experiments were conducted in which the two together with the

1-opt local search were applied to the three landscapes with $N = 1024$ used in the analysis above. The results are shown in Table 4. In the table, the average

Table 4. Performance of the Greedy Heuristic, 1-opt and k-opt Local Search.

	Greedy		1-opt LS		k-opt LS	
Instance	fitness	t/ms	fitness	t/ms	fitness	t/ms
C2-1024	0.7326 (2.33%)	76.9	0.7135 (4.88%)	19.7	0.7251 (3.32%)	52.3
D4-1024	0.7563 (4.34%)	173.2	0.7237 (8.47%)	28.4	0.7515 (4.94%)	114.8
B11-1024	0.7262 (5.56%)	22120	0.7094 (7.74%)	112.5	0.7403 (3.72%)	677.3

performance (fitness and average percentage excess in parentheses) and the average running time (t/ms) in milliseconds of a single run, is shown for the greedy heuristic and 1-opt and k-opt local search applied to randomly generated solutions. The values are averaged over 10000 runs except for the greedy heuristic and the problem instance B11-1024: Due to the long running time, 1000 runs were performed instead of 10000. The values given in parentheses denote the deviation from the best known solution in percent.

For $K = 2$ and $K = 4$, the greedy heuristic outperforms the local searches but requires more CPU time. For $K = 11$, the k-opt local search dominates over the two others. The CPU time required for a run of the greedy algorithm exceeds 22 seconds and is thus more than 32 times higher than for k-opt local search rendering the greedy heuristic impractical for such relative large K. For $K = 2$, the greedy heuristic is furthermore capable of producing comparable results to a GA in a single run and thus in 173 milliseconds, where the GA requires 1200 seconds [30]. Also for $K = 4$ and $K = 11$, the GAs in [30] are outperformed by the greedy heuristic and the k-opt local search in a single run, demonstrating even more drastically the inferior performance of traditional GAs on relatively large instances.

5.3 Memetic Algorithms with k-opt Local Search

To assess the performance of memetic algorithms with k-opt, additional experiments have been conducted. With the same time limit (1200 seconds) as chosen for the comparison of genetic algorithms with MAs in [30], the MAs with k-opt local search were applied to the three instances of size 1024. With a population size of 40, the production of 20 new offspring per generation, and restarts enabled as in [30], the MA were run with three different variation operators. The first MA uses the greedy heuristic in the initialization phase and the greedy recombination operator (GX). The second MA uses HUX as the recombination operator and the third MA uses the mutation operator (MUT) described above with $p = 3$. The results of the experiments are summarized in Table 5. For each algorithm, the average number of generations (gen) pro-

Table 5. Performance of k-opt Local Search MAs with three types of variation.

	C2-1024		D4-1024		B11-1024	
Op	gen	fitness, quality	gen	fitness, quality	gen	fitness, quality
GX	12505	0.750002, 0.01%	5750	0.787570, 0.39%		
HUX	11954	0.750009, 0.01%	5730	0.786874, 0.48%	216	0.753565, 1.99%
MUT	6402	0.744757, 0.71%	4306	0.772776, 2.26%	704	0.755747, 1.71%
HUX1	12615	0.748230, 0.25%	4540	0.783665, 0.89%	105	0.732874, 4.91%
Best		0.750065, 0.00%		0.790640, 0.00%		0.768882, 0.00%

duced is provided as well as the average fitness (fitness) of the final solution along with the percentage excess over the best known solution (quality). The results of the MA with 1-opt local search and HUX recombination (denoted HUX1) from [30] are included for easy comparison. Due to the long running times for the greedy heuristic on B11-1024, the MA with GX was not tested on this landscape.

For $K = 2$, the MA with greedy recombination and HUX recombination perform equally well. Both find the best known solution in one out of 20 runs and have the same worst result. For $K = 4$ and $K = 11$, the greedy recombination MA outperforms the others. The mutation based MA is as expected the worst out of the three for $K = 2$ and $K = 4$. For $K = 11$, the mutation based MA achieves a better average result than the MA with HUX. The same tendency appeared in the results of the MAs with 1-opt local search [30]: for the unstructured landscape with $K = 11$, recombination has no benefit compared to mutation. The recombination based MAs with k-opt local search perform clearly better than the algorithms with 1-opt local search in [30]. In particular new best solutions have been found for the three landscapes. Summarizing, the k-opt MAs have a higher potential and perform better if longer running times are chosen.

6 Conclusions

NK-landscapes have been introduced as a formal model of gene interaction in biological evolution, and since they are random, several statistical properties of the landscapes are known. To derive highly effective memetic algorithms for the *NK*-model, two new heuristics have been proposed, a greedy algorithm and a k-opt local search. The distribution of the solutions produced by these heuristics has been analyzed by performing a fitness distance correlation analysis on selected instances. The results allow to predict when greedy choices based on the greedy heuristic are favorable in a memetic framework and when not. Additionally, investigating the distribution of local optima in the landscapes allows to determine whether or not recombination is effective.

The greedy heuristic incorporated in the initialization phase as well as in the recombination operator of a MA with k-opt local search is shown to be highly effective for landscapes with low epistasis. The landscape analysis has shown that with increasing epistasis, the landscape becomes rapidly unstructured. Thus, for these instances, a k-opt local search MA with mutation instead of recombination has been shown to be favorable.

Moreover, the memetic algorithms with k-opt local search are shown to outperform previously proposed memetic algorithms with 1-opt local search: new best solutions have been found with the former for three landscapes.

There are several issues for future research. Firstly, the algorithms and landscape studies should be extended to cover other random search landscapes [6, 7]. Secondly, random walk correlation analysis may be applied on paths between local optima in the spirit of path relinking [33] to gain more insight in the effectiveness of recombination in memetic algorithm frameworks. Finally, the potentials of the algorithms described in the paper have to be investigated in other application domains of practical interest.

References

1. Kauffman, S.A.: Emergent Properties in Random Complex Automata. Physica D **10** (1984)
2. Kauffman, S.A.: The Origins of Order: Self-Organization and Selection in Evolution. Oxford University Press (1993)
3. Moscato, P.: On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms. Technical Report C3P Report 826, Caltech Concurrent Computation Program, California Institue of Technology (1989)
4. Moscato, P.: Memetic Algorithms: A Short Introduction. In Corne, D., Dorigo, M., Glover, F., eds.: New Ideas in Optimization. McGraw-Hill, London (1999) 219–234
5. Merz, P.: Memetic Algorithms for Combinatorial Optimization Problems: Fitness Landscapes and Effective Search Strategies. PhD thesis, Department of Electrical Engineering and Computer Science, University of Siegen, Germany (2000)
6. Smith, R.E., Smith, J.E.: An Examination of Tuneable, Random Search Landscapes. In Banzhaf, W., Reeves, C., eds.: Foundations of Genetic Algorithms 5. Morgan Kaufmann, San Francisco, CA (1999) 165–181
7. Smith, R.E., Smith, J.E.: New Methods for Tunable, Random Landscapes. In Martin, W.N., Spears, W.M., eds.: Foundations of Genetic Algorithms 6. Morgan Kaufmann, San Francisco (2001) 47–67
8. Laughunn, D.J.: Quadratic Binary Programming. Operations Research **14** (1970) 454–461
9. Merz, P., Freisleben, B.: Greedy and Local Search Heuristics for Unconstrained Binary Quadratic Programming. Journal of Heuristics **8** (2002) 197–213
10. Lin, S., Kernighan, B.: An Effective Heuristic Algorithm for the Traveling Salesman Problem. Operations Research **21** (1973) 498–516

11. Kernighan, B., Lin, S.: An Efficient Heuristic Procedure for Partitioning Graphs. Bell Systems Journal **49** (1972) 291–307
12. Kauffman, S.A., Levin, S.: Towards a General Theory of Adaptive Walks on Rugged Landscapes. Journal of Theoretical Biology **128** (1987) 11–45
13. Weinberger, E.D.: NP Completeness of Kauffman's N-k Model, A Tuneable Rugged Fitness Landscape. Technical Report 96-02-003, Santa Fe Institute, Santa Fe, New Mexico (1996)
14. Weinberger, E.D.: Correlated and Uncorrelated Fitness Landscapes and How to Tell the Difference. Biological Cybernetics **63** (1990) 325–336
15. Stadler, P.F.: Correlation in Landscapes of Combinatorial Optimization Problems. Europhysics Letters **20** (1992) 479–482
16. Weinberger, E.D.: Local Properties of Kauffman's N-k model: A Tunably Rugged Energy Landscape. Physical Review A **44** (1991) 6399–6413
17. Stadler, P.F.: Towards a Theory of Landscapes. In Lopéz-Peña, R., Capovilla, R., García-Pelayo, R., Waelbroeck, H., Zertuche, F., eds.: Complex Systems and Binary Networks. Volume 461 of Lecture Notes in Physics., Berlin, New York, Springer Verlag (1995) 77–163
18. Stadler, P.F.: Landscapes and their Correlation Functions. Joural of Mathematical Chemistry **20** (1996) 1–45
19. Fontana, W., Stadler, P.F., Bornberg-Bauer, E.G., Griesmacher, T., Hofacker, I.L., Tacker, M., Tarazona, P., Weinberger, E.D., Schuster, P.: RNA Folding Landscapes and Combinatory Landscapes. Physcal Review E **47** (1993) 2083–2099
20. Jones, T., Forrest, S.: Fitness Distance Correlation as a Measure of Problem Difficulty for Genetic Algorithms. In Eshelman, L.J., ed.: Proceedings of the 6th International Conference on Genetic Algorithms, Morgan Kaufmann (1995) 184–192
21. Radcliffe, N., Surry, P.: Formal Memetic Algorithms. In Fogarty, T., ed.: Evolutionary Computing: AISB Workshop. Volume 865 of Lecture Notes in Computer Science., Springer-Verlag, Berlin (1994) 1–16
22. Radcliffe, N., Surry, P.: Fitness Variance of Formae and Performance Prediction. In Whitley, L., Vose, M., eds.: Proceedings of the Third Workshop on Foundations of Genetic Algorithms, San Francisco, Morgan Kaufmann (1994) 51–72
23. Gorges-Schleuter, M.: ASPARAGOS: An Asynchronous Parallel Genetic Optimization Strategy. In Schaffer, J.D., ed.: Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann (1989) 422–427
24. Moscato, P., Tinetti, F.: Blending Heuristics with a Population-based Approach: A Memetic Algorithm for the Traveling Salesman Problem. Technical Report CeTAD, CeTAD, Universitad Nacional de La Plata (1994)
25. Krasnogor, N., Smith, J.: Emergence of profitable search strategies based on a simple inheritance mechanism. In Lee Spector *et al.*, ed.: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001), San Francisco, California, USA, Morgan Kaufmann (2001) 432–439
26. Merz, P., Freisleben, B.: Fitness Landscapes, Memetic Algorithms and Greedy Operators for Graph Bi-Partitioning. Evolutionary Computation **8** (2000) 61–91
27. Merz, P., Freisleben, B.: Fitness Landscape Analysis and Memetic Algorithms for the Quadratic Assignment Problem. IEEE Transactions on Evolutionary Computation **4** (2000) 337–352

28. Merz, P., Freisleben, B.: Memetic Algorithms for the Traveling Salesman Problem. Complex Systems **13** (2001) 297–345
29. Merz, P., Katayama, K.: Memetic Algorithms for the Unconstrained Binary Quadratic Programming Problem. Bio Systems (2002) To appear.
30. Merz, P., Freisleben, B.: On the Effectiveness of Evolutionary Search in High-Dimensional NK-Landscapes. In Fogel, D., ed.: Proceedings of the 1998 IEEE International Conference on Evolutionary Computation, Piscataway, NJ, IEEE Press (1998) 741–745
31. Eshelman, L.: The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination. In Rawlings, G.J.E., ed.: Foundations of Genetic Algorithms. Morgan Kaufmann (1991) 265–283
32. Schwefel, H.P.: Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie. Volume 26 of Interdisciplinary Systems Research. Birkhäuser Verlag, Basel (1977)
33. Glover, F.: Scatter Search and Path Relinking. In Corne, D., Dorigo, M., Glover, F., eds.: New Ideas in Optimization. McGraw-Hill, London (1999) 297–316

Self-Assembling of Local Searchers in Memetic Algorithms

Natalio Krasnogor and Steven Gustafson

Automatic Scheduling, Optimisation and Planning Group
School of Computer Science and IT
University of Nottingham, U.K.
http:\\www.cs.nott.ac.uk/~{nxk,smg}
{Natalio.Krasnogor,Steven.Gustafson}@nottingham.ac.uk

Summary. In this chapter we concentrate on one particular class of Global-Local Search Hybrids, Memetic Algorithms (MAs), and we describe the implementation of "self-assembling" mechanisms to produce the local searches the MA uses. To understand the context in which self-assembling is applied we discuss some important aspects of Memetic theory and how these concepts could be harnessed to implement more competitive MAs. Our implementation is tested in two problems, Maximum Contact Map Overlap Problem (MAX-CMO) and the NK-Landscape Problems.

Three lessons can be drawn from this paper:

- Memetic theory provides a rich set of metaphors and insights that can be harnessed within optimisation algorithms as to provide better search methods.
- The optimization of solutions can be done **simultaneously** with the self-assembling of local search strategies which can then be exploited by the Memetic Algorithm (or other metaheuristic)
- Local search strategies that are evolved to **supply building blocks** can greatly improve the quality of the search obtained by the Memetic Algorithm and do not seem to suffer from premature convergence (an ubiquitous problem for global-local hybrids).

1 Introduction

A vast number of very successful applications of Memetic algorithms (MAs) have been reported in the literature in the last years for a wide range of problem domains. The majority of the papers dealing with MAs are the result of the combination of highly specialized **pre-existing** local searchers and usually purpose-specific genetic operators. Moreover, those algorithms require a considerable effort devoted to the tuning of the local search and evolutionary parts of the algorithm.

In [23] and [25] we propose the so called "Self-Generating Metaheuristics". Self-Generating Metaheuristics can create on-the-fly the type of operators needed to successfully perform certain task. The self-generation concept

can be applied to any existing metaheuristic like simulated annealing, tabu search, etc. In the case of Memetic Algorithms, self-Generation implies that the MAs are able to **self-assemble** their own local searchers and to co-evolve the behaviors it needs to successfully solve a given problem. In Self-Generating Memetic Algorithms two evolutionary processes occur. On one hand evolution takes place at the chromosome level as in any other Evolutionary Algorithm; chromosomes and genes represent solutions and features of the problem one is trying to solve. On the other hand, evolution also happens at the memetic level. That is, the behaviors and strategies that individuals (also called agents) use to alter the survival value of their chromosomes are self-assembled from a set of components by means of, for example, an evolutionary process. As the self-assembeld memes (i.e. local search strategies) are propagated, mutated and are selected in a Darwinian sense, the Self-Generating MAs we propose are closer to Dawkins concept of memes than the previous works on memetic algorithms (e.g. [14],[33],[34],[4]). Additionally, they seem to be more robust and scalable than their single local searchers counterpart.

In this chapter we will review some important ideas arising from Memetic theory and we will describe the implementation we have choosen for the proposed algorithms. Results on the use of the Self-Assembling of local searchers for MAs are reported and future lines of research discussed.

2 The Memetic Metaphor

Memetic algorithms are not the first kind of algorithms to draw inspiration from natural phenomena. In this case the inspiration came from memetic theory. However, unlike Simulate annealing, Ant Colony optimization, GAs, etc., scholars working on MAs, as will be argued later, departed considerably from the metaphor and ignored its main features.

The common use of the term "memetic algorithm" refers to an evolutionary algorithm that employs as a distinctive part of its main evolutionary cycle (mutation, crossover and selection), a local search stage.

The name "memetic algorithm" is a very contested label that stirs critics and controversies among researchers and practitioners who usually adopt names such as lamarckian GAs, genetic local search, hybrid GAs, etc. Although very justificable in the large mayority of cases, these names obscure the fact that there is a large body of literature on memetic theory that is being neglected. We would like to argue in this section that if we were to put back the "memetic" into memetic algorithms then progress could be made with a new breed of algorithms that are more atune to the name "memetic algorithms".

Memetic theory started as such with the definition given by R. Dawkins of a meme in [11][1]:

[1] The definition was later refined in [12]

> I think that a new kind of replicator has recently emerged on this very planet. It is staring us in the face. It is still in its infancy, still drifting clumsily about in its primeval soup, but already it is achieving evolutionary change at a rate that leaves the old gene panting far behind. The new soup is the soup of human culture. We need a name for the new replicator, a noun that conveys the idea of a unit of cultural transmission, or a unit of imitation. "Mimeme" comes from a suitable Greek root, but I want a monosyllable that sounds a bit like "gene". I hope my classicist friends will forgive me if I abbreviate mimeme to meme.(2) If it is any consolation, it could alternatively be thought of as being related to "memory", or to the French word "même". It should be pronounced to rhyme with "cream". Examples of memes are tunes, ideas, catch-phrases, clothes fashions, ways of making pots or of building arches. Just as genes propagate themselves in the gene pool by leaping from body to body via sperms or eggs, so memes propagate themselves in the meme pool by leaping from brain to brain via a process which, in the broad sense, can be called imitation.

Many other researchers and philosophers "flirted" with the idea that cultural phenomena can somehow be explained in evolutionary terms even before Dawkins' introduction of a meme. Other symbols were introduced to refer to the elementary unit of cultural change and/or transmission (e.g., *m-culture* and *i-culture* [7], *culture-type* [42], etc.). See [13] for a comprehensive analysis. The merit of Dawkins contribution can be attributed to his insight into correctly assigning a new signifier, i.e., a label or symbol, to the thing being signified, i.e., the unit of cultural transmission. The term meme was a new word hence it was not loaded with preconceptions and misconceptions. From the computer sciences perspective it was appealing because it defined that concept as a discrete structure which can be easily harnessed in a computer program.

The fundamental innovation of memetic theory is the recognition that a dual system of inheritance, by means of the existence of two distinct replicators, mould human culture. Moreover, these two replicators interact and co-evolve shaping each other's environment. As a consequence evolutionary changes at the gene level are expected to influence the second replicator, the memes. Symmetrically, evolutionary changes in the meme pool can have consequences for the genes.

2.1 Memetic Theory in Evolutionary Computation

In any of the major evolutionary computation paradigms, e.g., GAs, Evolution Programs, Evolutionary Strategies, GPs, etc, the computation cycle shown in graph 1 takes place.

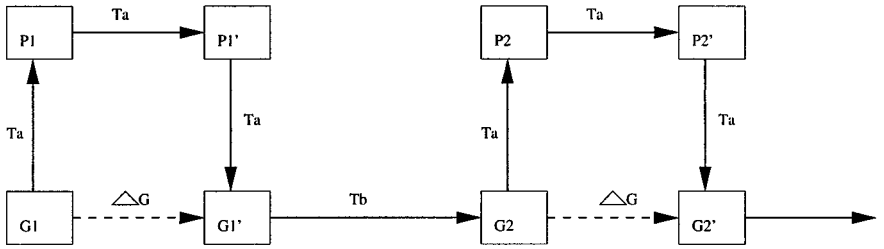


Fig. 1. *Evolutionary genetic cycle.*

In graph [2] 1 a hypothetical population of individuals is represented at two different points in time, generation 1 (G1) and at a later generation (G2). In the lower line, G_i for $i = 1, 2$ represents the distribution of genotypes in the population. In the upper line, P_i represents the distribution of phenotypes at the given time. Transformations T_A account for epigenetic phenomena, e.g., interactions with the environment, in-migration and out-migrations, individual development, etc., all of them affecting the distribution of phenotypes and producing a change in the distribution of genotypes during this generation. On the other hand transformations T_B account for the Mendelian principles that govern genetic inheritance and transforms a distribution of genotypes G_1' into another one G_2. Evolutionary computation endeavors concentrate on the study and assessment of many different ways the cycle depicted in 1 can be implemented. This evolutionary cycle implicitly assumes the existence of only one replicator: genes.

On the other hand what memetic algorithmicists should somehow investigate, if they were more faithful to the natural phenomena that inspired the methodology, is the implementation of a more general and complex dual evolutionary cycle where two replicators co-exist. This is shown in 2[3].

In the context of memetic algorithms, memes represent instructions to self-improve. That is, memes specify sets of rules, programs, heuristics, strategies, behaviors, etc, individuals can use in order to improve their own fitnesses under certain metric.

As we mentioned earlier, the fundamental difference between the later graph and the former resides in the fact that graph 2 reflects a coevolutionary system where two replicators of a different nature interact. Moreover the interactions between genes and memes are indirect and mediated by the common carrier of both: individuals. A truly memetic system should not be confused with other coevolutionary approaches where different "species", subpopulations or just different individuals interact by ways of a combination of cooperation, competition, parasitism, symbiosis, etc. In coevolutionary approaches like those described by [19],[36],[37],[38],[39],[40] and others, only

[2] This graph is adapted from [13] page 114.
[3] This graph is adapted from [13] page 186.

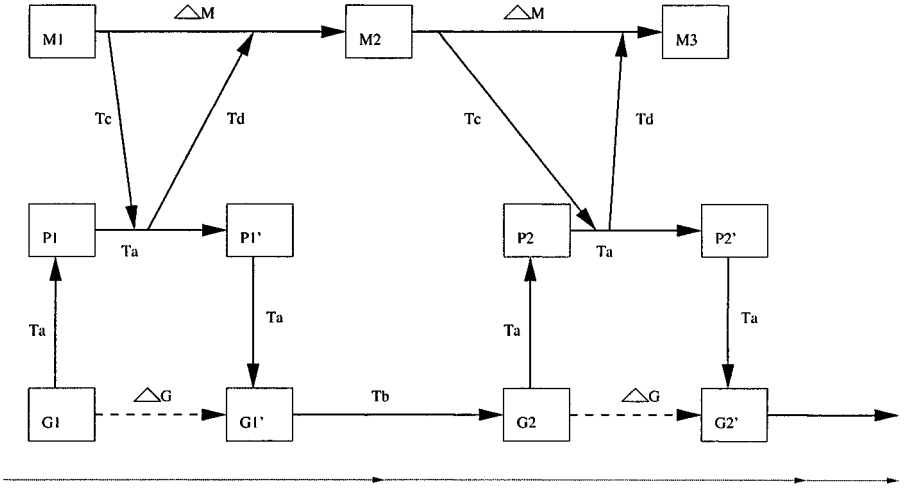


Fig. 2. *Coevolutionary memetic-genetic cycle.*

Mendelian transformations are allowed and sometimes in-migration and out-migration operators are also included. In a memetic system, memes can potentially change and evolve using rules and time scales other than the traditional genetic ones. In the words of Feldman and Cavalli-Sforza[6] memetic evolution is driven by:

> ...the balance of several evolutionary forces: (1) *mutation*, which is both purposive (innovation) and random (copy error); (2) *transmission*, which is not as inert as in biology [i.e., conveyance may also be horizontal and oblique]; (3) *cultural drift* (sampling fluctuations); (4) *cultural selection)* (decisions by individuals); and (5) *natural selection* (the consequences at the level of Darwinian fitness) ...

In graph 2 we have the same set of transformations as before between genes and phenotypes, but also meme-phenotypes and memes-memes relations are shown. There are mainly two transformations for memes that are depicted, T_C and T_D. Transformations T_C represents the various ways in which "cultural" instructions can re-shape phenotypes distributions, e.g., individuals learn, adopt or imitate certain memes or modify other memes. T_D, on the other hand, reflects the changes in memetic distribution that can be expected from changes in phenotypic distributions, e.g., those attributed to teaching, preaching, etc.

Memetic Algorithms as they were used so far failed completely, or almost completely, to implement this dual inheritance system to any degree, except for the works initiated with [22],[23] and continued with [25],[45],[24],[26]. Consequently, it is not surprising that researchers hesitate to call a GA (or other evolutionary approach) that uses local search a memetic algorithm.

2.2 Memes Self-Assembling

In [23],[25],[45],[24] it was proposed and demonstrated that the concept of Self-Generating Memetic algorithms can be implemented and, at least for the domains considered in those papers, beneficial. In the context of SGMAs, memes specify sets of rules, programs, heuristics, strategies, behaviors, or move operators the individuals in the population can use in order to improve their own fitnesses (under a given metric). Moreover the interactions between genes and memes are indirect and mediated by the common carrier of both: individuals (sometimes also called agents).

Gabora[15] mentions three phenomena that are unique to cultural (i.e. memetic) evolution. Those phenomena are *Knowledge-based, imitation* and *mental simulation.*

It is these three phenomena that our Self-Generating Memetic Algorithm implements and use to self-assemble its own local search strategies. The representation of the low level operators (in this chapter the local searchers) includes features such as the acceptance strategy (e.g. next ascent, steepest ascent, random walk, etc), the maximum number of neighborhood members to be sampled, the number of iterations for which the heuristic should be run, a decision function that will tell the heuristic whether it is beneficial for a particular solution or a particular region of a solution and, more importantly, the move operator itself in which the low level heuristic will be based[23].

Previous technologies for Evolutionary Algorithms, GRASP, Simulated Annealing, Tabu Search, etc have concentrated so far in, for example, self-adapting the probabilities with which different move operators[46] are used during the search for a problem's solution, the size of the tabu lists[49] or the size of populations[48], the adaptation of the aspiration criteria [1], the crossover points[43], mutations frequencies and intensities[47], the weights in the choice functions of Hyperheuristics [9], the appropriate local searcher that must be used by a Memetic Algorithms [28], the intensity of search[30], the degree of exploitation and exploration of local search [27], etc. However, only recentely some exploration on the on-line self-assembling of local searchers has been developed.

The role played by local search in both Memetic and Multimeme algorithms has traditionally been associated to that of a "fine tuner". The evolutionary aspect of the algorithm is expected to provide for a global exploration of space while the local searchers are assumed to exploit current solutions and to fine tune the search in the vicinity of those solutions (i.e. exploitation)

We will show next that local search strategies can be self-assembled in the realm of NK-Landscape problems and a graph theory combinatorial problem. Equally important, we will suggest a new role for local search in evolutionary computation in general and memetic algorithms in particular: *the local searcher not as a fine-tuner but rather as a supplier of building-blocks*[4]. Ini-

[4] For an overview of selectorecombinative evolutionary algorithms from a building-blocks perspective please refer to [17]

tial explorations of the many concepts described here appeared before in [25], [26] and [24] and we invite the reader to also consult those papers for further details.

3 The NK-Landscapes Experiments

The NK model of rugged fitness landscapes are particularly useful to understand the dynamics of evolutionary search[20] as they can be tunned to represent low or high epistasis regimes. The amount of epistasis is related to the level of interdependency of genes within a genome. That is, the fitness contribution of a particular gene's allele depends not only on the identity of that allele by also on which are the specific alleles in the remaining genes. To model this situation an NK-landscape instance consists of two integer n and k representing the total number of genes n and the number of other genes a gene i is epistatically realted to. The values k can take are $0 \leq k \leq n-1$. Besides n and k, a $n \times 2^{k+1}$ matrix E with elements sampled randomly from the (usually) uniform distribution $U(0,1)$ is also required to completely define an instans. A solution to an NK-landscape problem instance is represented as a binary string S with length n. The fitness of S is given by $fitness(S) = \frac{1}{n} * \sum_{i=1}^{i=n} f_i(S_i, S_{i_1}, \ldots, S_{i_k})$ where $f_i(\cdot)$ is an entry in E, S_i the value of string S at position i and S_{i_j} is the value of string S at the $j-th$ neighbour of bit i. The neighbours, not necessarily adjacent, j of bit i are part of the input as well.

In figure 3 we show a $(10,3)$ NK-landscape. The genome is formed by 10 genes ($N = 10$) and each of the genes is epistatically linked to 3 other genes ($K = 3$). In the example this is depicted by the curved arrows going out from gene i towards adyacent genes.

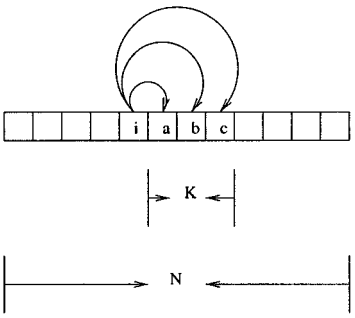


Fig. 3. *An example of NK-Landscapes*

Low values for k represent low epistatic problems while large k value make up highly epistatic landscapes. The extreme case of an uncorrelated random fitness landscape is when $k = n-1$. The optimization version of this problem

can be solved in polynomial time by dynamic programming if the neighborhood structure used is that of *adjacent neighbours*. The problem becomes NP-Hard if the structure used is that of *random neighbours*[50].

NK-Landscapes have been the subject of intensive and varied studies. Kaufmann et al. [31] explore a phase change in search when a parameter τ of a local search algorithm reaches a certain critical value on some NK-Landscape problems .

In their paper the authors show experimentally that the quality of the search follows an $s-$shape curve when plotted against τ making evident a change in phase. M. Oates et al. [35] showed performance profiles for evolutionary search based algorithms where phase changes were also present. Krasnogor and Smith [28] and Krasnogor [23] showed the existence of the "solvability" phase transition for GAs (instead than LS) and demonstrated that a self-adapting MA can learn the adequate set of parameters to use. Merz [32] devotes at least one whole chapter of his Ph.D. dissertation to the development of efficient Memetic Algorithms for this problem (we will return to his MAs later on). With a different target as the object of research O.Sharpe in [44] performs some analysis on the parameter space of evolutionary search strategies for NK landscapes.

The NK-Landscapes represent a rich problem and they are an ideal test case for our purposes. We will describe the behavior of our Self-Generating Memetic Algorithms in 4 different regimes: *low epistasis and poly-time solvable*, *high epistasis and poly-time solvable*, *low epistasis and NP-hard* and *high epistasis and NP-hard.*

In [23] and in previous sections we argued briefly about the need to creatively adapt every aspect of the local searchers, that is, the acceptance strategy, the maximum number of neighborhood members to be sampled, the number of iterations for which the meme should be run, a decision function that will tell the meme whether it is worth or not to be applied on a particular individual and, more importantly, the move operator itself. In this part of the chapter we will focus only on the self-generation of the move operator itself as a proof of concept[5].

The MA will be composed of two simultaneous processes. Individuals in the MA population will be composed of genetic and memetic material. The genetic material will represent a solution to NK-Landscapes problems (i.e. a bit string) while the memetic part will represent "mental constructs" to optimize the NK-Landscape string. As such we will be evolving individuals whose goal is to self-optimize by genetic evolution (first process) and memetic evolution (second process) as sugested by figure 2.

3.1 The Self-Generating Memetic Algorithm

The pseudocode in Figure 2 depicts the algorithm used to solve the NK-Landscape Problem.

[5] The other aspects are actively being investigated.

```
Memetic_Algorithm():
Begin
  t = 0;
  /* Initialize the evolutionary clock(generations) to 0 */
  Randomly generate an initial population P(t);
  /* The individuals in the population */
  /* are composed by genes & memes */
  /* both randomly initialized */
  Repeat Until ( Termination Criterion Fulfilled ) Do
    Variate individuals in M(t);
    /* The variation of an individual includes */
    /* both genetic and memetic variation */
    Improve_by_local_search( M(t));
    /* The local search is performed accordingly */
    /* to the individual's meme */
    Compute the fitness f(p) ∀p ∈ M(t) ;
    Generate P(t+1) by selecting from P(t) and M(t);
    t = t+1;
  endDo
  Return best p ∈ P(t-1);
End.
```

Fig. 4. The memetic algorithm employed.

The initial population in P is created at random. Each individual is composed of genetic material in the form of a bit string (B). The bit string represent the solution to the NK landscape instance being solved. The memetic material is of the form $* \rightarrow S$ where the $*$ symbol matches any bit in the solution string and S is another bit string. The only variation mechanism is bitwise mutation (applied with probability 0.05) to the chromosomes. The replacement strategy is a $(20, 50)$. There is no genetic crossover but the SIM mechanism, as described in [28], is used to transfer memes between individuals. Memetic mutation occurs with an innovation rate[23] of 0.2. A meme can be mutated (with equal probability) in three ways: either a random bit is inserted in a random position, a bit is deleted from a random position, or a bit is flipped at a random position. The length of memes cannot decrease below 0 nor increase beyond $3 * k$ for an $(n, k)-$problem.

The Local Search Procedure: Memes description for NK-Landscapes

A meme is represented as a rule of the form $* \rightarrow S$. During the local search stage this meme is interpreted as follows:

Every bit in the chromosome B has the opportunity to be improved by steepest hill-climbing. In general NK-Landscapes are epistatic problems so flipping only one bit at a time cannot produce reasonable improvements except of course in problems with very low k. To accommodate for this fact, for each bit, one wants to optimize the value of that bit and that of $|S|$ other bits. A sample of size n is taken from all the $(|S|+1)!$ possible binary strings. Based on the content of S, these sample strings serve as bits template with which the original chromosome B will be modified. If $|S| = 0$ then only B_i (the i^{th} bit of B) will be subjected to hill-climbing. On the other hand, if $|S| > 0$ then the local searchers scans the bits of S one after the other. If the first bit of S is a 0, then the bit $B_{(i+1)}$ will be set accordingly to what one of the n samples template mandates. On the other hand, if B_i is a 1 then bit $B_{(i+r)\%n}$ will be set as what one of the n samples template mandates. Here r is a random number between 0 and $n-1$. By distinguishing in S between ones and zeroes memes can reflect the adjacent neighbour or the random neighbour version of the NK-landscapes. The larger the size of S the more bits will be affected by the local search process. As an example consider the case where the rule is $* \rightarrow 0000$. This rules implies $S = 0000$. In this case, for every bit i in B we will produce a sample of size n out of the possible 2^5 binary strings. Each one of these samples will be used as a template to modify B. As S is built out of all 0's, a fully-adjacent neighbourhood is considered. Suppose $B = 1010101010101111 0$ and the bit to be optimized is the fourth bit. B_4 equals 0 in the example and its four adjacent neighbours are $B_5 = 1, B_6 = 0, B_7 = 1, B_8 = 0$. If one of the n samples is 11111 then B will be set to $B' = 1011111101010111 0$ provided B' has better fitness than B. The process is repeated in every bit of B once for every sample in the sample set.

Several complex local search strategies have been applied to the NK-landscapes domain. For example Merz in [32] uses various optimisation features with the aim of acelerating his MAs. Furthermore, in chapter 6 in this book he describes varios $K-opt$, $Lin-Kernighan$ and other sophisticated heuristics for NK problems. In the following case studies, initially explored in [25] and [57], we use simpler local searchers to serve only as a proof of concept. The evolved memes induce a variable-sampled $k-opt$ local search strategy. We say variable as k varies with the size of S and it can be as small as 0 or as large as $3 * k$. It is sampled as we do not exhaustively explore all the 2^{k+1} possible ways of settings the bits in a chromosome but rather take a reduced sample of size n.

3.2 Results

In previous sections we described our self-generating MAs. What sort of behaviors can we expect to see emerging? Four different scenarios needs to be analysed: low epistasis-poly-time solvable, high epistasis-poly-time solvable, low epistasis-NP-hard and high epistasis-NP-hard landscapes. The level of epistasis is controlled by the n and k. The closer k is to 0 the more negli-

gible the epistatic interactions among loci. If k grows up to $n-1$ then the induced problems is a random field. The transition between polynomial time solvability to NP-hardness depends on the type of neighborhood used as it was explained before. We should expect the emergence of short strings (i.e. $|S|$ not too big) for the low epistasis regimes while longer strings will be favored in high epistasis cases. We should be able to compare the length of the evolved local searcher with the k of the problem that is being solved. That is, we expect to see memes emerging with lengths close to k. We should probably also see distinct patterns of activity for the different problem regimes. The range of problems we experimented with are:

- low epistasis, poly-time solvable: $(50,1),(50,4)$ with adjacent neighbours.
- high epistasis, poly-time solvable: $(50,8),(50,10),(50,12),(50,14)$ with adjacent neighbours
- low epistasis, NP-hard: $(50,1),(50,4)$ with random neighbours.
- high epistasis, NP-hard: $(50,8),(50,10),(50,12),(50,14)$ with random neighbours.

3.3 Discussion

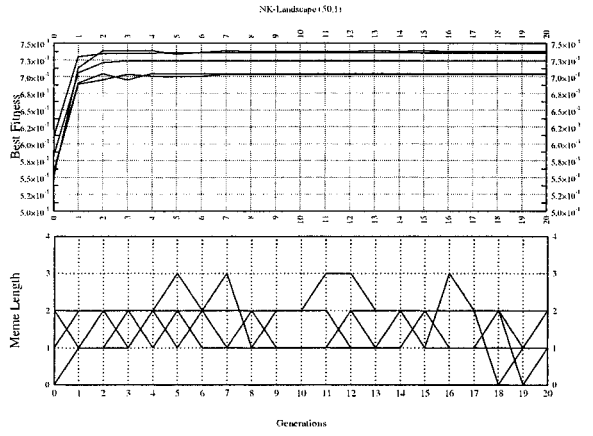
In the following figures we plot the evolution of the length of the meme associated with the fittest individual as a function of time and the evolution of fitness. For clarity, just 5 runs are depicted.

Low epistasis, poly-time solvable:

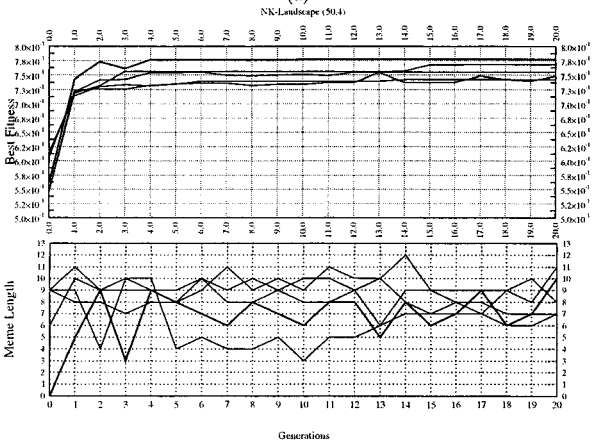
In Figures 5(a) and 5(b) we can observe the behaviour of the system. For the case $n = 50, k = 1$ the main activity occurs at the early generations (before generation 4). After that point the system becomes trapped in a local (possible global) optimum. The length of the memes evolved oscillates between 1 and 2. As the allowed length are restricted to be in the range $[0, 3 * k]$, the expected length of memes is 1.5. It is evident that the problem is solved before any creative learning can take place. When the Self-Generating MA is confronted with problem $n = 50, k = 4$ (a value of k just before the phase transitions mentioned in previous sections) the length of the meme in the best run oscillates between a minimum value of 3 (after generation 1) and a maximum of 10 for the run marked with a thick line (the best run). In this case the expected length (if a purely random rule was chosen) for a meme is 6 which is the most frequently visited value. For these simple NK-Landscape regimes, it does not seem to be of benefit to learn any specific meme, but rather, a random rule seems to suffice.

High epistasis, poly-time solvable:

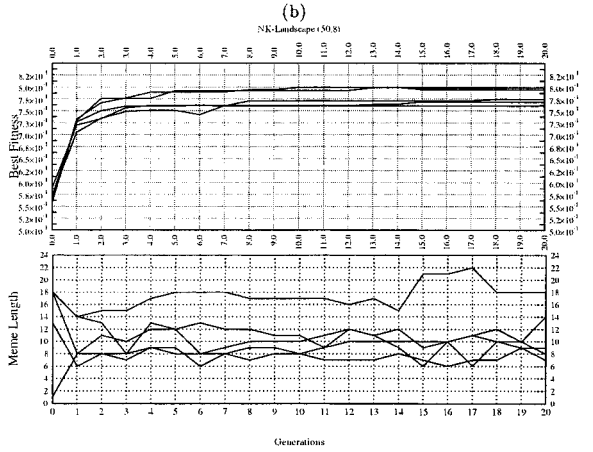
In Figure 5(c) we can see the system's behaviour for a value of k after the phase transition mentioned in [31] and [23]. In this case there is effective evolutionary



(a)



(b)



(c)

Fig. 5. *NK(50,1) in (a), NK(50,4) in (b) and NK(50,8) in (c). Adjacent neighbours.*

activity during the whole period depicted. Also, we can see clearly that the length of the meme employed by the most successful individual converges towards the value of k(in this case 8). If a purely random rule was used the expected length would have been 12. The case shown in Figure 6(a) is even clearer. All but one of the runs converge towards a meme length almost identical to $k = 10$, except for one that is very close to the expected length of 15.

The same trends can be seen in Figures 6(b) and 6(c) where meme lengths converge to values around to $k = 12$ and $k = 14$ respectively. It is interesting to note that although the values are very close to our predictions they do not remain at a fixed value but rather oscillates. This is a very intriguing behaviour as it resembles the variable-neighborhood nature of Lin-Kernighan, the most successful local search strategy for NK-Landscapes and other combinatorial problems. It will be interesting to investigate on the range of values that the Memetic Algorithms presented in [32] (which uses $K-opt$ and Lin-Kernighan) effectively employs; we speculate that the range of changes, i.e. the number of bits modified in each iteration of LS, will be close to the epistatic parameter of the problem instance.

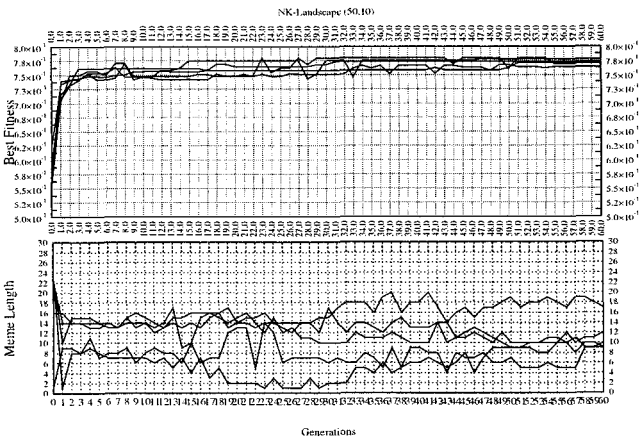
Low epistasis, NP-hard:

In Figure 7(a) we start to investigate the behaviour of the Self-Generating MA on the NP-Hard regime (i.e. the random neighborhood model). Figure 7(a) is similar to the adjacent neighborhoods version shown in Figure 5(a) except that oscillations are more frequent in the former. Comparisons between 7(b) and 5(b) reveal very similar trends.

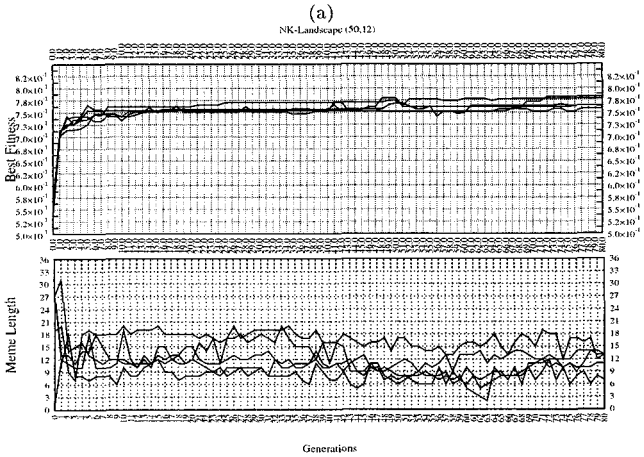
High epistasis, NP-hard:

The experiments with $(n = 50, k = 8)$ under the random neighbours model reveal marked differences with the consecutive neighbour model (see Figures 7(c) and 5(c) respectively). While in the later all the runs converged toward a meme length very close to k, the random model shows a richer dynamics. Meme length were divided into 3 groups. In one group, the emerged meme length were very close to the value of k, 8 in this case. The other two groups either continually increase the size of the memes or decreased it. Two of the most successful runs are identified with a cross or circle and each belong to a different group. Interestingly, the run that converges first to the local optimum is the one that uses very short memes. In contrast, the run that uses memes with length equivalent to a value of k show a continued improvement. It is important to note that none of the evolved memes converged towards the expected length of 12. Figure 8(a) seems to reveal a similar 3-grouped pattern.

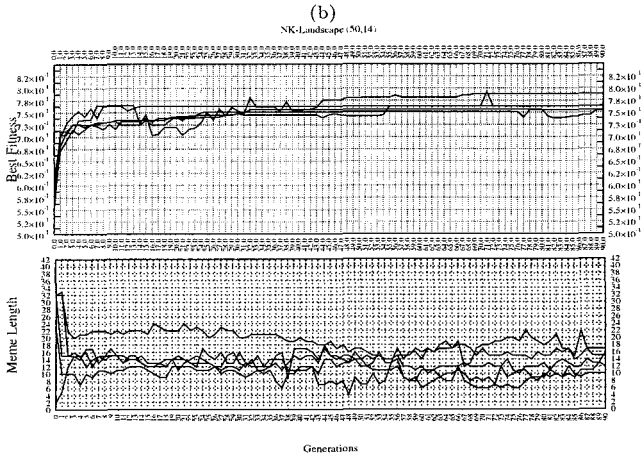
The runs that correspond to instances of $(n = 50, k = 12)$ differ notably from previous ones. The meme length seems to be converging towards a value



(a)

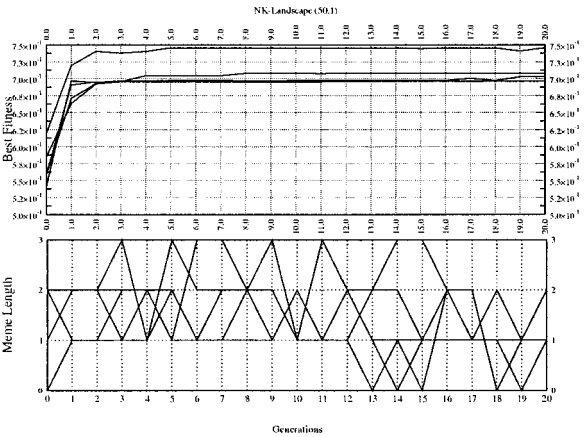


(b)

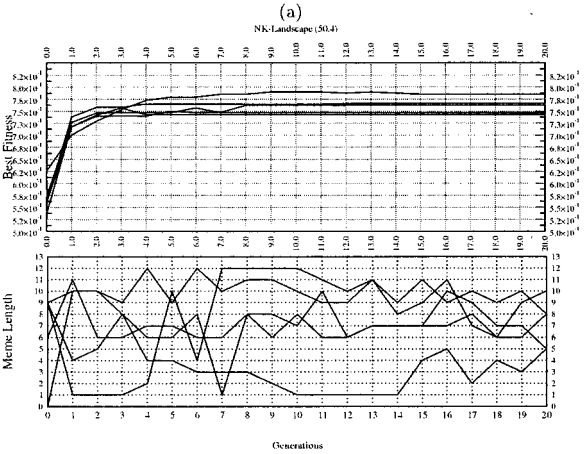


(c)

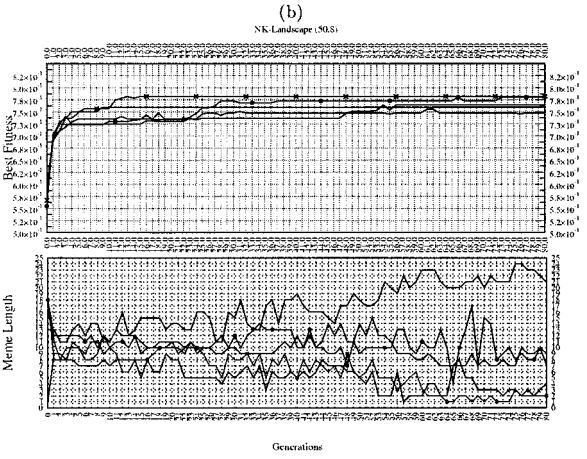
Fig. 6. *NK(50,10) in (a), NK(50,12) in (b) and NK(50,14) in (c). Adjacent neighbours.*



(a)

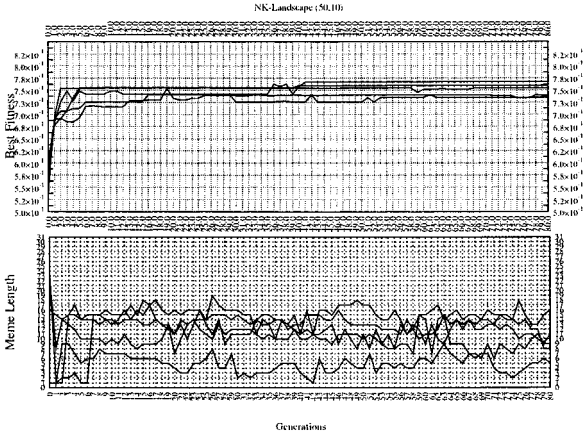


(b)

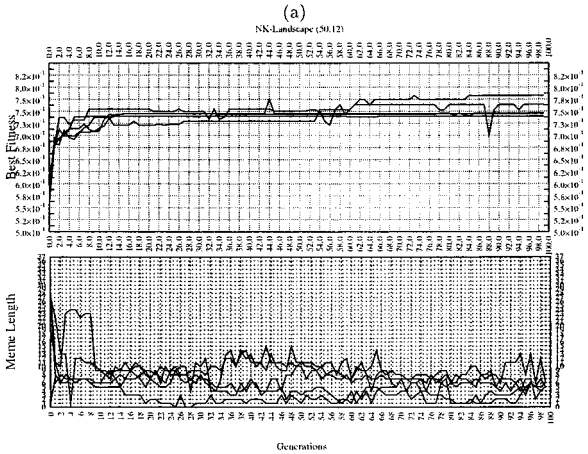


(c)

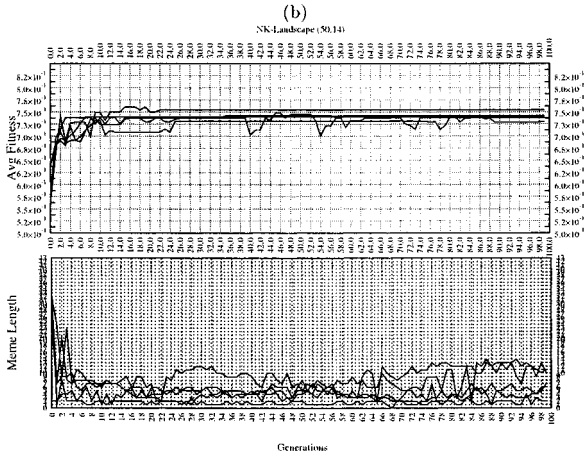
Fig. 7. *NK(50,1) in (a), NK(50,4) in (b) and NK(50,8) in (c). Random neighbours.*



(a)



(b)



(c)

Fig. 8. *NK(50,10) in (a), NK(50,12) in (b) and NK(50,14) in (c). Random neighbours.*

well below the expected length of 18 and even the epistatic value $k = 12$ for these problems. However, between generation 34 and 68 the meme lengths oscillates very close to $k = 12$ values. The next figure, 8(c), presents similar features as that of 8(b). However, now two clusters appear, one that suggest length around the value of k and another with length values of 6.

From the analysis of the previous figures we can see that our expectation that memes of length proportional to k will arise confirmed. However, other interesting features are evident. There are clear differences between memes that are evolved to solve the poly-time solvable cases and the NP-hard cases. In the first case, all the memes length for $k > 4$ converged toward values in the proximity of k. However, for the random neighborhood model and for high epistasis ($k > 4$) problems, the runs were clustered mainly around memes lengths close to k or close to around 6 (regardless the value of k). This is indeed a very interesting behaviour that deserves further studies as values of k in the range $[4, 5, 6]$ are *on the edge* of the phase transitions described in [31],[23] and [28]. That is, between 4,5 or 6 bits were the optimum number of bits that need to be considered to boost the efficiency of the search. Moreover, in the case of the NP-hard random neighbourhood with $k = 8$ three clusters are noted; we speculate that problems in this range are on the so called "edge of chaos" where emergent behaviours are more likely to occur[8],[20].

4 The Maximum Contact Map Overlap Experiments

We explore next the evolved local searcher as a supplier of building block in the context of a problem drawn from computational biology. A *contact map* is represented as an undirected graph that gives a concise representation of a protein's 3D fold. In this graph, each residue[6] is a node and there exists an edge between two nodes if they are neighbors. Two residues are deemed neighboors if their 3D location places them closer than certain threshold. Figures 9 & 10 show two contact maps. An *alignment* between two contact maps is an assignment of residues in the first contact map to residues on the second contact map. Residues that are thus aligned are considered equivalents. The value of an alignment between two contact maps is the number of contacts in the first map whose end-points are aligned with residues in the second map that, in turn, are in contact (i.e. the number of size 4 undirected cycles that are made between the two contact maps and the alignment edges). This number is called the *overlap* of the contact maps and the goal is to maximize this value. The complexity of Max CMO problem was studied in [18] and later in [23].

[6] A residue is a constitutent element of a protein.

4.1 Self-Generating Memetic Algorithms for MAX-CMO

The overall architecture of the Memetic Algorithm is similar to that described by the pseudocode in Figure 4. The backbone of the MA is a genetic algorithm in which chromosomes are represented by a vector $c \in [0, \ldots, m]^n$. Here m is the size of the longer protein and n the size of the shorter. A position j in c, $c[j]$, specifies that the j^{th} residue in the longer protein is aligned to the $c[j]^{th}$ residue in the shorter. A value of -1 in that position will signify that residue j is not aligned to any of the residues in the other protein (i.e., a structural alignment gap). Unfeasible configurations are not allowed, that is, if $i < j$ and $v[i] > v[j]$ or $i > j$ and $v[i] < v[j]$ (e.g., a crossing alignment)then the chromosome is discarded. It is simple to define genetic operators that preserve feasibilities based on this representation. Two-point crossover with boundary checks was used in [29] to mate individuals and create one offspring. Although both parents were feasible valid alignments the newly created offspring can result in invalid (crossed) alignments. After constructing the offspring, feasibility is restored by deleting any alignment that crosses other alignments. The mutation move employed in the experiments is called a sliding mutation. It selects a consecutive region of the chromosome vector and adds, slides right, or subtracts, slides left, a small number. The phenotypic effect produced is the tilting of the alignments. In [29] a few variations on the sliding mutation were described and used. Further implementation details can be found also in [23] and [5]. We describe next the make-up of memes.

The Local Search Procedure: Memes description for MAX-CMO

As mentioned in previous sections, we seek to produce a metaheuristic that creates from scratch the appropriate local searcher to use under different circumstances. A meme represents one particular way of doing local search. Memes can adapt through changes in their parameter set or through changes in the actions they perform. The local search involved can be very complex and composed of several phases and processes. In the most general case we want to be able to explore the space of all possible memes. One can achieve this by using a formal grammar that describes memes and by letting a genetic programming[21] based system to evolve sentences in the language generated by that grammar[23]. The sentences in the language generated by this grammar represent syntactically valid complex local searchers and they are the instructions used to implement specific search behaviors and strategies. To describe the particular representation employed to self-assemble the move operator used by a local search strategy we resort to a few examples.

In Figure 9 we can see two contact maps ready to be aligned by our algorithm. To simplify the exposition, both contact maps are identical (i.e. we are aligning a contact map with itself) and have a very specific pattern of contacts among their residues. In the present example a residue is connected to either its nearest neighbor residue, to a residue that is 4 residues away in

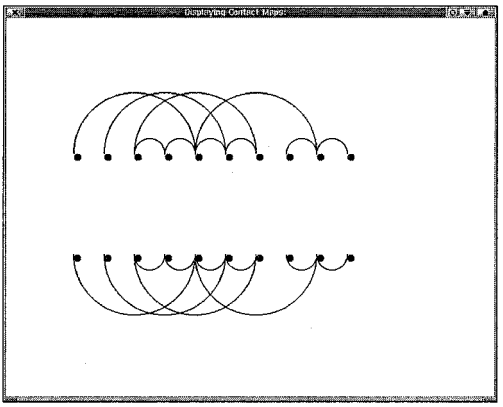


Fig. 9. *A contact map snapshot. The two randomly generated proteins have 10 residues.*

the protein sequence, or to both (with a given probability). In Figure 9 the contact map is 10 residues long, while in 10 it is 50 residues long (but with the same connectivity patterns). This contact pattern can be represented by the string $1-4$, meaning that the residue which occupies the ith position in the protein sequence is in contact in the native state with residues $(i+1)$th and $(i+4)$th. That is, the pattern $1-4$ is a succint representation of a possible building block which, if matched by the local searcher, could be propagated later on by crossover into other solutions.

An appropriate move operator for a local searcher acting in any of the contact maps on Figures 9 & 10 would be one that iterates through every residue in one of the contact maps, checking which residues on the lower contact map fulfills the pattern of connectivity and making a list of them. The same procedure would be applied to the top contact map producing a second list of residues. The local searcher then would pair residues of one list with residues of the second list thus producing a new and correct alignment which includes that building blocks.

The number of residues that verifies the pattern in each list puts an upper bound on how expensive the local search move operator can be. If the size of the first list is L_1 and the size of the second list is L_2, and without loss of generality we assume that $L_1 \leq L_2$ then there are at most $\sum_{i=1}^{i=L_1} \frac{L_2!}{(L_2-i)!}$. Clearly this number is too big to be searched exhaustively, this is why the previous grammar allows for the adaptation of the sample size. Moreover, although it is well known that real proteins present these contact patterns[10] it is impossible to know a priori *which* of these patterns will provide the best fitness improvement for a particular pair of protein structures. Hence, the Self-Generating MA needs to discover this itself.

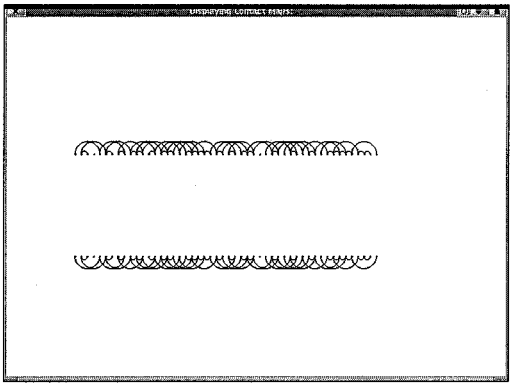


Fig. 10. *A contact map snapshot. The two randomly generated proteins have 50 residues and the patterns of contacts are similar to those in Fig. 9.*

If the graphs to be aligned were different (in the previous cases a graph was aligned with itself for the sake of clarity), then a move operator able to account for that variation in patterns must be evolved.

The defined move operator induces a neighborhood for every feasible alignment. If an alignment s is represented as explained above and L_1, L_2 are the list of vertices that matches the move operator, then every *feasible* solution that can be obtained by adding to s one or more alignments of vertices in L_1 with vertices on L_2 is a neighbor of s. The other components of a meme will then decide how to sample this neighborhood and which solutions to accept as the next one. As this paper is an account of the initial investigations we performed on the use of SGMA, we fixed several aspects of the memes that could otherwise be evolved. In this paper all memes employ first improvement ascent strategy and they are applied after crossover. The sample size was either 50 or 500 and the local search was iterated 2 times.

As described in the introduction, there were three memetic processes: imitation, innovation and mental simulation. Upon reproduction, a newly created offsprings inherits the meme of one of its parents accordingly to the simple inheritance mechanism described in [28]. In addition to this mechanism, and with a certain probability (called "imitation probability"), an agent could choose to override its parental meme by copying the meme of some successful agent in the population to which it was not (necessarily) genetically related. In order to select from which agent to imitate a search behavior, a tournament selection of size 4 was used among individuals in the population and the winner of the tournament was used as role model and its meme copied. Innovation was a random process of mutating a meme's specification by either extending, modifying or shortening the pattern in a meme (either before or after the $\rightarrow$). If during 10 consecutive generations no improvement was produced by either the local search or the evolutionary algorithm a stage of mental simulation was

started. During mental simulation, each individual (with certain probability) will intensively mutate its current meme, try it in the solution it currently holds, and if the mutant meme produces an improvement, both the newly created solution and the meme will be accepted as the next state for that agent. That is, mental simulation can be considered as a guided hill-climbing on memetic space. If ten mental simulation cycles finished without improvements, then metal simulation was terminated and the standard memetic cycle resumed.

4.2 Results

We designed a random instance generator with the purpose of parameterizing the complexity of the contact map overlap problems to be solved. The input to the random instance generator is a list of the form:

$r\ d\ n\ p_1\ pr_1\ p_2\ pr_2\ \ldots\ p_n\ pr_n$ where r is the number of residues in the randomly generated contact map, d is the density of random edges (i.e. noise) and n is the number of patterns in the contact map. For each of the n patterns two numbers are available, p_i and pr_i, where p_i specifies that a residue j is connected to residue $j+p_i$ with probability pr_i for all $i \in [1,n]$. That is, every pattern occurs with certain probability in each residue, thus an upper bound on the expected number of contacts is given by $r*d+r*\sum_{i=1}^{i=n} pr_i \leq r*(n+d)$. In our experiments $r \in \{10,50,100,150,200,250\}$, $d = 0.01$ and $n \in \{1,2,3,4\}$, that is, contact maps as short as 10 residues and as long as 250 residues were considered. For each contact map length, every possible number of patterns was used, this gives rise to 24 pairs of (r,n) values. For each pair, 5 random instances were generated spanning from low density contact maps to high density contact maps[7]. A total of 120 instances were generated. From all the possible parings of contact maps we randomly choose a total of 96 pairs to be aligned by means of 10 runs each.

We present next comparisons of the performance of a Genetic Algorithm versus that of the SGMA. In this experiment we would like to elucidate whether the overhead of learning suitable local searchers is amortized along the run and whether our proposed approach is ultimately useful. In order to run the experiments we implemented a GA as described previously. We were able to reproduce the results of [29] and [5] hence we considered our implementations as equivalent to the earlier ones. The difference between the GA and the SGMA are described below. In graphs 11,12,13 and 14 we compare the overlap values[8] against the *first hitting times*. First hitting time (FHT) is the time (in number of fitness evaluations) at which the best value of a run was encountered. Each graphs presents the results for 1,2,3 and 4 patterns respectively and for a range of contact maps sizes. The particular parameters used in the

[7] The program to generate random contact maps was written in java 1.1.8 as is available by request from the author.

[8] A higher overlap value means a better structural alignment.

GA are 0.15, 0.75 for mutation and crossover probabilities, and a (50, 75) replacement strategy. The Self-Generating MA uses 0.15,0.75,1.0,1.0,1.0,1.0 for the probabilities of mutation, crossover,local search, imitation, mental simulation and innovation respectively. The algorithms uses the same replacement strategy and for both local search and mental simulation a cpu budget of 50 samples is allocated.

4.3 Discussion

The graphs in 11,12,13 and 14 are good representatives of the results obtained with the two types of algorithms. That is, under a variety of changes to the parameter values mentioned above the results remain equivalent to those shown here.

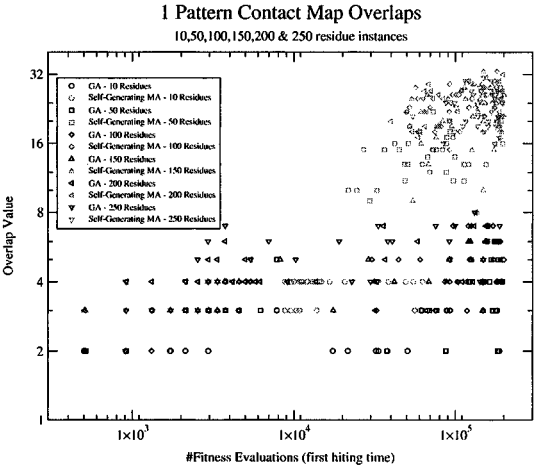


Fig. 11. *Comparison of the first hitting times and the quality of overlaps obtained for GA and SGMA on increasingly difficult randomly generated instances. Complexity increases as a function of residues number. Contact maps present one pattern.*

From Figures 11,12,13 and 14 we can see that the Self-Generating Memetic Algorithm produces a much better amortized overlap value than the simple GA. That is, if enough time is given to the SGMA, it will sooner or later discover an appropriate local searcher move that will supply new building blocks. In turn, this will deliver an order of magnitude better overlaps than the Genetic Algorithm. Also, it seems that the GA is oblivious to the size (i.e. residues number) of the contact maps as it seems to produce mediocre local optima solutions even when given the maximum cpu time allocation (in these experiments $2 * 10^5$ fitness evaluations) for the whole range of 10 to 250 residues. The GA converges very quickly into local optima. This is seen in the graphs by bands parallel to the $x-axis$ over the range of energy evaluations for low overlap values. However, as the SGMA continuously improves its solutions,

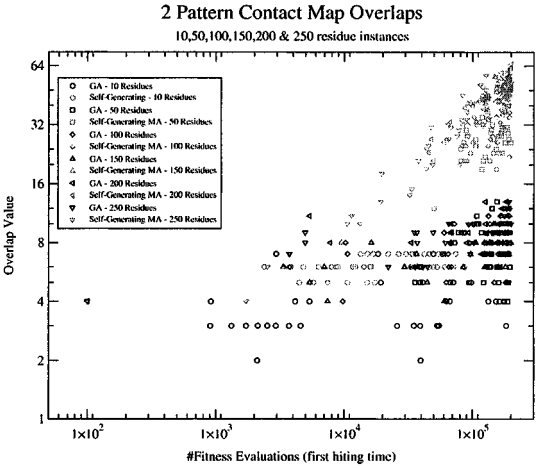


Fig. 12. *Comparison of the first hitting times and the quality of overlaps obtained for GA and SGMA on increasingly difficult randomly generated instances. Complexity increases as a function of residues number.Contact maps present two patterns.*

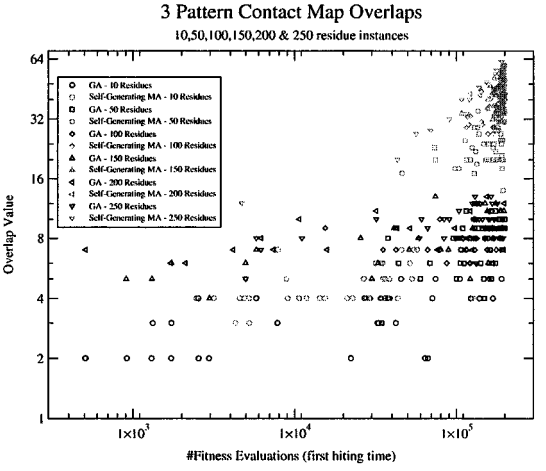


Fig. 13. *Comparison of the first hitting times and the quality of overlaps obtained for GA and SGMA on increasingly difficult randomly generated instances. Complexity increases as a function of residues number.Contact maps present three patterns.*

it is not until very late in the execution (i.e. to the right of the $x - axis$) that the best solutions are found.

In contrast to the GA, the SGMA (as expected) is sensitive to the number of residues in the contact maps involved, that is, longer contact maps require larger cpu time to come up with the best value of the run (which is seen in the graph in the clustering patterns for the different residues number). Another

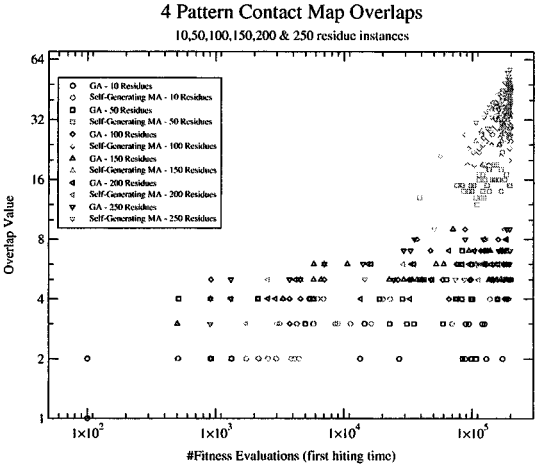


Fig. 14. *Comparison of the first hitting times and the quality of overlaps obtained for GA and SGMA on increasingly difficult randomly generated instances. Complexity increases as a function of residues number.Contact maps present four patterns.*

important aspect to note is that both the $x - axis$ and the $y - axis$ are represented in logarithmic scales. Taking this into consideration it is evident that the quality of the overlaps produced by the SGMA are much better than those produce by the GA. As it is evident from the graphs, for sufficiently small instances (e.g all the 10 residues long and some of the 50 residues long) it is not worth using the SGMA as it requires more cpu effort to produce same quality of overlaps as the GA.

On the other hand, as the number of residues increases beyond 50, then instances are sufficiently complex to allow for the emergence of suitable local searchers in time to overtake and improve on the GA results. Also, as the number of patterns that are present in the instances increases both algorithms, as expected, require larger amounts of CPU to come up with the best solution of a run. However, it is still seen that the GA is insensitive to the number of residues, while the SGMA is clustered in the upper right corner (of Figure 14). This indicates that during all its execution the algorithm is making progress toward better and better solutions, the best of which is to be found near the end of the run. Moreover, this behavior indicates that the SGMA is not prematurely trapped in poor local optima as is the GA.

The ability of the SGMA to overcome local optima comes from the fact that the evolved local searchers will introduce good building-blocks that match the particular instance. This supply of building-blocks is essential for a synergistic operation of both the local searcher and the genetic operators. That is, using Goldberg's notation [17], we have that for the SGMA the *take over time* t^* is greater than the *innovation time* t_i, which allows the algorithms to continuously improve. In Figure 15 10 runs of the GA are compared agaisnt

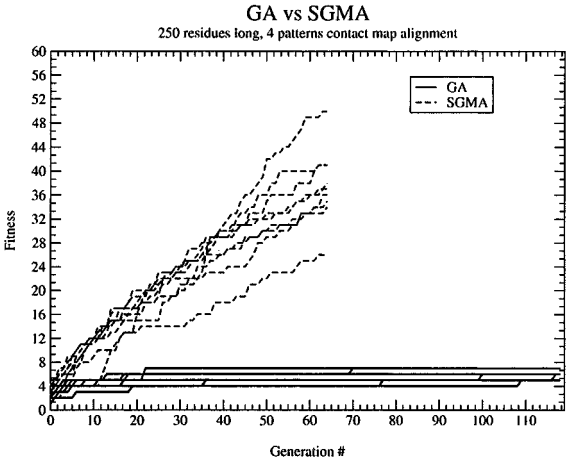


Fig. 15. *Representative example of GA and SGMA runs for a 250 residues and 4 patterns instance.*

10 runs of the SGMA. It can be seen that the GA runs get trapped very early (around the 20th generation) in poor local optima while the SGMA keeps improving durin all the run. All the runs in Figure 15 use the same total number of fitness evaluations.

5 Conclusions

In this chapter we discussed concepts arising from Memetic theory that could be use to produce a new breed of optimisation algorith,s We tied some of these memetic ideas with the concept of "Self-Generating Metaheuristics" and we exemplified the use of the resulting algorithms in two hard combinatorial problems.

The Memetic algorithms described in this paper do not resort to human-designed local searchers but rather they assemble *on-the-fly* the local search strategies that best suits each particular situation.

In this paper we argued that from an optimization point of view there are obvious advantages in self-assembling the local search behaviours for memetic algorithms. MAs that can self-generate the local searchers will be able to adapt to each problem, to every instance within a class of problem and to every stage of the search. A similar strategy could be use in other metaheuristics (e.g. Simulated Annealing, Tabu Search, Ant Colonies, GRASP, etc) where more sophisticated GP implementations might be needed to co-evolve the used operators.

One of the reasons for the success of the SGMA is that the evolved local searchers act as a (low and medium order) building block supplier. These

continuous supply of building blocks aids the evolutionary process to improve solutions continuously by producing a more synergistic operation of the local and gloabl operators.

It is our hope that researchers confronted with new problems for which there are not "silver bullet" local search heuristics (like is the case for TSP and Graph Partitioning where $K-$opt and Lin-Kernighan are known to be extremely efficient) with which to hybridize a Memetic Algorithm will try the obvious: the Dawkins method of self-assembling of local search behaviors. That is, use memes to evolutionary self-assemble appropriate local search strategies.

References

1. R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA Journal on Computing*, 6(2):126–140, 1994.
2. S. Blackmore. *The Meme Machine.* Oxford University Press, 1999.
3. E.K. Burke, J.P. Newall, and R.F. Weare. A memetic algorithm for university exam timetabling. In E.K. Burke and P. Ross, editors, *The Practice and Theory of Automated Timetabling*, volume 1153 of *Lecture Notes in Computer Science*, pages 241–250. Springer Verlag, 1996.
4. E.K. Burke and A.J. Smith. A memetic algorithm for the maintenance scheduling problem. In *Proceedings of the ICONIP/ANZIIS/ANNES '97 Conference, Dunedin, New Zealand*, pages 469–472. Springer, 24-28 November 1997.
5. R.D. Carr, W.E. Hart, N. Krasnogor, E.K. Burke, J.D. Hirst, and J.E. Smith. Alignment of protein structures with a memetic evolutionary algorithm. In W.B. Langdon, E. Cantu-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M.A. Potter, A.C. Schultz, J.F. Miller, E. Burke, and N. Jonoska, editors, *GECCO-2002: Proceedings of the Genetic and Evolutionary Computation Conference*, 2002.
6. L.L. Cavalli-Sforza and M.W. Feldman. *Cultural Transmission and Evolution: A Quatitative Approach.* Princeton University Press, Princeton, NJ., 1981.
7. F.T. Cloak. Is a cultural ethology possible. *Human Ecology*, 3:161–182, 1975.
8. P. Coveney and R. Highfield. *Frontiers of Complexity, the search for order in a chaotic world.* faber and faber (ff), 1995.
9. P. Cowling, G. Kendall, and E. Soubeiga. A hyperheuristic approach to scheduling a sales summit. In E. Burke and W. Erben editors, editors, *Theory of Automated Timetabling PATAT 2000, Springer Lecture Notes in Computer Science,*, pages 176–190. Springer, 2001.
10. T. E. Creighton, editor. *Protein Folding.* W. H. Freeman and Company, 1993.
11. R. Dawkins. *The Selfish Gene.* Oxford University Press, New York, 1976.
12. R. Dawkins. The extended phenotype. 1982.
13. W.H. Durham. *Coevolution: Genes, Culture and Human Diversity.* Stanford University Press, 1991.
14. P.M. França, A.S. Mendes, and P. Moscato. Memetic algorithms to minimize tardiness on a single machine with sequence-dependent setup times. In *Proceedings of the 5th International Conference of the Decision Sciences Institute, Athens, Greece*, July 1999.

15. L.M. Gabora. Meme and variations: A computational model of cultural evolution. In L.Nadel and D.L. Stein, editors, *1993 Lectures in Complex Systems*, pages 471–494. Addison Wesley, 1993.
16. A. Godzik, J.Skolnick, and A.Kolinski. A topology fingerprint approach to inverse protein folding problem. *Journal of Molecular Biology*, 227:227–238, 1992.
17. D.E. Goldberg. *The Design of Innovation: Lessons from and for Competent Genetic Algorithms.* Kluwer Academic Publishers, 2002.
18. D. Goldman, S. Istrail, and C. Papadimitriou. Algorithmic aspects of protein structure similarity. *Proceedings of the 40th Annual Symposium on Foundations of Computer Sciences*, pages 512–522, 1999.
19. D. Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. In *Artificial Life II: Proc. of the 2nd Conf. on Artificial Life*, 1992.
20. S.A. Kauffman. *The Origins of Order, Self Organization and Selection in Evolution.* Oxford University Press, 1993.
21. J.R. Koza, F.H. Bennet, D. Andre, and M.A. Keane. *Genetic Programming III, Darwinian Invention and Problem Solving.* Morgan Kaufmann Publishers, 1999.
22. N. Krasnogor. Co-evolution of genes and memes in memetic algorithms. In A.S. Wu, editor, *Proceedings of the 1999 Genetic And Evolutionary Computation Conference Workshop Program*, 1999.
23. N. Krasnogor. http://www.cs.nott.ac.uk/ ~nxk/papers.html. In *Studies on the Theory and Design Space of Memetic Algorithms.* Ph.D. Thesis, University of the West of England, Bristol, United Kingdom., 2002.
24. N. Krasnogor. Self-generating metaheuristics in bioinformatics: The protein structure comparison case. *To appear in the Journal of Genetic Programming and Evolvable Machines. Kluwer academic Publishers*, 5(2), 2004.
25. N. Krasnogor and S. Gustafson. Toward truly "memetic" memetic algorithms: discussion and proof of concepts. In D.Corne, G.Fogel, W.Hart, J.Knowles, N.Krasnogor, R.Roy, J.E.Smith, and A.Tiwari, editors, *Advances in Nature-Inspired Computation: The PPSN VII Workshops.* PEDAL (Parallel, Emergent and Distributed Architectures Lab). University of Reading. ISBN 0-9543481-0-9, 2002.
26. N. Krasnogor and S. Gustafson. The local searcher as a supplier of building blocks in self-generating memetic algorithms. In J.E. Smith W.E. Hart and N. Krasnogor, editors, *Fourth International Workshop on Memetic Algorithms (WOMA4).* In GECCO 2003 workshop proceedings, 2003.
27. N. Krasnogor and J.E. Smith. A memetic algorithm with self-adaptive local search: Tsp as a case study. In D. Whitley, D. Goldberg, E. Cantu-Paz, L. Spector, I. Parmee, and Hans-Georg Beyer, editors, *GECCO 2000: Proceedings of the 2000 Genetic and Evolutionary Computation Conference.* Morgan Kaufmann, 2000.
28. N. Krasnogor and J.E. Smith. Emergence of profitable search strategies based on a simple inheritance mechanism. In L. Spector, E.D. Goodman, A. Wu, W.B. Langdon, H. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshj, M.H. Garzon, and E. Burke, editors, *GECCO 2001: Proceedings of the 2001 Genetic and Evolutionary Computation Conference.* Morgan Kaufmann, 2001.
29. G. Lancia, R. Carr, B. Walenz, and S. Istrail. 101 optimal pdb structure alignments: a branch-and-cut algorithm for the maximum contact map overlap prob-

lem. *Proceedings of The Fifth Annual International Conference on Computational Molecular Biology, RECOMB 2001*, 2001.
30. M.W.S. Land. Evolutionary algorithms with local search for combinatorial optimization. *Ph.D. Thesis, University of California, San Diego*, 1998.
31. W.G. Macready, A.G. Siapas, and S.A. Kauffman. Criticality and parallelism in combinatorial optimization. *Science*, 261:56–58, 1996.
32. P. Merz. *Memetic Algorithms for Combinatorial Optimization Problems:Fitness Landscapes and Effecitve Search Strategies.* Ph.D. Thesis, Parallel Systems Research Group. Department of Electrical Engineering and Computer Science. University of Siegen., 2000.
33. P. Moscato. Memetic algorithms: A short introduction. In D. Corne, F. Glover, and M. Dorigo, editors, *New Ideas in Optimization.* McGraw-Hill, 1999.
34. P.A. Moscato. Problemas de otimizacão np, aproximabilidade e computacão evolutiva:da prática à teoria. *Ph.D Thesis, Universidade Estadual de Campinas,Brasil*, 2001.
35. M.J. Oates, D.W. Corne, and R.J. Loader. Tri-phase profile of evolutionary search on uni- and multi-modal search spaces. *Proceedings of the Congress on Evolutionary Computation (CEC2000)*, 1:357–364, 2000.
36. B. Olsson. A host-parasite genetic algorithm for asymmetric tasks. In C. Nédellec and C. Rouveirol, editors, *Machine Learning: ECML-98 (Proceedings of the 9th European Conference on Machine Learning)*, pages 346–351. Springer-Verlag, 1998.
37. B. Olsson. *Algorithms for Coevolution of Solutions and Fitness Cases in Asymmetric Problem Domains.* PhD thesis, University of Exeter, 1999.
38. B. Olsson. Handling asymmetric problems with host-parasite algorithms, 1999.
39. J. Paredis. Chapter c7.4: Coevolutionary algorithms. In T. Back, D.B. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*, page C7.4. IOP publishing Ltd and Oxford Unviersity Press, 1997.
40. M.A. Potter and K.A. De Jong. A cooperative coevolutionary approach to function optimization. In *Parallel Problem Solving From Nature*, 1994.
41. P.L. Privalov. Physical basis of the stability of the folded conformations of proteins. In T.E. Creighton, editor, *Protein Folding.* W.H. Freedman and Company, 1999.
42. P.J. Richerson and R. Boyd. A dual inheritance model of the human evolutionary process: I. basic postulates and a simple model. *Journal of Social and Biological Structures*, I:127–154, 1978.
43. J. Schaffer and A. Morishima. An adaptive crossover distribution mechanism for genetic algorithms. In J.J. Grefenstette, editor, *Proceedigns of Second International Conference on Genetic Algorithms.* Lawrence Erlbaum, 1987.
44. O. Sharpe. Introducing performance landscapes and a generic framework for evolutionary search algorithms. *Proceedings of the Congress on Evolutionary Computation (CEC2000)*, 1:341–348, 2000.
45. J.E. Smith. Co-evolution of memetic algorithms : Initial results. In Beyer Fgenandez-Villacans Merelo, Adamitis and Schwefel (eds), editors, *Parallel problem solving from Nature - PPSN VII, LNCS 2439.* Springer Verlag, 2002.
46. J.E. Smith and T.C. Fogarty. Operator and parameter adaptation in genetic algorithms. *Soft Computing*, pages 81–87, 1997.
47. Jim Smith and T.C. Fogarty. Self adaptation of mutation rates in a steady state genetic algorithm. In *Proceedings of the Third IEEE International Conference on Evolutionary Computing*, pages 318–323. IEEE Press, 1996.

48. R.E. Smith and E. Smuda. Adaptively resizing populations: Algorithms, analysis and first results. *Complex Systems*, 1(9):47–72, 1995.
49. E. G. Talbi, Z. Hafidi, and J-M. Geib. A parallel adaptive tabu search approach. *Parallel Computing*, 24(14):2003–2019, 1998.
50. E.D. Weinberger and A. Fassberg. Np completness of kauffman's n-k model, a tuneably rugged fitness landscape. In *Santa Fe Institute Technical Reports*, 1996.

Designing Efficient Genetic and Evolutionary Algorithm Hybrids

Abhishek Sinha[1], Ying-ping Chen[2], and David E. Goldberg[3]

[1] Siebel Systems
San Mateo, CA 94402, USA
Abhishek.Sinha@siebel.com
[2] Department of Computer Science and Department of General Engineering
University of Illinois, Urbana, IL 61801, USA
ypchen@illigal.ge.uiuc.edu
[3] Department of General Engineering
University of Illinois, Urbana, IL 61801, USA
deg@illigal.ge.uiuc.edu

Summary. Genetic and evolutionary algorithms (GEAs) are being employed to solve a wide range of problems in search and optimization. Most real-world applications use GEAs in combination with domain specific methods to achieve superior performance. Such combinations, often referred to as hybrids, stand to gain much from a system-level framework for efficiently combining global searchers such as GEAs with domain-specific and local searchers. This chapter presents the foundations for such a framework. The theory herein attempts to attain the optimal division of labor between global and local search so that the desired solution quality can be obtained in the minimum time, or given a fixed time budget, the best solution quality can be obtained. It relies on a two-fold decomposition: the hybrid is composed of a global searcher and a local searcher, and the search space is divided into basins of attraction from where the local search can lead to the desired solution quality. The framework allows us to choose between different schedules so as to maximize chances of success. The framework utilizes knowledge of run duration theory and uses the quality of solution at each generation to compute the parameters needed by the theory. The study also looks at characteristics of a class of functions (known as traps) that determine the speedups that can be obtained from using local search.

1 Introduction

Genetic and evolutionary algorithms (GEAs) have enjoyed considerable success as search and optimization techniques. They have been successfully applied to a wide range of problems across disparate domains. Oftentimes, GEA success is gauged by its efficiency in finding the solution. Although certain state-of-the-art GEAs, such as the hierarchical Bayesian optimization algorithm (hBOA) [18, 19], can solve difficult problems in subquadratic time,

better efficiency is still required for GEAs to tackle large scale real-world applications. Based on the methodology of design decomposition and with the help of existing facetwise models of GEAs, four principled efficiency enhancement techniques were proposed [8] to make GEAs capable of handling hard problems in practice: parallelization, time utilization, evaluation relaxation, and hybridization. These principled efficiency enhancement methods can be used in conjunction with one another so that modest gains from each method are compounded to result in substantial speedups.

This chapter focuses on one of the principled efficiency enhancement techniques—hybridization. GEAs are good at exploring the search space to find promising regions but have limited capability of fine-grained search. On the other hand, local and problem-specific search methods are adept at fine-grained search. Hence, intuitively, a combination of a GEA with a local searcher, often referred to as a *hybrid*, should yield significant performance improvements. Past work has indicated that GEAs alone can seldom outperform a hybrid in real-world applications. The primary goals of optimization [10] can be stated as (a) achieving a specified solution quality in minimum time or (b) achieving the best possible solution quality in a given time. Hybrids have been known to be successful at both tasks. One of the key issues in designing hybrids is the division of labor between global and local searchers. Therefore, the present work aims to find the optimal switch-over point between global and local searchers to make the most out of a fixed budget of fitness evaluations.

1.1 Rationale behind the Framework

The framework presented in this chapter is based on a number of assumptions which may require information unavailable in practice. However, the goal of this study is to provide a system-level framework to tackle what is, otherwise, an intractable problem. Another significant merit of the proposed hybrid theory is its reliance on few parameters. The assumptions for the theory in this study is described as follows. First, we assume knowledge of the search space: the sizes and number of the basins and the best solution. For many problems this information may not be available. Second, we assume knowledge of the time-to-criterion (TTC) for each basin. The calculation or estimation of this value may be feasible for many functions and local searchers but may not be possible for others. Also, it is assumed that the parameters are stationary. But for many global searchers (such as genetic algorithms), the probabilities, P_G and P_{λ_a}, will not be constant across iterations. Later, the theory is modified appropriately to handle such global searchers.

1.2 Organization

This chapter is organized as follows. Section 2 presents a taxonomy to categorize GEA hybrids according to the purpose of hybridization and the architecture of the hybrid. Section 3 introduces the theory for optimizing hybrids.

The hybrid theory utilizes a two-fold decomposition—the hybrid consists of global and local searchers and the search space is divided into targets and basins of attraction from which local search can lead to a target. The theory is then verified with the global method as random search and the local method as a quasi-Newton method with a simple multimodal test function. The utility of local search for a class of trap functions is demonstrated in Sect. 4. Population-sizing requirements for such hybrids are touched upon. The relation between speedups from local search and the problem size is also explored. Section 5 starts with a comparison of a GA with random search vis á vis the global-local hybrid theory. The theory is extended to GEAs to see how it helps to decide among different schedules for hybrids. The run duration theory enables us to estimate the theory parameters and extend the global-local hybrid theory to GEAs. Finally, Sect. 6 summarizes the study, describes the significance of the study for researchers and practitioners, and makes suggestions for future work.

2 Hybrid Taxonomy

GEA hybrids are being increasingly used in real-world applications. Numerous studies have used local searchers to achieve varying objectives. These studies brought forward the critical issues of hybrid design. It would be useful to view these developments in perspective before proceeding toward better design of hybrids. This section explores the state-of-the-art in GEA hybrids and ties together past developments through a taxonomy of GEA hybrids. A taxonomy of GEA hybrids will view past developments in perspective, help to gain insight into their shortcomings, and enable them to be improved. GEA hybrids can be classified along the following lines: purpose of hybridization and hybrid architecture. These classifications are not mutually exclusive. The aim here is to highlight the motivations for hybridization and get a high-level view of what has been studied in the GEA community.

2.1 Purpose of Hybridization

Motivations for incorporating local search have been numerous, and it is useful to look at hybrids from this perspective. This classification divides various hybrids according to the objectives of hybridization, including

- **Exploitation**: This class consists of hybrids using a local search technique for fast convergence to the optimum once the GEA has led the search to promising regions. The present study focuses on this class of hybrids.
- **Repair**: These hybrids use local search to generate legal solutions from parents and to repair infeasible solutions.
- **Parameter optimization**: In this class, the GEA technique is used to optimize the parameters of a secondary method.

- **GEA functionality substitution/enhancement**: In this class, secondary methods are used to perform some function of the GEA or enhance the performance of the GEA through better control.

The above classes cover the motivations for using GEA hybrids. The book edited by Davis [5] also provides strong motivations and guidelines for hybridization. Other categorizations based on the interaction between GEAs and local search are also possible and are discussed next.

2.2 Hybrid Architecture

GEA hybrids can also be classified according to the nature of the coupling between the GEA and the secondary method (or how and when the secondary method is utilized). The following is a modified version of the classification proposed by Yen et al. [27]. Most hybrids fit into one of the categories:

- **Pipelined hybrids**: These are characterized by two distinct sequential stages one of which is a GEA. This category can be further divided into: preprocessor, primary or postprocessor, and staged.
- **Asynchronous hybrids**: This paradigm involves the asynchronous cooperation of two methods such that intermediate values of each method might be utilized later in the run.
- **Hierarchical hybrids**: This class includes procedures with multiple levels of optimization utilizing different techniques, at least one of which being a GEA.
- **Embedded hybrids**: This class is characterized by a secondary method embedded inside some GEA module and can be further subdivided according to the stage of embedding into the following sub-classes: initialization, fitness evaluation, crossover, mutation, and special operators.

This classification of hybrids is also not mutually exclusive. There may be several schemes which fit into more than one category. Furthermore, GEA hybrids can also be classified based on other criteria, such as the other methods utilized in a GEA hybrid. Interested readers are referred to more comprehensive surveys [12, 22] on this topic.

3 Hybrid Theory and Verification

In last section, the need for a high-level theory for hybrid design was discussed, and the theory proposed by Goldberg and Voessner [10] was mentioned in this context. In this section, first, the basic approach of decomposing the hybrid and the search space as well as the parameters for the theory are described. Two formulations—one for minimizing the time to find the solution and one for maximizing the reliability—are then presented that enable the user to decide the proper mix between local and global search. Finally, the reviewed hybrid theory is empirically verified with random search as the global searcher and a quasi-Newton method as the local searcher.

3.1 Global-Local Hybrid Theory

This part is mainly drawn from other works [10, 7], and the interested reader is urged to refer to these papers for further details. A typical hybrid, H, consists of a global method G and a local method L. The theory looks at a continuum of choices. G alone and L alone lie at the opposite ends of this continuum. An iteration of H consists of one iteration of the G to generate a candidate solution which serves as the starting point for L which is invoked multiple times each consuming no more than an allowable time $\lambda_a : 0 \geq \lambda_a \leq \lambda_{max}$. This process continues until we exceed an allowable time T_a or the desired solution quality is obtained. Choosing a suitable value for λ_a is key to optimizing the hybrid. The solution quality is the solution accuracy target $\phi \leq \phi_\tau$. In Fig. 1, β_i (depicted as tessellated polygons) are the basins of attraction within which L can lead to the target solution which are depicted as islands τ_i. Figure 2 illustrates the notion of solution quality. Here ϕ_1 is said to be of a higher quality than ϕ_2 (for a maximization problem). Setting the desired solution quality to a lower value may bring more basins into the picture thereby increasing the chances of G hitting a basin. Also note that not all basins are useful. For example, getting into the rightmost basin in Fig. 2 is pointless since we will never reach a solution of desired quality from there.

The sought solution is better than some target value $\phi_\tau (= \phi^* + \Delta\phi$, where ϕ^* is the globally optimal solution and $\Delta\phi$ is the amount by which the sought solution quality differs from ϕ^*). $\Delta\phi$ would be positive (negative) for a minimization (maximization) problem. Now we consider the possible ways in which we can get to the target islands, τ_i. The union of the targets, the global region, $R_G = \bigcup_i \tau_i$. The probability of hitting R_G in a single invocation of the global searcher is denoted as P_G. For random search with uniform distribution, P_G may be calculated by summing the areas of the targets and dividing by the total area of the search space. For more complex global searchers, this calculation will be more complicated.

The local time-to-criterion (TTC) values, λ_i, are defined as the average number of time units required to get to the target starting from within the basin of attraction β_i. Although the time taken to reach the local optimum would depend on the exact point in the basin where we start from, here we consider a single λ_i over the whole basin for simplicity. The probability of hitting basin β_i (exclusive of the target τ_i) with an invocation of G is denoted by P_i. Suppose G lands in a basin from where L does not reach a solution of desired quality or in a basin where L fails to converge in $\lambda \leq \lambda_{max}$ time units. Such regions are called *dead zones*. The probability of hitting a dead zone is denoted by P_D and can be calculated as

$$P_D = 1 - P_G - \sum_i P_i \,. \tag{1}$$

For simplicity, we assume that P_G and P_i are constant over all iterations of the global searcher. This assumption is relaxed later and can be handled with

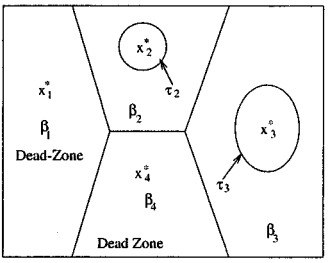


Fig. 1. This 2-D sketch of the search space shows the target islands, τ_i, basin of attraction under local method L to those targets, β_i, and dead zones

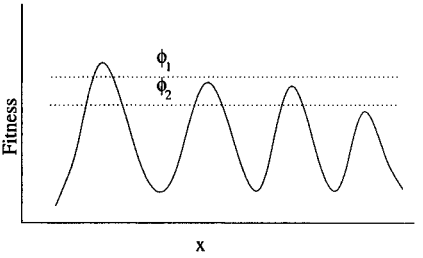


Fig. 2. Notion of solution quality. For maximization, ϕ_1 is a higher solution quality level than ϕ_2. The x axis represents the phenotype space

slight modifications to the theory. The global search is assumed to take one unit time, and the local search time is calculated relative to that value. Calling the allowable local time constant λ_a, and the average local time constant $\overline{\lambda}$, the solution time T consumed in n global-local iterations is

$$T = (1 + \overline{\lambda})n \,. \tag{2}$$

Some of the basins may have a local TTC higher than the allowable time for local search, λ_a. Hence, the probability of hitting the global zone can be found by summing the probability of hitting the global region initially (by the global searcher) and the probability of hitting the basins with a TTC less than λ_a:

$$P_{\lambda_a} = P_G + \sum_{i:\tau_i \neq 0, \lambda_i \leq \lambda_a} P_i \,, \tag{3}$$

where P_{λ_a} is the probability of hitting the global region when $\lambda = \lambda_a$. While seeking a solution of certain quality, we either seek to maximize the probability (or *reliability*) of reaching the solution in a given time budget, or we seek the desired solution in the minimum time with given reliability.

Minimum Time Formulation

The reliability is measured in terms of the probabilistic error, α, that is defined as the probability of not reaching the desired solution. Therefore, the lower the value of α the higher the reliability. For a specified allowable error α_a, the reliability condition can be written as

$$\alpha_a = (1 - P_{\lambda_a})^n \,, \tag{4}$$

where n is the number of global-local iterations. By eliminating n with (2) and minimizing, we get

$$\min(\lambda_a + 1)\frac{\ln \alpha_a}{\ln(1 - P_{\lambda_a})} \,. \tag{5}$$

This gives us the minimum time required to reach a solution with a specified allowable error α_a. The minimization of the above expression yields an optimal choice of λ_i.

Maximum Reliability Formulation

Based on (2), we have $n = \frac{T_a}{\lambda_a+1}$. The maximum allowable time $\lambda_{max} \leq T_a - 1$. We need to minimize the probabilistic error in order to maximize reliability. Minimizing the error and substituting for n gives

$$\min(1 - P_{\lambda_a})^{\frac{T_a}{\lambda_a+1}} \,. \tag{6}$$

These two formulations—minimum time to achieve the desired solution quality and maximum reliability of reaching the desired solution quality—cover the most sought after objectives in optimization. The problem is reduced to finding the λ_a which minimizes the above expression. Although it assumes knowledge of the search space, the theory promises a systems-level design capability for global-local hybrids.

3.2 Verification for Random Search

In this section, the hybrid theory is verified using random search as the global searcher [10, 21]. We consider three cases—uniform λ_i across all basins, variation in λ_i across basins, and variation in λ within a basin—and see how the theory can be used to handle each case. The theory helps us decide the optimal combination of global and local search. Also, the effect of desired solution quality on the optimal hybrid is explored.

Experiments

This theory has been verified previously using random search as G and a quasi-Newton method, the Broyden, Fletcher, Goldfarb, and Shanno (BFGS) method [20] as L for uniform λ across several basins [10]. The test function used in that work was

$$f(x,y) = \begin{cases} \frac{d_i}{r_i^2}(\overline{r}^2)(2 - \frac{\overline{r}^2}{r_i^2}) - d_i & \text{for } \overline{r}^2 \leq r_i^2 \\ 0 & \text{otherwise} \end{cases}, \tag{7}$$

where $\overline{x} = x - cx_i$, $\overline{y} = y - cy_i$, $\overline{r}^2 = \overline{x}^2 + \overline{y}^2$, and $c_i = \{(2.0, 8.0), (3.0, 4.0), (5.0, 7.0), (7.0, 8.5), (7.0, 4.0)\}$, $r_i = \{1.5, 2.0, 0.5, 1.0, 2.5\}$, $d_i = \{2.0, 3.0, 2.0, 4.0, 2.0\}$. The global minima is -4.0 at $(7.0, 8.5)$. This function has been used for all of the following experiments except where mentioned otherwise.

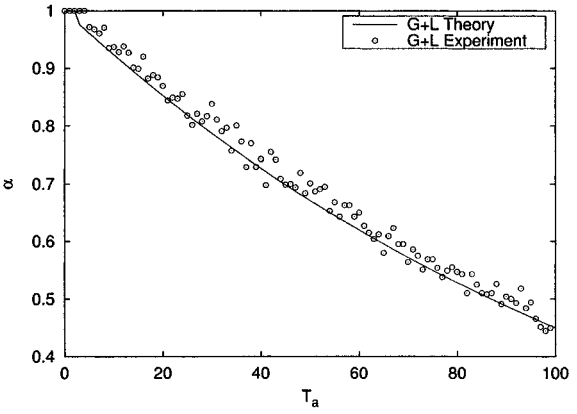
The termination criterion for all simulations was a maximum error of 0.01%. For random search, P_G is calculated by summing the areas of the targets and dividing by the total area of the space. A Baldwinian approach is used to handle the results from the local searcher. In the Baldwinian approach, only the fitness of an individual is replaced with that obtained from the local searcher, while in the Lamarckian approach, both the fitness and the genotype of an individual are replaced [13, 26, 3]. Since in many real-world applications, function evaluations tend to be the bottleneck, appropriate weights are assigned to function evaluations, and this computation is considered a measure of execution time. In this work, each function evaluation and each derivative evaluation has a unit cost associated with it.

Case I: Uniform λ_i Across Basins

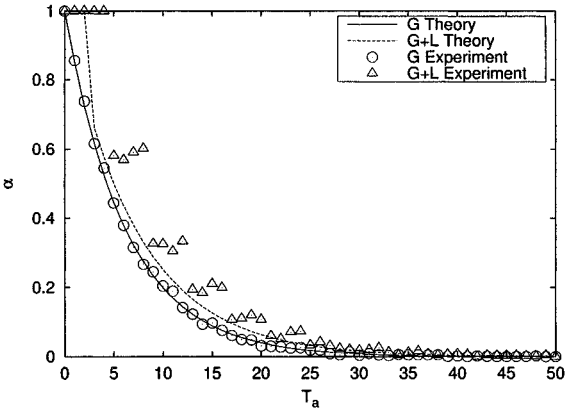
The simplest case is when all the basins in the search space have the same TTC values, λ_i. We have two possible situations: (a) the probability of the global searcher hitting a solution by itself is negligible, and (b) the probability of global searcher hitting a solution on its own is considerable. We illustrate these cases, and the theory is able to predict beforehand whether we should go with G alone or $G + L$ in combination.

First, the TTC value was estimated for all the basins by averaging the time taken to reach a solution of desired quality from within the basin over 1000 trials. Since the function is quadratic and the local searcher is a quasi-Newton method, we get a small value, $\lambda_i \approx 3.0$, for each basin. Hence, the allowable time for local search, λ_a, was set to 3.0 for this case.

This is illustrated by the following two cases. In the first case, we set $\phi_\tau = -3.99$. This yields $P_D = 0.9686$ and $P_G = 0.0001$. Because ϕ_τ is so close to its global value, G alone has a negligible chance of finding the target. In this case, $G + L$ has a better chance of succeeding. Figure 3(a) shows that the experimental results are a good match with the theory. In the second case, $\phi_\tau = -1.0$. Now, $P_D = 0.5760$ and $P_G = 0.1490$. G has a high chance



(a) $\phi_\tau = -3.99$, $P_G = 0.0001$, $\lambda_i \approx 3.0$, $\lambda_a = 3.0$. In this case, $G + L$ yields lower error



(b) $\phi_\tau = -1.0$, $P_G = 0.1490$, $\lambda_i \approx 3.0$, $\lambda_a = 3.0$. In this case, G yields lower error

Fig. 3. The probabilistic error α is shown as a function of the allowable time T_a for the case of uniform λ_i across basins when G = Random Search and $L = BFGS$

of succeeding on its own. Going with $G + L$ is costlier in terms of function evaluations. Figure 3(b) shows that the experiment matches the theory closely.

Case II: Variation in λ_i Across Basins

With different λ_i for different basins, choosing an appropriate value for λ_a becomes critical. The possible choices for λ_a are the different λ_i. A higher λ_a is appropriate if the cumulative probability of success increases sufficiently. The following procedure is adopted for choosing an optimal value of λ_a. First, the basins are arranged in ascending order of λ. Locally optimal choices are obtained by comparing the λ values of the ith basin and the $(i+1)$th basin on the basis of the probabilistic error. The error is given by (4) with $n = T_a/(\lambda_a + 1)$:

$$\alpha_a = (1 - P_{\lambda_a})^{T_a/(\lambda_a+1)} = P_D^{T_a/(\lambda_a+1)} . \tag{8}$$

The locally optimal choice with the least error gives the globally optimal choice of λ_a. This procedure yields $\lambda_a = 8.0$ as the optimum value for the aforementioned function.

Although the BFGS method is still the local searcher, we simulate the varying λ_i as follows. In particular, we simply *assume* that different basins have different λ_i values. The assignment was $\lambda_i = \{6.0, 12.0, 4.0, 10.0, 8.0\}$. Whenever the global searcher lands in a basin, it is assumed that the local searcher takes the assigned amount of time to reach a target. The desired solution quality was $\phi_\tau = -1.0$. Figure 4(a) shows the results for this case. The least error is with $\lambda_a = 8$, which was predicted by the theory. For clarity, only three of the five possible choices of λ_a were plotted. Experimental results for $\lambda_a = 8$ and $\lambda_a = 12$ are shown separately in Fig. 4(b). The other choices of λ_a (which are not shown) led to inferior performance.

Case III: Variation of λ Within a Basin

For many real-world functions and local search methods, the TTC values depend on the starting point of the local searcher. It can be illustrated by the Griewank function [25] (for the two-dimensional case):

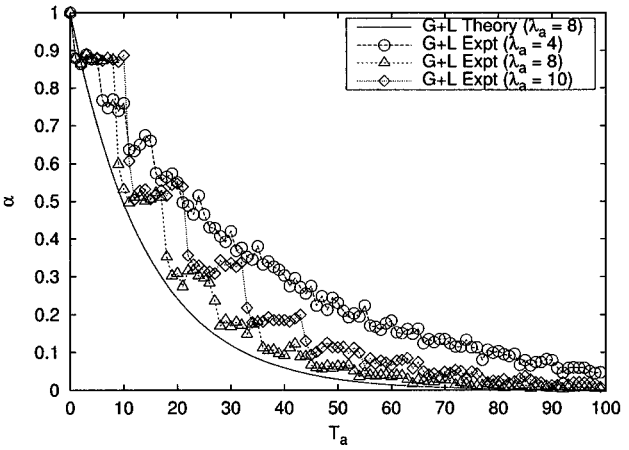
$$f(x_1, x_2) = 1 + \frac{{x_1}^2 + {x_2}^2}{4000} \cos(x_1) \cos(\frac{x_2}{\sqrt{2}}) , \tag{9}$$

where $x_1, x_2 \in [-512, 511]$. This is a differentiable, multimodal function with the global minima as 0.0 located at $x_1 = 0.0$ and $x_2 = 0.0$.

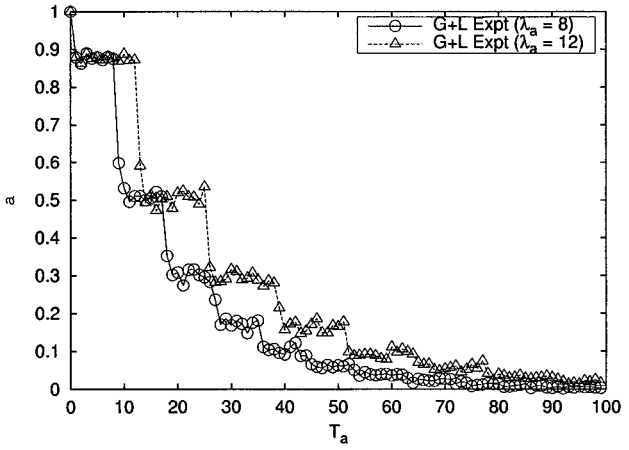
For a solution quality of $\phi_\tau = 1.5\ \lambda$ varied from 3.0 to 23 for each basin. All values (except 4) in this range were observed. For convenience, only a portion of the space was considered. The area of each basin was approximated by a

Table 1. The probabilistic error for each choice of λ_a

Choice of λ_a	4.0	6.0	**8.0**	10.0	12.0
Error $= (1 - P_{\lambda_a})^{\frac{1}{\lambda_a+1}}$	0.998	0.967	**0.956**	0.960	0.957



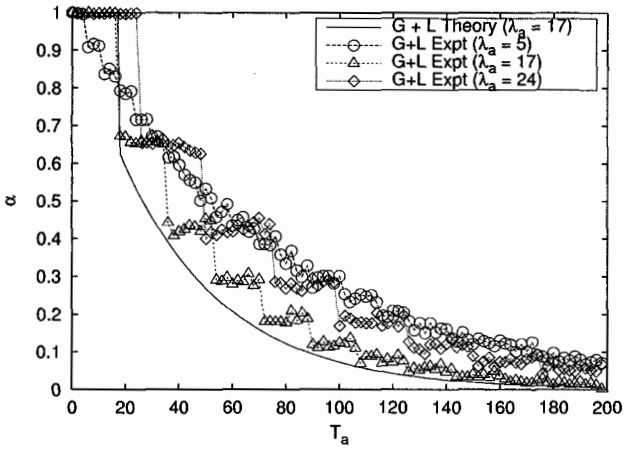
(a) The lowest error is obtained for $\lambda_a = 8.0$, as suggested by the theory



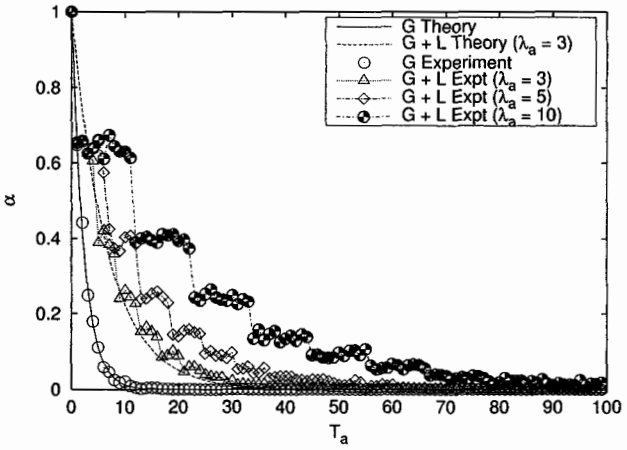
(b) This shows the similar behavior between $\lambda_a = 8, 12$. But $\lambda_a = 8$ has better performance

Fig. 4. The probabilistic error α is shown as a function of the allowable time T_a for the case of uniform λ_i across basins when G = Random Search and L = BFGS. Different basins were assumed to have the following TTC values $\lambda_i = \{6.0, 12.0, 4.0, 10.0, 8.0\}$

circle. To calculate P_i, λ was chosen so that 90% of the observed λ were below this value. The above procedure reduced the effective area of the basin by 10%.



(a) Here $\phi_\tau = 0.02, P_G = 0.002$ and $\lambda_a = 17.0$.
In this case, $G + L$ yields lower error



(b) Here $\phi_\tau = 1.5, P_G = 0.356$ and $\lambda_a = 3.0$.
In this case, G alone performs better

Fig. 5. The probabilistic error α is shown as a function of the allowable time T_a when G = Random Search and $L = BFGS$, with variation in λ_i within basins

This was factored into the calculation of P_i. Since the behavior was similar across basins, λ_i were taken to be uniform. Figure 5(a) shows the results for a desired solution quality $\phi_\tau = 0.02$. The results match the theory. Here,

$P_G = 0.002, P_D = 0.6237, \lambda_a = 17.0$. The global searcher cannot succeed without the help of L. Figure 5(b) shows the results for a desired solution quality $\phi_\tau = 1.5$. Here, $P_G = 0.356, P_D = 0.55, \lambda_a = 3.0$. With a higher P_G, we see that G alone performs better than $G + L$. According to the theory one should proceed with G only if $P_D > (1 - P_G)^{\lambda'}$ which is the case here. The theory holds in this test condition.

4 Problem Characterization and Population Sizing

Characterization of problems where local search can improve the efficiency of genetic search is a crucial aspect of hybrid design. In many problems, hybrids may slow down the search for the optimum, if applied without careful consideration. Therefore, identification and characterization of features which can aid the local searcher becomes very important. This section delves into this and another relevant issue: population sizing for hybrids. Efficient population sizing is critical for getting the most out of a fixed budget of function evaluations. The section starts by describing existing models for population sizing. A class of trap functions is studied, and features which aid the local searcher are identified. Then, the effect of these features on speedups is also studied. Finally, empirical results for population sizing are presented.

The rest of this work assumes the following with regard to the GA and the test problems. The building blocks (BBs) are assumed to be tightly linked. Selectorecombinative GAs (mutation probability set to 0) and only fixed length, binary encodings are used. Also, non-overlapping populations are used. Tournament selection with a tournament size of two is used throughout. All fitness functions are assumed to be stationary, and uniform BB crossover has been used to avoid BB disruption.

4.1 Population-Sizing Models

Decision models play an integral role in the population sizing models reviewed in this section. Such models attempt to determine the chances of choosing the individual with the best BB at a locus in the presence of noise coming from other loci in the individual. Goldberg et al. [9] considered m partitions in an individual. *Partition* refers to the locus of one complete BB. The fitness signal difference between the best and the second best BBs is denoted by d. They used statistical decision making to derive the required population size in terms of the number of partitions and the signal difference. The interested reader is urged to refer to that study for further details.

The gambler's ruin model [11] for population sizing improves upon the decision model. A random walk in a single dimension with two absorbing states—either the gambler ends up with all of his opponent's cash or he loses all his capital—was utilized to model convergence in GAs. Each partition can be assumed to converge independently. The run starts with a few of the BBs

in the partition correct. We assume that as the run proceeds, the probability of one more BB converging correctly remains constant. At the end of the run, all BBs in a partition are converged to the correct BB. An important (although conservative) assumption in this model is that the competition is always between individuals with the best and the second best BB. The resulting expression [11] for population size is given by

$$n = -2^{k-1} \ln(\alpha) \frac{\sigma_{bb} \sqrt{\pi m'}}{d}, \tag{10}$$

where σ_{bb} is the BB variance, $m' = m - 1$ is the number of partitions contributing to the noise, and d is the signal difference.

4.2 Test Functions

This section describes the test functions used in this study. All of them are functions of unitation (the number of ones in the BB). We also look at the characteristics of these functions to see why they are suitable for this study.

OneMax Domain

OneMax is probably the most common test function for GAs. The fitness is given by the unitation of the string. The contribution of each allele is equal and independent. A BB consists of a single allele. OneMax does not pose any problems for a GA but serves as a baseline for designing better GAs.

A useful extension to OneMax is the problem where a number of BBs are concatenated. This is referred to as the OneMax BB function [14]. The contribution of each BB is scaled down to one so that the fitness of the optimal individual is m, the number of BBs. The minimum fitness of each BB and the whole string is 0. For a BB of size k, the fitness is x/k, where x is the unitation of the BB. Such BBs may be concatenated to increase the problem size.

MaxTrap

Another useful class of test function in GA design has been deceptive functions [6, 23]. These function were designed to deceive GAs toward converging to sub-optimal solutions. The basic assumption that short optimal BBs combine to form longer optimal BBs is violated. At the first few levels, this holds true, but at higher levels, the BBs do not lead to the global optimum. The maximal deceptive function, or MaxTrap, for a BB of size k is where BBs till size $k-1$ lead to a sub-optimal solution. The function can be written as:

$$f(x) = \begin{cases} 1 & \text{if } x = k \\ (1-d)\frac{k-1-x}{k-1} & \text{otherwise} \end{cases}, \tag{11}$$

where d is the signal difference between the best and the second best BBs.

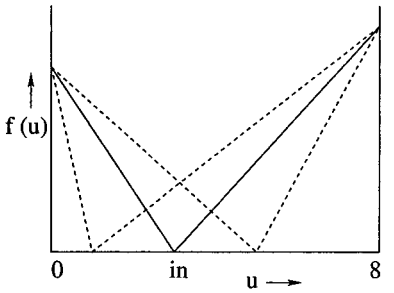


Fig. 6. The spectrum of unitation functions with OneMax at one end and MaxTrap at the other for an 8-bit BB

4.3 Utility of Local Search for a Class of Trap Functions

For fixed length BBs, a whole spectrum of trap functions similar to the MaxTrap function can be defined. Figure 6 shows this range where we get a different function for each choice of the inflection point, *in*. At one end, the function is OneMax; at the other end, the function is the MaxTrap. The expectation is that the contribution from local search should decrease as we move *in* from left to right. This is because of the decrease in the extent of the "slope" afforded by the function for a hillclimber. An important point is that for OneMax, local search alone is the best algorithm since the function is unimodal and thereby, "hillclimbing easy". But beyond that (for $in = 1 - 7$), we encounter false peaks in all the functions. Hence, local search alone will not do the job, and it becomes necessary to use an algorithm which can preserve diversity so that the search does not get stuck on any of the false peaks.

Table 2. Population size requirements with different amounts of Lamarckian local search on different functions for $k = 8$ and $l = 80$ using bisection method. LS denotes the number of function evaluations spent on local search per individual per generation. The sought solution quality was 0.98 (proportion of correct BBs)

Inflection	LS = 0	LS = 1	LS = 10	LS = 20
0	1900	1	1	1
1	1850	50	90	90
2	1750	40	90	90
3	1700	50	90	90
4	1400	40	90	90
5	1350	90	90	90
6	1250	190	160	160
7	1100	1050	900	800

These intuitions were corroborated by empirical evidence which show that as we move *in* from left to right, the gain from adding local search to the GA diminishes. The problem was constructed by concatenating $m = 10$ BBs of length $k = 8$. The minimum population size requirements for different functions and different amounts of local search are shown in Table 2. These empirical results were obtained by averaging over 100 runs. The solution quality sought was 0.98 (proportion of correct BBs). The architecture was a staged hybrid wherein a fixed amount of local search is applied to each individual during each generation.

The local searcher used is a Lamarckian next-ascent hillclimber (NAHC), which works in the following manner. First, a randomly selected bit is inverted. If this improves the fitness, the change is accepted; otherwise, it is not. This incurs the cost of a single function evaluation. The number of function evaluations per individual per generation is a parameter of the hybrid. A selectorecombinative GA is used with uniform BB crossover with a probability of 1.0. The performance metric for measuring performance was the proportion of correct BBs at the end of the run.

For each function, the following algorithms were tested: the GA alone and hybrids with 1, 10, and 20 evaluations of local search per individual per generation. The results are shown in Fig. 7. The fastest convergence is observed with 1 evaluation spent on local search for all cases except for $in = 7$. The decline in speedup from local search is marked beyond $in = 3$. "Too much" local search may lead to sub-optimal solutions. This is seen for $in = 6$ when $LS = 10$ and higher results in failure to converge to the correct BBs.

An important observation here is the decrease in population sizing requirements when using local search. Table 2 shows that the required population size initially decreases as the amount of local search is increased, but after a point, larger population sizes are required. A high amount of local search on a small population can quickly lead to loss of diversity and premature convergence. The decrease in population sizing requirements can be explained as follows. Since local search constantly leads toward basin optima, it results in a decrease in the fitness variance. Because the population size is directly dependent on variance, we see a decrease in the population size requirements.

Another interesting point is the variation in speedup from local search with increase in problem size. Intuitively, from the gambler's ruin model we expect the required population to grow as the square root of the problem size (l). With a small amount of local search, the overall convergence behavior is not affected drastically. As a result, we expect the speedups to be inversely proportional to the square root of the problem size. Empirical results confirm this intuition. Figure 8 compares the experimental results with the theory for different trap functions. Problem sizes used were 80, 160 and 320. One unit of 95% Baldwinian local search was used per individual per generation. The solution quality sought was 0.98 (proportion of correct BBs) and $k = 8$. The speedup was computed as the ratio of function evaluations needed by a GA alone to that required by the hybrid to achieve the sought solution quality.

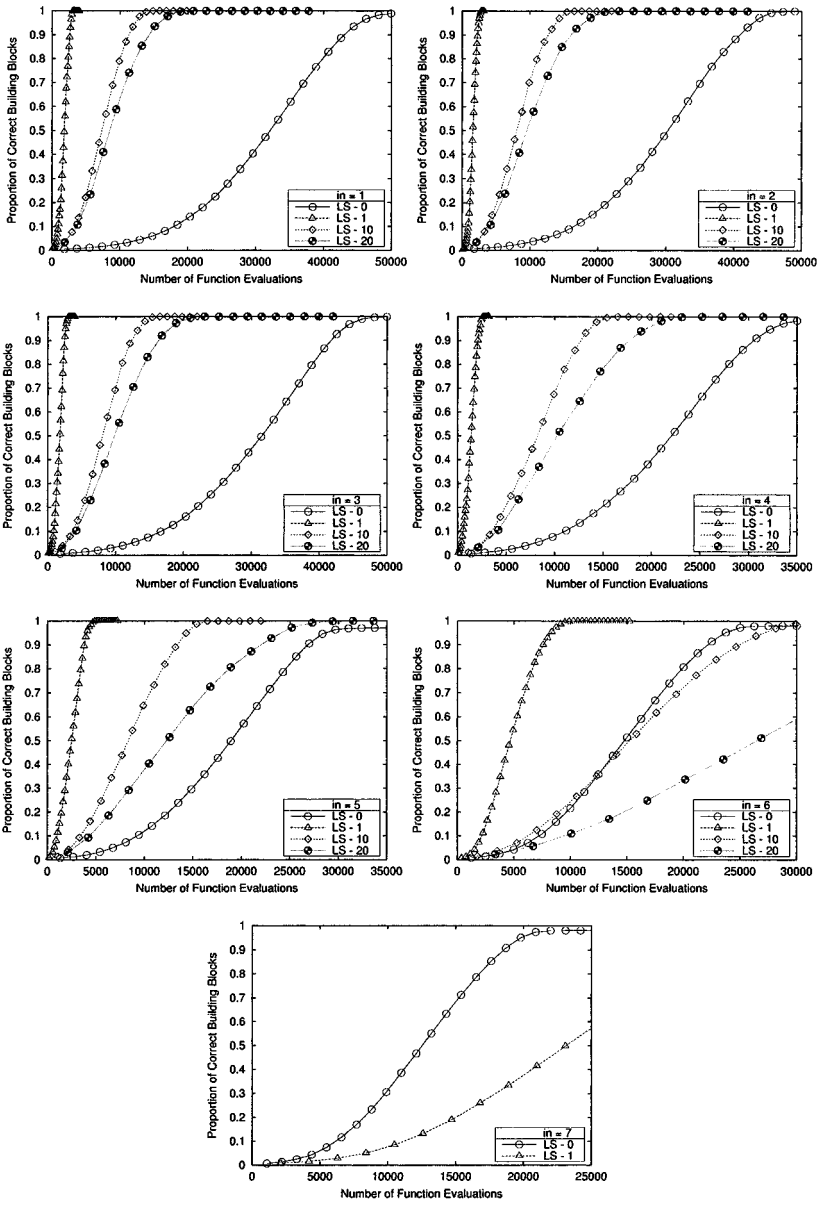


Fig. 7. The utility of different amounts of local search is depicted for various trap functions. LS gives the amount of Lamarckian local search per individual per generation. LS = 0 corresponds to a GA alone. For $in = 7$, adding local search only degrades the performance, so the corresponding results for LS > 1 are not shown

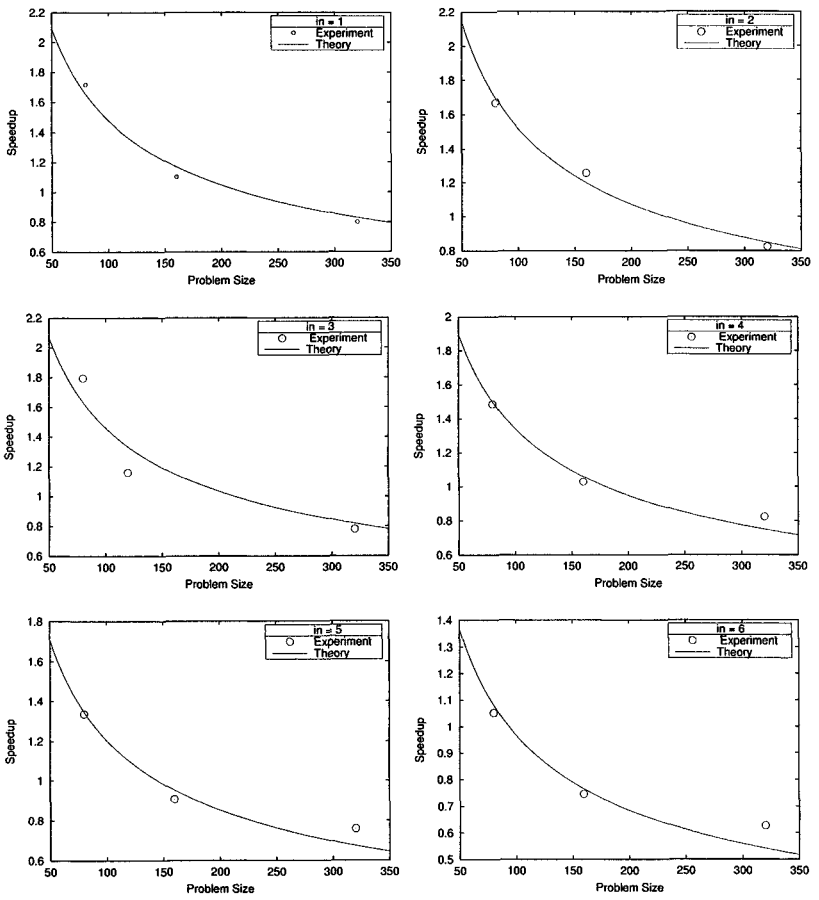


Fig. 8. The speedup from local search for different trap functions is shown as a function of problem size. The theoretical curves are for $c/\sqrt{l}$. One unit of 95% Baldwinian local search was used per individual per generation. The solution quality sought was 0.98 and $l = 80, 160, 320$ and $k = 8$. The speedup was computed as the ratio of function evaluations needed by a GA alone to that required by the hybrid

5 Incorporating GAs as Global Searchers

In previous sections, the hybrid theory was verified with random search as the global search. Our next goal is to extend the theory to GAs. The main problem here is the estimation of the theory parameters that also happen to be non-stationary. We start with a discussion of how a GA affects the search compared to random search in the context of the hybrid theory. Run duration theory is then reviewed and used to estimate the theory parameters. The probabilistic

error is estimated for GA hybrids. Using this error as the performance metric, the theory allows us to choose from different hybrid schedules.

5.1 Genetic Algorithm Versus Random Search

With uniform random search as the global searcher, the probabilities of hitting the global region, P_G, and that of hitting a basin of attraction, P_B, remain constant during the process. However, with a GA as the global searcher, these probabilities increase in successive generations. For the following experiments, the probabilities were computed by observing the number of times an optimal individual was found in the population over 1000 runs. A generation-wise random search was used wherein a population of individuals was random created at each generation. This ensured that equal number of function evaluations were spent on the GA and random search. The population sizes (for different functions) for the GA were set to those found in the previous section. Strings of length 80 with BB size $k = 8$ were used. Figures 9 and 10 show the results. For different trap functions, the probability of getting at least one optimal individual in the population and the proportion of individuals within the basin indicate that the GA easily outperforms random search.

5.2 Run Duration Theory

Run duration theory is an important aspect of GA design. It has been studied in terms of takeover time models and convergence time models. Takeover time models estimate the time for the best individual to proliferate in the population. Convergence time models consider the time taken for the average fitness of the population to converge. This section briefly describes the convergence time model for the OneMax domain [17, 24, 1, 15, 16].

The basic equation [4, 17] governing GA dynamics is given as

$$\mu_{t+1} = \mu_t + I\sigma_t \ , \tag{12}$$

where μ_t and σ_t denote the population fitness mean and variance at generation t, and I is the selection intensity. This enables us to predict the average fitness in successive generations. For the OneMax domain, the population fitness mean and variance can be expressed in terms of p_t as $\mu_t = lp_t$ and $\sigma_t = \sqrt{lp_t(1-p_t)}$, where l is the problem size. This result [17, 24] comes from the assumption that the probability of "1" at any locus is given by a binomial distribution with probability p_t. Substituting this in (12), we get

$$p_{t+1} - p_t = I\sqrt{\frac{p_t(1-p_t)}{l}} \ . \tag{13}$$

This equation can be solved for the proportion of correct alleles as

$$p_t = \frac{1}{2}\left(1 + \sin\frac{It}{\sqrt{l}}\right) \ . \tag{14}$$

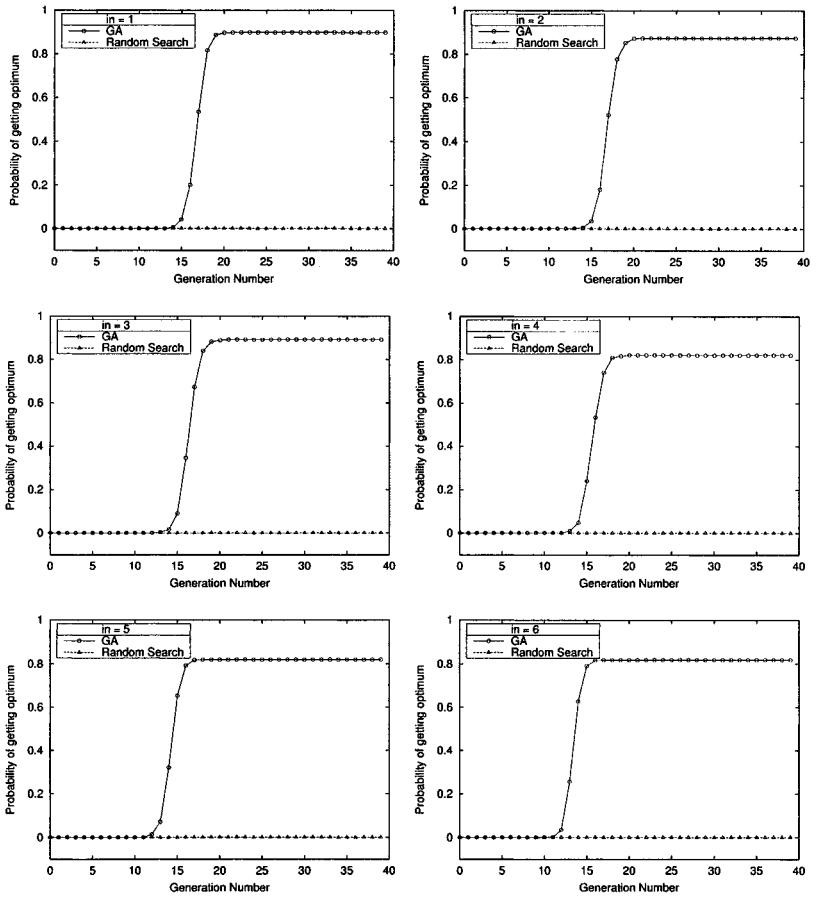


Fig. 9. The probability of getting at least one optimal individual in the population at a given generation is measured over 1000 runs for trap functions of $l = 80$, $k = 8$

For randomly initialized GAs, we can assume $p_0 = 0.5$. The time to convergence ($p_t = 1.0$) is therefore

$$t = \frac{\pi\sqrt{l}}{2I} \,. \tag{15}$$

In this derivation [17], I has been assumed to be independent of p_t. The selection intensity for tournament selection is given by the maximal order statistic $\mu_{s:s}$ [2, 16]. Order statistic $\mu_{i:j}$ denotes the expected value of the ith biggest sample out of a sample of size j drawn from a unit normal distribution $N(0, 1)$. For binary tournament selection $I = \mu_{2:2} = 0.5423$.

Until now, we have been working at the allele level. The next step is to extend the theory from alleles to BBs. Miller [14] proposed two ways to

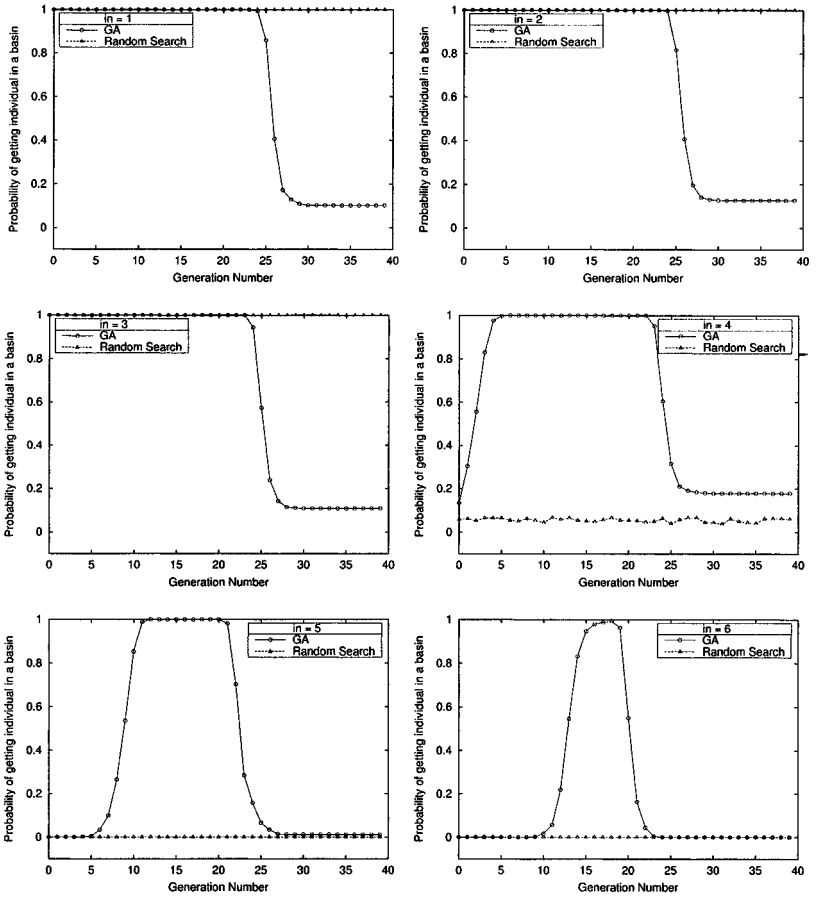


Fig. 10. The probability of getting at least one individual in a basin at a given generation is measured over 1000 runs for trap functions of $l = 80$, $k = 8$

go about this task. Either, we can adapt the theory to the BB level, get fitness mean and variance in terms of the proportion of the optimal BB, and proceed. Or, we can stick to allele-wise modeling and plug in an appropriate termination criterion. For the OneMax BB domain, the BB distribution is binomially distributed, and the proportion of correct BBs increases as the distribution converges to the optimal BB. We use the allele-wise model to estimate the convergence time for the OneMax BB domain. The proportion of correct BBs is denoted by $p_{bb,t}$. The main idea is to use $p_{bb,t} = p_t^k$ so that $p_t = \sqrt[k]{p_{bb,t}}$. Hence, if the desired quality is $p_{bb,t}$, we can use $p_t = \sqrt[k]{p_{bb,t}}$. In other words, the time to convergence to proportion p_t at the allele level is equal to the time taken to converge to $p_{bb,t}$ at the BB level.

For many functions, such as MaxTrap, it is difficult to express fitness mean and variance in terms of p_t. In such cases, we can use the results from the OneMax to approximate convergence time if the BB distribution is roughly normal, and the proportion of correct BBs increases as the BB distribution converges to the optimal BB. The procedure followed is similar to the OneMax BB domain except that domain-specific BB variance values are used.

5.3 Estimation of Theory Parameters for $G = GA$

First, we consider the notion of a basin of attraction for unitation functions. Once we have a point to the right of the inflection point, there is no way we can reach any optimum other than the global one with local search. Thus, all points to the right of the point of inflection, in, are in the basin of attraction of the global optimum. Ideally, our aim is to use a global searcher which maximizes the chance of landing in the basin of attraction of the global optimum and then use a local searcher from that point.

Knowing how to predict the probability of "1" at a locus (p_t) for the OneMax BB domain, we can calculate the proportion of correct BBs in the population which are in the basin of attraction as follows: first, at the BB level, the probability of getting a right BB is given by

$$p_g = {p_t}^k \,. \tag{16}$$

Figure 11 shows the proportion of correct BBs at each generation. These results, averaged over 100 runs for $k = 8$ and $l = 80$, show good agreement with the above theoretical prediction.

Then, considering the location of the BB, the probability of getting a BB in the basin of attraction (inclusive of the optimum) is given by

$$p_{bb} = \sum_{j=in+1}^{k} \binom{k}{j} {p_t}^j (1-p_t)^{k-j} \,. \tag{17}$$

Figure 12 compares the theoretical and experimental results for the proportion of BBs which are in the basin of attraction. We expect the proportion of BBs within the basin to rise initially and then drop off as more and more BBs converge to the optimum. For $in = 1-4$, the experimental results shows good agreement with the theory. Beyond that, the theory starts overestimating the proportion at each generation.

Moving on to the individual level, if we want at least q out of m BBs to be correct, we can write the probability of an individual being optimal as

$$P_G = \sum_{i=q}^{m} \binom{m}{i} {p_g}^i (1-p_g)^{m-i} \,. \tag{18}$$

If we use the same criteria (q out of m should be in the basin of attraction), the probability of getting an individual in the basin of attraction is therefore

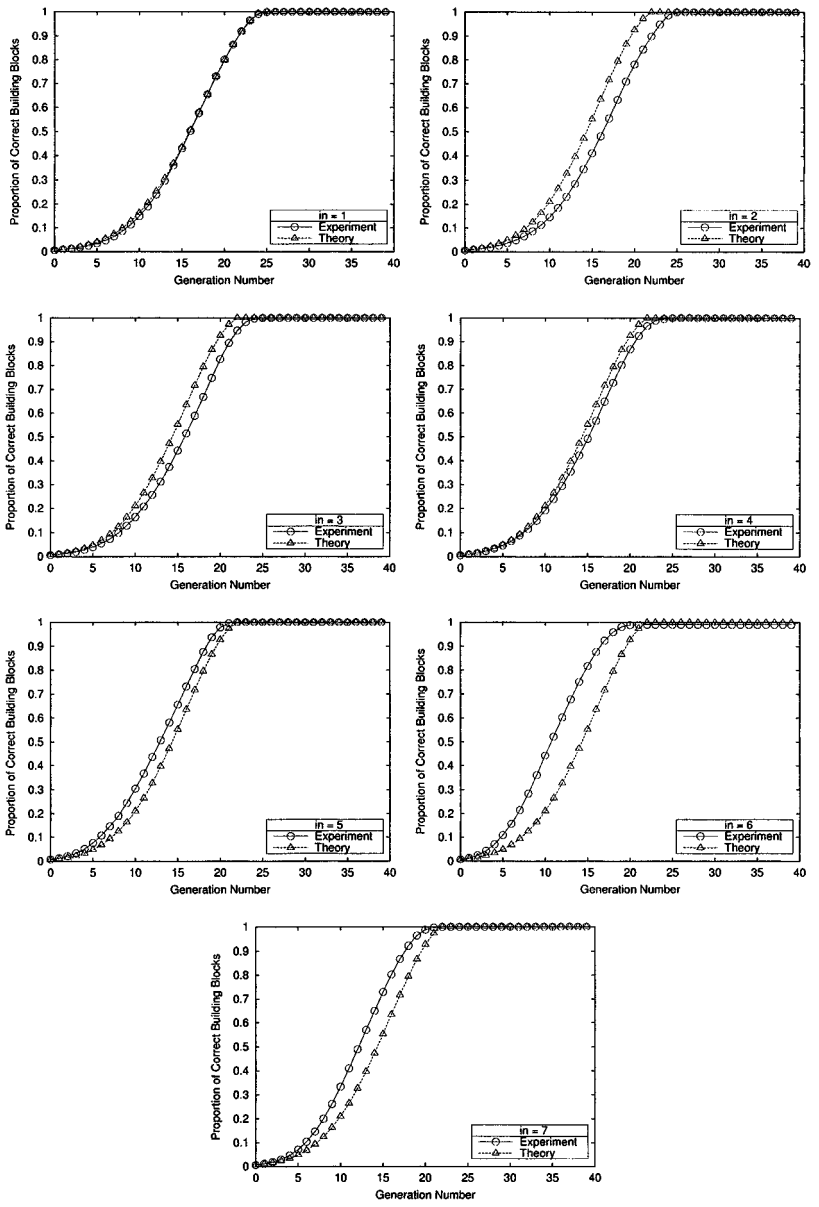


Fig. 11. The theoretical and experimental curves for the proportion of correct BBs are shown for $k = 8$, $l = 80$ for a range of trap functions with $in = 1 - 7$. The experiments traced the proportion of a correct BBs at a fixed location in the string

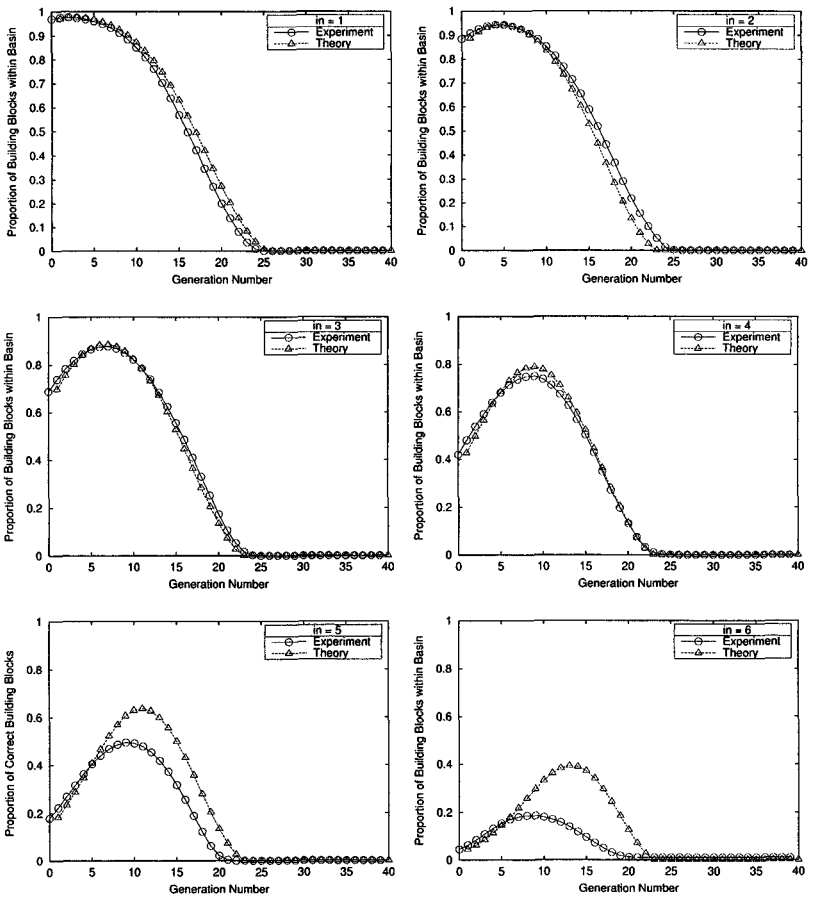


Fig. 12. The theoretical and experimental curves for the proportion of BBs within a basin for a single BB are shown for $k = 8$, $l = 80$ for a range of trap functions with $in = 1 - 6$. The experiments traced the proportion of BBs within a basin at a fixed location. Since there is no "basin" for $in = 7$, it is not shown in the figure

$$P_{Basin} = \sum_{i=q}^{m} \binom{m}{i} {p_{bb}}^{i} (1 - p_{bb})^{m-i} - P_G \ . \tag{19}$$

This includes the case where some (but not all) of the q BBs may be correct with the rest being in the basin of attraction.

Now, we introduce a new parameter, P_{LS}, the fraction of the population on which local search is performed at each generation. These individuals are selected randomly from the population. There are three possibilities: (a) the solution is in the target region; (b) it is in a basin of attraction; (c) it is in the dead zone. The probability of a solution being in each of these zones is

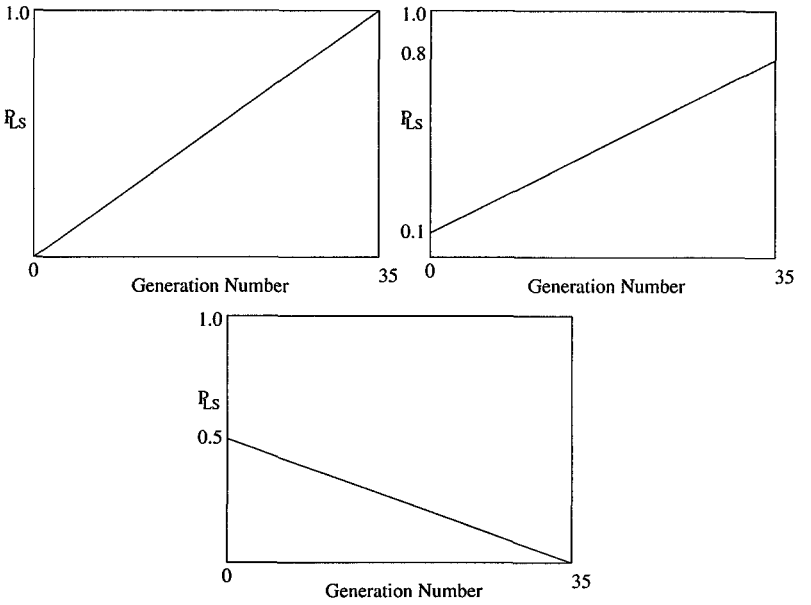


Fig. 13. Three different schedules based on variation of P_{LS}

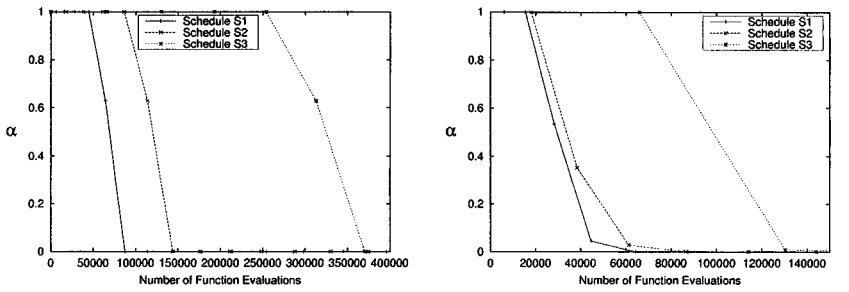
already known. The hybrid can fail only when we do not have a solution of desired quality, and none of the nP_{LS} individuals chosen for local search is in a basin of attraction. Thus, we have the probabilistic error as

$$\alpha = \sum_{i=0}^{n-nP_{LS}} \binom{n}{i} P_{Basin}^{i}(1 - P_G - P_{Basin})^{n-i} . \tag{20}$$

Equation (20) allows us to choose among different hybrid schedules.

5.4 Experiments

The theory developed in previous sections for adopting an appropriate schedule to apply global and local searchers over time in a GEA hybrid was verified through experiments. A GA with $l = 80$, $k = 8$ with Baldwinian local search (next ascent) was used. The coupling between GA and local search has lesser effect on the GA when Baldwinian local search is employed. The duration of local search at each generation was kept constant at 80 evaluations. This was to ensure that if we are in a basin of attraction, local search always leads to the optimal solution. The fraction of population which undergoes local search, P_{LS}, varies from generation to generation. Our aim is to choose a schedule, from a host of schedules, that results in the lowest probabilistic error or the lowest failure probability.



(a) Theoretical prediction (b) Experimental results

Fig. 14. Comparison of probabilistic error for different hybrid schedules

The test functions are trap functions with different points of inflection. For these functions, doing a full local search proves too expensive, and the GA alone performs better. To circumvent this, we scale down the cost associated with local search. The idea is that if we had a local searcher that had these scaled down costs, we could apply the theory to choose a schedule. More importantly, since optimal population sizing for hybrids will decrease the required population size, this analysis can still hold. Three schedules are considered: $S1$, $S2$, and $S3$. These schedules differ in how the fraction of the population that undergoes local search at each generation (which is P_{LS}) is varied over the evolutionary process. In $S1$ and $S2$, P_{LS} is increased gradually. In $S3$, P_{LS} is the maximum at the start of the run and gradually decreases over the run. Figure 13 shows the three schedules.

The theory suggests that $S1$ should yield the lowest error, and the experiments corroborate this finding. Figure 14 compares the probabilistic error estimates of the theory (averaged over 1000 runs) to the experimental results. The population size was 8000. The results are for the trap function with $in = 4$, $d = 0.55$, $k = 8$, and $l = 80$. Since we are doing a complete Baldwinian local search (NAHC without replacement) on selected individuals every generation, we expend 80 function evaluations on each selected individual. The theory correctly predicts the relative performance of different schedules. By providing us qualitative insights into the relative performance of different schedules, it enables us to choose the best schedule. Ideally, we would like to generate an optimal schedule that results in the lowest probabilistic error among all possible schedules. This theory is a positive step in this direction.

The theory is unable to predict strictly accurate behavior. The difference between theoretical and empirical results can be explained as follows. First, the theory assumes that the behavior for these functions is similar to MaxTrap. Second, the theory ignores the improvement in probability of hitting the basin as a result of local search. It causes the theory to overestimate the failure

probability at each generation. Nonetheless, the theory still gives us a fair idea about the relative performance of different schedules and can be used for guidance when choosing a schedule for hybridization.

6 Summary and Conclusions

This chapter started with presenting state-of-the-art GEA hybrids. A taxonomy for hybrids and outstanding issues pertinent to hybrid design was then described. A principled approach to design of GEA hybrids, based on past work, was introduced and verified by considering three cases—uniform time-to-criterion (TTC) across basins, different TTC across basins, and different TTC within basins—with random search as the global searcher and a quasi-Newton method as the local searcher. Following this, the utility of local search for a class of trap functions was illustrated. A decrease in population sizing requirements for GAs was demonstrated after combining it with a local searcher. Also, the speedup from local search turns out to be inversely proportional to the square root of the problem size.

The hybrid theory was then extended to incorporate GAs as the global searcher. The parameters required by the theory were estimated with the run duration theory in the literature. The probabilistic error was derived as a function of the fraction of the population undergoing local search at a given generation. Although the theory does not generate optimal schedules, it gives a fair estimate of the relative performance of different schedules and enables the user of GEA hybrids to decide among different schedules.

This study presented a high level framework for designing GEA hybrids. A number of extensions are possible:

- **Test on a rigorous test suite**: Although the theory has been extended to handle GAs as global searchers, it needs to be tested more extensively. Test problems involving sources of problem difficulty other than deception are needed to demonstrate the applicability of the theory to a broader range of problems.
- **Methods for offline and online estimation of theory parameters**: The theory parameters: probabilities of hitting the global zone and the basins of attraction and the TTC values for different basins may not be directly available in many cases. We need efficient techniques (offline or online) for estimating these parameters.
- **More accurate models**: The current theory does not account for the improvement in probabilities for global search due to local search. This causes the theory to overestimate the time required to reach the desired solution quality. A more exact model that incorporates this effect to predict the proportion of individuals within the basins of attraction in a given generation is needed.
- **Methods to generate optimal schedules**: The utility of the theory has been demonstrated. However, as problems become more complex, efficient

methods to generate schedules for local and global search will be required to achieve an optimal division of labor.

- **Combinations of more than two methods**: Currently, the theory is designed to handle combinations of two methods, but many practical hybrids adopt more than two methods. Hence, suitable extensions to handle hybrids that use several search procedures are needed.
- **Population sizing for hybrids**: This study showed decrease in population size requirements for GEA hybrids. This was attributed to a decrease in variance thanks to local search. A systematic exploration of the population sizing requirements is required.

These extensions can provide answers to some important questions: When is the theory expected to be good? What are the dimensions of hybrid difficulty? How should one estimate the theory parameters in the absence of complete knowledge of the fitness landscape? As practitioners seek answers to these questions, researchers will soon direct their efforts to explore these avenues.

GEA hybrids have come a long way, but until recently, the focus had been on empirical approaches toward hybrid design. Practitioners have felt the need for of a systems-level framework for designing and understanding hybrids. This study has presented a more principled approach for hybrid design with a systems-level framework. Compared to the ad hoc nature of previous approaches, it helps the practitioner in making key decisions during design and promotes a greater understanding of the underlying issues. This study focuses on achieving an optimal division of labor between the global and local searchers in order to get the best performance out of the hybrid as a whole.

The hybrid theory enables practitioners to choose among different schedules of hybrids based on the chances of success. Practitioners should also take into account the lower population size requirements for GEA hybrids. The speedups from local search should be expected to be inversely proportional to the square root of the problem size. For GEA researchers, this hybrid theory holds promise of a systems-level framework for hybrid design. This work, through a more principled approach, provides a good start to an otherwise daunting problem of GEA hybrid design.

Acknowledgments

The work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant F49620-00-0163. Research funding for this work was also provided by a grant from the National Science Foundation under grant DMI-9908252. The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or

endorsements, either expressed or implied, of the Air Force Office of Scientific Research, the National Science Foundation, or the U.S. Government.

References

1. T. Bäck. Selective pressure in evolutionary algorithms. *Proceedings of the 1994 IEEE International Conference on Evolutionary Computation*, 1994.
2. T. Bäck. Generalized convergence models for tournament and (μ, λ) selection. *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA-95)*, pages 2–8, 1995.
3. R. K. Belew and M. Mitchell, editors. *Adaptive Individuals in Evolving Populations.* Addison-Wesley, 1996.
4. M. G. Bulmer. *The mathematical theory of quantitative genetics.* Oxford University Press, Oxford, 1985.
5. L. Davis, editor. *The Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York, 1991.
6. K. Deb and D. E. Goldberg. Analyzing deception in trap functions. *Foundations of Genetic Algorithms 2*, pages 93–108, 1993.
7. D. E. Goldberg. Real-coded genetic algorithms, virtual alphabets, and blocking. *Complex Systems*, 5(2):139–167, 1991.
8. D. E. Goldberg. *Design of innovation: Lessons from and for competent genetic algorithms.* Kluwer Academic Publishers, 2002.
9. D. E. Goldberg, K. Deb, and J. Clarke. Genetic algorithms, noise and sizing of populations. *Complex Systems*, 6:333–362, 1992.
10. D. E. Goldberg and S. Voessner. Optimizing global-local search hybrids. *Proceedings of Genetic and Evolutionary Computation Conference 99 (GECCO-99)*, pages 220–228, 1999.
11. G. Harik, E. Cantú-Paz, D. E. Goldberg, and B. L. Miller. The gambler's ruin problem, genetic algorithms, and the sizing of populations. *Evolutionary Computation*, 7(3):231–253, 1999.
12. W. E. Hart. *Adaptive Global optimization with Local Search.* PhD thesis, University of California, San Diego, CA, 1994.
13. G. E. Hinton and S. J. Nowlan. How learning can guide evolution. *Complex Systems*, 1(1):495–502, 1987.
14. B. Miller. *Noise, sampling, and efficient genetic algorithms.* PhD thesis, University of Illinois, Urbana-Champaign, Urbana, IL, 1997.
15. B. Miller and D. E. Goldberg. Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, 9(3):193–212, 1995.
16. B. Miller and D. E. Goldberg. Genetic algorithms, selection schemes and the varying effects of noise. *Evolutionary Computation*, 4(2):113–131, 1996.
17. H. Mühlenbein and D. Schlierkamp-Voosen. Predictive models for the breeder genetic algorithm: I. Continuous parameter optimization. *Evolutionary Computation*, 1(1):25–49, 1993.
18. M. Pelikan and D. E. Goldberg. Hierarchical problem solving and the bayesian optimization algorithm. *Proceedings of Genetic and Evolutionary Computation Conference 2000 (GECCO-2000)*, pages 267–274, 2000.
19. M. Pelikan and D. E. Goldberg. Escaping hierarchical traps with competent genetic algorithms. *Proceedings of Genetic and Evolutionary Computation Conference 2001 (GECCO-2001)*, pages 511–518, 2001.

20. W. H. Press, S. A. Teukolsky, A. Saul, W. T. Vetterling, and B.P. Flannery. *Numerical Recipes in C.* Cambridge University Press, Cambridge , MA, 2nd edition, 1992.
21. A. Sinha and D. E. Goldberg. Verification and extension of global-local hybrid theory. *Proceedings of Genetic and Evolutionary Computation Conference 2001 (GECCO-2001)*, 2001.
22. A. Sinha and D. E. Goldberg. A survey of hybrid genetic and evolutionary algorithms. Technical Report 2003004, Illinois Genetic Algorithms Laboratory, 2003.
23. D. Thierens and D. E. Goldberg. Mixing in genetic algorithms. *Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA-93)*, pages 38–45, 1993.
24. D. Thierens and D. E. Goldberg. Convergence models of genetic algorithm selection schemes. *Parallel Problem Solving from Nature-94*, pages 119–129, 1994.
25. A. Törn and A. Žilinskas. *Global Optimization.* Springer, Berlin, 1989.
26. D. Whitley, V. S. Gordon, and K. Mathias. Lamarckian evolution, the Baldwin effect and function optimization. *Parallel Problem Solving from Nature-94*, pages 6–15, 1994.
27. J. Yen, J. Liao, D. Randolph, and B. Lee. A hybrid approach to modeling metabolic systems using genetic algorithms and simplex method. *Proceedings of the 11th IEEE Conference on Artificial Intelligence for Applications (CAIA95)*, pages 277–283, 1995.

The Design of Memetic Algorithms for Scheduling and Timetabling Problems

E.K. Burke and J.D. Landa Silva

Automated Scheduling, Optimisation and Planning Research Group
School of Computer Science and IT, The University of Nottingham, UK
(ekb|jds)@cs.nott.ac.uk

Summary. There are several characteristics that make scheduling and timetabling problems particularly difficult to solve: they have huge search spaces, they are often highly constrained, they require sophisticated solution representation schemes, and they usually require very time-consuming fitness evaluation routines. There is a considerable number of memetic algorithms that have been proposed in the literature to solve scheduling and timetabling problems. In this chapter, we concentrate on identifying and discussing those strategies that appear to be particularly useful when designing memetic algorithms for this type of problems. For example, the many different ways in which knowledge of the problem domain can be incorporated into memetic algorithms is very helpful to design effective strategies to deal with infeasibility of solutions. Memetic algorithms employ local search, which serves as an effective intensification mechanism that is very useful when using sophisticated representation schemes and time-consuming fitness evaluation functions. These algorithms also incorporate a population, which gives them an effective explorative ability to sample huge search spaces. Another important aspect that has been investigated when designing memetic algorithms for scheduling and timetabling problems, is how to establish the right balance between the work performed by the genetic search and the work performed by the local search. Recently, researchers have put considerable attention in the design of self-adaptive memetic algorithms. That is, to incorporate *memes* that adapt themselves according to the problem domain being solved and also to the particular conditions of the search process. This chapter also discusses some recent ideas proposed by researchers that might be useful when designing self-adaptive memetic algorithms. Finally, we give a summary of the issues discussed throughout the chapter and propose some future research directions in the design of memetic algorithms for scheduling and timetabling problems.

1 Introduction

It is possible to think of a memetic algorithm as an evolutionary algorithm that incorporates knowledge about the problem domain being solved (see [29, 33]). This knowledge can be in the form of specialised operators, heuristics and

other *helpers* that contribute towards a self-improvement ability in the individuals of the population. Most memetic algorithms described in the literature are a combination of genetic algorithms with local search heuristics and these approaches are also known as genetic local search, hybrid genetic algorithms, hybrid evolutionary algorithms and other names (e.g. [13, 22, 25, 38, 44]). This type of hybrid approach has been applied to a vast number of optimisation problems with considerable success (see [34] for a list of example references).

In this paper we concentrate on the application of memetic algorithms to scheduling and timetabling problems such as machine scheduling, educational timetabling and personnel rostering [47]. One goal here is to identify and discuss those strategies that appear to be applied more frequently by researchers and practitioners, when designing memetic algorithms for the type of problems mentioned above. Another goal is to discuss some recent ideas proposed by researchers in this area that might help to design more robust memetic algorithms for scheduling and timetabling problems. We do not attempt to provide an exhaustive survey of memetic approaches to such problems. First, in Sect. 2 we briefly describe the paradigm behind memetic search followed by a short discussion of scheduling and timetabling problems in Sect. 3. Then, in Sect. 4 we concentrate on those strategies that are frequently employed when implementing memetic algorithms for scheduling and timetabling (and perhaps related problems). Scheduling and timetabling problems are usually highly constrained and one of the difficulties when solving these problems is how to deal with infeasibility. Section 4.1 discusses the strategies employed for this purpose within the context of memetic algorithms. For most scheduling and timetabling problems, a complete evaluation of the solution fitness is computationally costly. Therefore, the use of approximate evaluation routines can help to implement more efficient memetic algorithms. This topic is discussed in Sect. 4.2. Selecting an appropriate solution encoding is essential when designing memetic algorithms because both genetic and local search should operate on this encoding in an efficient way. This is discussed in Sect. 4.1. The importance of obtaining knowledge of the fitness landscape to aid the design of better memetic algorithms is discussed in Sect. 4.4. Another very important aspect when designing memetic algorithms is to establish a good balance between the genetic search and the local search parts of the algorithm and this is the subject of Sect. 4.5. Most implementations of memetic algorithms incorporate *memes* designed a priori and these remain unchanged during the search. However, one of the original ideas that motivated the conception of memetic algorithms is the *evolution of memes* and this is discussed in Sect. 4.6. Section 5 describes some local search strategies that have been proposed recently by researchers and that might also be useful if they are incorporated within memetic algorithms. Finally, this paper presents some final remarks and suggests some future research directions in Sect. 6.

2 Memetic Algorithms

It is generally believed that memetic algorithms are successful because they combine the explorative search ability of recombinative evolutionary algorithms and the exploitive search ability of local search methods. An analogy is that the evolutionary part of a memetic algorithm attempts to simulate the genetic evolution of individuals through generations, while the local search part attempts to simulate the individual learning within a lifetime. The majority of memetic algorithms proposed in the literature are a result of incorporating some form of local search to a genetic algorithm. This is illustrated in Fig. 1.

This local search can be for example, constructive heuristics, repair methods, specialised self-improvement operators, etc. The local search phase can be applied before, after or in between the genetic operations. However, as discussed in [29], the interaction between the *memes* and the *genes* can be even more sophisticated than that and most implementations of memetic algorithms fail to reflect the complex interactions of the memetic paradigm. Krasnogor [29] argues that in a truly memetic system:

1. *Memes* also evolve representing the way in which "individuals learn, adopt or imitate certain memes or modify other memes" and,
2. the distribution of *memes* changes dynamically within the population representing the effects of "teaching, preaching, etc." within the population of individuals.

Krasnogor also proposed a formulation of memetic algorithms that attempts to better represent the memetic paradigm including adaptive and self-adaptive *memes*, for more details see [29]. For a more detailed discussion of memetic algorithms see [29, 33, 34] and the references therein.

3 Scheduling and Timetabling Problems

Broadly speaking, the task in scheduling and timetabling problems is to accommodate a set of entities (for example, events, activities, people, vehicles, etc.) into a pattern of time-space so that the available resources are utilised in the best possible way and the existing constraints are satisfied. This paper is mainly concerned with three types of scheduling problems: machine scheduling, educational timetabling and personnel scheduling (also called rostering). However, we will also allude to papers which consider other scheduling problems. A brief description of machine scheduling, educational timetabling and personnel scheduling follows. For more details please see the given references. In machine scheduling, the problem is to schedule a set of jobs for processing on one or more machines [37]. Educational timetabling refers to the allocation

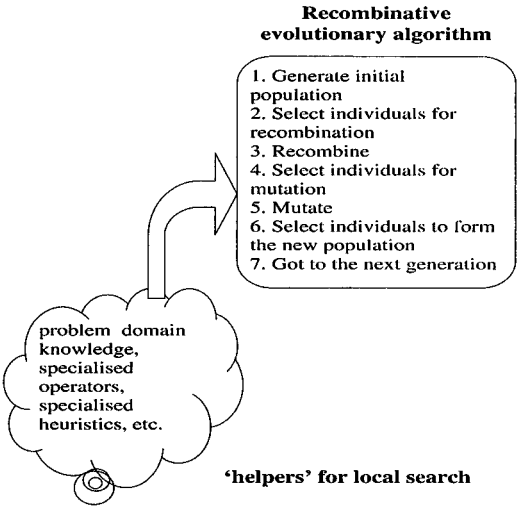


Fig. 1. Most memetic algorithms are a combination of genetic algorithms and local search strategies.

of events (teaching sessions, exams, lab sessions, etc.) to time slots and possibly to rooms [42]. In personnel scheduling, employees must be accommodated into shift patterns [19].

Scheduling and timetabling problems arise in many situations and hence, there is a need for developing effective and efficient automated solution methods. But as with many other combinatorial optimisation problems, scheduling and timetabling are difficult problems to tackle with computer algorithms. The following characteristics are those that make scheduling and timetabling problems very difficult:

Huge search space. The combinatorial nature of scheduling and timetabling problems implies that the size of the search space increases dramatically with the size of the problem making it practically impossible to explore all solutions but for very small problems.

Highly constrained. It is commonly the case that a considerable number of constraints exist in these problems. Constraints limit the possible ways in which a schedule can be constructed. Some constraints must be satisfied for the solutions to be feasible (hard constraints) while other constraints are desirable but not absolutely essential (soft constraints). Examples of constraints are: job A must be processed before job C but after job D (machine scheduling); lecturers cannot teach more than two consecutive sessions (educational timetabling); a minimum number of nurses must be scheduled during busy times (personnel scheduling).

Difficult to represent. It is often difficult to find a representation with associated data structures that capture all details of the problem including the complete set of constraints. Most of the times the problem is simplified, otherwise very elaborate representations are required to model it (see the sections below).

Time-consuming fitness evaluation. Computing the fitness of solutions in scheduling and timetabling problems is usually time consuming. The main reason for this is the existence of many constraints. When a solution is modified even slightly during the search, a number of constraints might be affected and therefore, a complete computation of the whole solution is required.

There is a significant school of thought which says that for most scheduling and timetabling problems, the improvement of solutions can be achieved more effectively with local search heuristics than with recombinative operators. These local search heuristics are usually tailored to the application problem by incorporating knowledge of the problem domain in order to deal with the constraints in a more effective way. It is also generally believed that, because memetic algorithms operate on a population of solutions, they are less dependant on the quality of the initial solutions than local search methods which operate on a single solution. Then, the appeal of applying memetic algorithms for scheduling and timetabling problems is that the powerful improving mechanism of local search is maintained and at the same time, enriched by the addition of a population of individuals.

4 Designing Memetic Algorithms

There is a considerable number of applications of memetic algorithms to scheduling problems reported in the literature including: machine scheduling (e.g. [22, 24]), educational timetabling (e.g. [2, 5, 6, 16, 36]), personnel scheduling (e.g. [1, 9, 26]), maintenance scheduling (e.g. [12, 13]) among many others.

As mentioned above, the addition of local search *helpers* into genetic algorithms is the most common approach reported in the literature. The variety of helpers that have been proposed range from the use of tailored chromosome representations and operators (e.g. [17, 26]) and simple repairing methods based on constraint-based reasoning (e.g. [16]) to very sophisticated combinations of algorithm components in which a memetic algorithm is embedded into a genetic algorithm (e.g. [3]). In this Sect., we discuss the strategies that have been used more frequently by researchers and practitioners when designing memetic algorithms for scheduling and timetabling problems.

4.1 Dealing with Infeasibility

A major issue of concern in scheduling and timetabling is how to deal with the infeasibility of solutions. Due to the large number of hard constraints that typically exist in these problems, generating feasible solutions and keeping them feasible during the search is a difficult task. In general, the first decision that has to be made is whether to consider or not infeasible solutions as part of the searching process. In both cases, the design of adequate solution representations and search operators is an essential ingredient for an effective and efficient operation of the search algorithm. The incorporation of knowledge of the problem domain in the form of choosing the representation and operators according to the existing constraints in the problem, can be considered to be a memetic approach in itself if we accept the broad description of memetic algorithms [33].

Elaborate Encodings and Operators

One way to deal with infeasible solutions is to forbid them completely. That is, only feasible solutions are generated in the initialisation phase and then, the genetic and local search operators are restricted to work only in the feasible region. This approach was used by Burke et al. in the nurse scheduling problem [9] and by Erben and Keppler in the course timetabling problem [18]. Very elaborate solution encodings can also be used to avoid the generation of infeasible solutions altogether. For example, Erben applied grouping genetic algorithms to examination timetabling problems in [17]. In a grouping genetic algorithm, the chromosome representation is made of groups of genes (a group can be, for example, the events scheduled in the same time slot). Then, while in a direct representation of a timetable each gene represents one event, in a grouping representation each gene represents a group of events. In this way, it is easier to design genetic operators that recombine timetables without destroying the feasibility of solutions (i.e. in this case keeping events in the same time slot). Similarly, Kawanaka et al. employed a very elaborate encoding that guaranteed the feasibility of solutions for the nurse scheduling problem [26]. Also, Aickelin and Dowsland [1] implemented two levels of crossover operators for the nurse scheduling problem, one operating at the individual nurse schedule level and another operating at a higher level for the complete schedule. In the first operator, the genes are the time slots while in the second operator the genes are the whole individual schedules of the nurses.

Elaborate encodings and operators help to maintain good sub-solutions (parts of schedules) and aim to generate better complete schedules by mixing good building blocks. However, this can generate more design issues which need to be solved. For example, how do we measure fitness in each type of operator? This might then lead on to investigating various selection schemes within the same algorithm while it is necessary to somehow preserve the good parts of solutions that are already created in order to obtain sensible results,

we should keep in mind that restricting the search to only the feasible regions of the solution space could considerably limit the explorative ability of the memetic algorithm.

Repair Methods and Penalty Schemes

Another approach to deal with infeasibility is to penalise and/or repair infeasible solutions. That is, if the solution encoding and associated search operators permit the generation of infeasible solutions, then repairing heuristics that recover the feasibility of solutions can be implemented. The reparing method should be easy to implement so that no excessive overhead is added to the search process. Also, if the reparing method is too elaborate, it may happen that most of the changes made by the search operators to obtain the new solution from the previous one are reversed by the repairing method resulting in a very inefficient process. For example, repairing heuristics were used by Colorni et al. in the application of genetic and memetic algorithms to high school timetabling problems [14].

An alternative strategy when infeasible solutions are allowed, it is to heavily penalise them in order to discourage their survival. The selection of the penalties must be carefully made. The recommendation is that the penalties should discourage the inclusion of infeasible solutions but without completely eliminating them because infeasible solutions may be required for the algorithm to have a better explorative ability. In our experience, if local search is used, relatively low penalties for infeasibility should be set in many cases because if these penalties are too high, then the local search attempts to recover feasibility first and this could increase substantially the violation of soft constraints [7, 9]. Penalties can be fixed or they can be adapted during the search. Aickelin and Dowsland implemented an adaptive scheme where the infeasibility penalty depends on the number of violated hard constraints [1]: if $q > 0$ then $g_{demand} = \alpha \times q$ where q is the number of violated hard constraints, α is a severity parameter and g_{demand} is the infeasibility penalty weight used in the fitness function.

Multi-Phased Strategies

Some researchers have employed multi-phased approaches to deal with hard constraints and hence, infeasibility of solutions by dividing the solution process in multiple stages. In the first phase the goal is to generate semi-complete feasible solutions. For example, in the nurse scheduling problem, the emphasis can be on the generation of a semi-complete schedule in which it is guaranteed that there are enough nurses available to meet the requirements in each shift, without assigning actual working time slots to each nurse [1]. Then, in the subsequent phases the schedule is incrementally completed by assigning working time slots to individual nurses and forbidding the violation of hard constraints.

Burke and Newall also used a multi-phased approach in their memetic algorithm for the examination timetabling problem [5] (this is an improved version of their previous algorithm presented in [6]). Their memetic approach, however, did not use recombination operators, only mutation operators followed by a hill-climbing algorithm. They constructed a partial timetable and then iteratively applied a memetic algorithm with the aim of scheduling a subset of events in each iteration, i.e. the memetic approach is applied to a subproblem in each iteration. This can be represented in diagrammatic form as shown in Fig. 2.

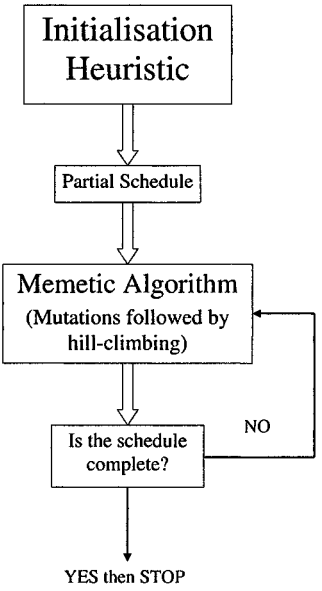


Fig. 2. The multi-phase memetic algorithm implemented by Burke and Newall [5] acts as a peckish (not too greedy) constructive heuristic.

4.2 Approximate Fitness Evaluation

It has been shown that using approximate evaluation (also known as delta evaluation) to measure the quality of solutions helps to improve the efficiency of the search algorithm in problems for which the fitness evaluation function is very time consuming (e.g. [5, 6, 20, 31, 46]). With approximate evaluation, instead of a complete and accurate evaluation of each newly generated solution, only the difference between the previous solution and the new one is computed. Approximate evaluation can be applied as follows to reduce the computation time spent by the memetic algorithm. Two fitness evaluation routines are implemented. One routine completely evaluates the fitness of the new solution (*complete evaluator*). The second one only makes an estimation

of the new fitness based on the previous solution and the changes made (*approximate evaluator*). Suppose that a population of new solutions is created in each generation. Instead of computing the fitness of these solutions with the *complete evaluator*, their fitness is measured with the *approximate evaluator*. Then, only a fraction of the population of new solutions is re-assessed with the *complete evaluator*. These solutions can be for example, the best ones, those that have a minimum quality level, or only those that represent an improvement with respect to the previous solutions. This simple strategy can save a considerable amount of computation time because it is very likely that few solutions would need to be re-evaluated with the *complete evaluator* in the first stages of the evolutionary process. As the search progresses and the overall quality of the population improves, more solutions will have high fitness and perhaps, only the *accurate evaluator* will be used. For more details on how this kind of strategy was implemented for the warehouse scheduling problem see [46], for the examination timetabling problem see [5, 6] and for the space allocation problem see [31].

Of course, for this strategy to be effective it is required that the structure of the combinatorial optimisation problem is such that the quality of the new solutions can be updated by evaluating only the changes made to the previous solution. Approximate evaluation can be applied in any of the stages of a memetic algorithm, i.e. during the local search phase or during the genetic search phase. As mentioned above, a common characteristic of many real world scheduling and timetabling problems is that a considerable number of constraints exist in these situations. Hence, it is very likely that many of the constraints in a particular problem instance are affected by even simple changes to the previous solution. This means that the degree of inaccuracy when using the *approximate evaluator* can be higher than in less constrained combinatorial problems and the implementation of the approximate routine must be carried out carefully.

4.3 Encodings Based on Linked Lists

Another aspect that must be considered when using approximate evaluation is that the solution encoding selected and associated data structures should allow an efficient implementation of the approximate evaluation routine (i.e. much more efficiently that the complete evaluation). In this respect, for scheduling and timetabling problems (and combinatorial problems in general) solution encodings and data structures based on linked lists have been shown to be very helpful for implementing moves and fitness evaluation routines more efficiently (e.g. [5, 31, 39]). This type of representation is advantageous in combinatorial optimisation for several reasons:

- Linked lists can dynamically shrink/grow easily by deleting/adding elements,
- they can also be modified efficiently by only changing pointers,

- linked lists can be used to represent virtually any structure (array, matrix, set, etc.), and
- local search operators such as move, swap, invert, add, delete, change, etc. can be performed directly and very easily with linked lists.

An example of this type of encoding is shown in Fig. 3 for the space allocation problem. In this problem, a set of entities (staff, lecture rooms, labs, etc.) must be allocated to a set of available rooms in such a way that all hard constraints are satisfied, the space misuse is minimised, and the violation of soft constraints is minimised (see [31] for more details). In the encoding of Fig. 3, the lists *Entities*, *Rooms* and *Constraints* hold details of the problem being solved, then these lists remain unchanged throughout the search. That is, these lists hold, for example, the required space for each entity, the capacity of rooms, the type and nominal penalty of constraints, etc. The lists *EntityGene*, *RoomGene* and *ConstraintGene*, hold details of a solution or allocation, i.e. fitness statistics, pointers to the problem data, and pointer that define the structure of the solution. In the solution represented in Fig. 3, entity E1 is allocated to room R5, room R2 is empty, entity E3 is not allocated, constraints C3 and C4 apply to entity E5, etc. With this data structure, it is easy to implement local search moves by only changing the appropriate pointers. Similarly, fitness evaluation routines can be performed efficiently because it is easy to identify which constraints have been affected (by walking along the corresponding linked lists) after a change to the solution structure.

4.4 The Fitness Landscape

Another aspect that makes scheduling and timetabling problems difficult to tackle is that, as in most combinatorial optimisation problems, the shape of the fitness landscape usually depends on each particular problem instance. Moreover, the penalties associated to the various soft constraints in the problem can have an effect on the distribution of local optima. For example, consider two soft constraints SC_1 and SC_2 with associated penalties Γ_1 and Γ_2 respectively. Let SC_1 be 'more important' than SC_2 but 'less difficult' to satisfy than SC_2. That is, more solutions are expected to violate the 'less important' SC_2 than SC_1. If $\Gamma_1 >> \Gamma_2$ it is likely that the search will be biased towards finding many very attractive solutions that satisfy SC_1 (the more important one), but improvements in SC_2 are likely to be overlooked. One the other hand, if $\Gamma_1 << \Gamma_2$ improvements in SC_2 will be preferred over improvements in SC_1 and it is likely that the number of attractive solutions satisfying SC_1 will be less than in the previous case. Further, for the same problem instance, different search operators (local search, crossover, mutation, etc.) explore the solution landscape in very different ways. An ideal situation would be that some knowledge of the fitness landscape for a given problem could be available before the search, but this is rarely the case. Even when preliminary examinations are carried out, it turns out that the solution landscape presents very different characteristics for each problem instance [1].

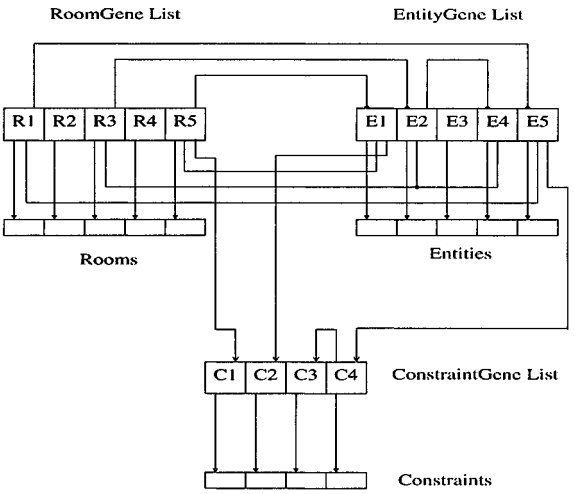


Fig. 3. Solution encoding based on linked lists used for the space allocation problem. The global lists *Entities*, *Rooms* and *Constraints* hold data corresponding to the problem instance being solved. The linked lists of genes *EntityGene*, *RoomGene* and *ConstraintGene*, hold details of a particular allocation or solution.

In this respect, Merz and Freisleben proposed the idea of *fitness landscape analysis* as a means to obtain, a priori, some knowledge of the fitness landscape which could help to design better memetic approaches [32]. They suggest that the first step to perform *fitness landscape analysis* is to define the fitness landscape for the problem instance. For this it is necessary to assign a fitness value to each solution in the search space, i.e. define the fitness function. Then, the spatial structure of the landscape should be defined by defining a metric that measures the distance (in the genotypical space) between two solutions. As Merz and Freisleben note, a simple metric to define the distance between two solutions s and t could be the minimum number of applications of an operator ω required to obtain t from s. Then, they suggest to carry out some preliminary experiments on the problem instance and perform calculations to estimate the properties of the fitness landscape that are known to have an effect on the performance of heuristic methods (see [32] for full details):

- The difference in fitness value between neighbouring solutions.
- The number of local optima.
- An estimation of the distribution of the local optima.
- The topology of the basins of attraction of the local optima.

Knowles and Corne have also carried out some studies on the analysis of the fitness landscape in combinatorial optimisation problems [28] (their approach is discussed below).

4.5 Balance Between Genetics and Memetics

An issue that has received considerable attention when designing memetic algorithms is how to establish the right balance between the work performed by the genetic search and the work performed by the local search. Ideally, in a memetic algorithm, genetic and local search, i.e. the two broadly defined groups of operators, should be able to work together in cooperation instead of against each other [29]. This balance can be tuned from three perspectives among others:

1. The balance between the sophistication of the genetic operators and the local search operators.
2. The decision as to which solutions each group of operators is applied.
3. The balance between the computing time allocated to each type of search.

Some Recommendations

Tuning Genetic and Local Search

For example, with respect the sophistication of the operators, Burke et al. noted that recombining large parts of schedules to form a child solution was very ineffective because the local search heuristics in their memetic algorithm were not powerful enough to improve on these solutions [9]. They observed that instead, it was more beneficial to combine small parts from the parents so that more diversity could be obtained and the local search part could be performed more effectively. But if the local search part of the memetic approach is too powerful, it dominates the search as observed by Burke and Smith when a well tuned tabu search procedure was incorporated as the local search phase in a memetic algorithm for the maintenance scheduling problem [13]. One might feel tempted to use the powerful local search operators and heuristics already known for scheduling and timetabling problems, but the difficulties mentioned above can be encountered.

In the case of which solutions should be applied to each group of operators, an approach that has been used is to improve by local search only a number of the best solutions in the population (e.g. [1]). In [22] Ishibuchi et al. implemented a first version of a memetic algorithm for the flow-shop scheduling problem (named genetic local search in that paper) where they proposed not to examine the whole neighbourhood but only a fraction of it (i.e. best of k instead of best of all) and stop the search when no better neighbour is found after a small number of iterations. Later, they also proposed to apply local search to only good offspring to improve the search ability of

their genetic local search approach [23]. Although in that paper Ishibuchi et al. consider multi-objective flow-shop scheduling problems, this idea can be transferred from the multi-objective domain to the single-objective domain by noting that good solutions are not those that offer a good coverage of the front but solutions that represent a good subset of the population because of fitness and diversity. In a more recent study, Ishibuchi et al. proposed the following strategies [24]:

- To apply local search to a subset of the population selected based upon a given probability p and on the fitness of the solutions according to preset criteria.
- To apply the local search procedure not after each generation but every $T > 1$ generations.
- To carry out preliminary experiments for tuning the values of the above mentioned parameters (k, p and T).
- To carry out preliminary experiments to establish the adequate values for the crossover and mutation probabilities for tuning the genetic search (provided the parameters of k, p and T have been fixed).

With respect to the genetic operators, it has been proposed to apply them only to parent solutions that have a certain distance between them in the genotypical space (this is called a mating restriction) [35]. It has been observed that applying the local search for a limited number of iterations enables better results in the long run as reported by Burke and Smith for the maintenance scheduling problem [12].

Archives of Solutions

The use of archives of solutions (a form of elitism) as in [27] can also be employed to enhance the balance of genetic and local search. Such an archive can be used to store elite solutions from which to chose the appropriate ones before carrying out the genetic operations.

Reacting to the Shape of the Fitness Landscape

Measuring the characteristics of the fitness landscape as the search progresses can help in the design of a dynamic method to balance the genetic and the local search. This has been investigated by Knowles and Corne in the context of the multi-objective quadratic assignment problem [28]. Fitness landscape analysis techniques can help to identify the structure of a given problem, not only before the search but perhaps also dynamically during the search [32]. The study of the fitness landscape is a very promising avenue of research that can be of considerable benefit to enhance the performance of memetic algorithms, particularly on problems such as scheduling and timetabling. This is because adequate operators could then be designed and the search tuned according to the landscape.

Common Annealing Schedules

Krasnogor and Smith have investigated a form of adapting the algorithm performance according to the search, so that the emphasis can be adapted: 1) to improve the fitness of the population, or 2) to diversify the population [30]. They used a common temperature for the whole population to control the acceptance of solutions during the local search phase. The temperature is inversely proportional to the spread of fitness in the population. Therefore, as the population converges (spread of fitness is reduced) the temperature increases and more non-improving solutions are accepted in order to induce more exploration. Once the spread of fitness is recovered, the temperature falls so that only improving solutions are accepted and the search acts as a local search procedure. Burke et al. implemented a population-based annealing algorithm in which a common annealing schedule is used to control the acceptance probability of solutions generated by each of the individuals in the population [7]. In their approach, there is no recombination of solutions and the balance between exploration and intensification is managed only by the evolution of the population by self-improvement. They applied the algorithm to the space allocation problem which shares various features with the class of timetabling problems and was briefly described above (see [31]). The idea behind their common annealing schedule was to allow a certain degree of flexibility in an attempt to use the mechanism as a diversification strategy.

4.6 Towards Adaptive Memes

Most of the research related to memetic algorithms for scheduling and timetabling problems has concentrated on: 1) the design of specialised operators that enable constraints to be dealt with more efficiently, and/or 2) the comparison between the performance of different operators. For example, Alkan and Ozcan recently carried out a comparison of various mutation operators that are directed towards the satisfaction of specific constraints (these can be considered as local search heuristic more than mutation operators, see our discussion in the final Sect.) [2]. Another aspect that Alkan and Ozcan investigated was the comparison of various hill-climbers. They designed a specific hill-climber for each type of constraint and all hill-climbers were combined under a single hill-climber controlling the whole local search phase.

This idea of designing specialised local search heuristics to target a particular constraint or group of constraints has also been investigated by Viana et al. in what they call *constraint oriented neighbourhoods* [45]. Their idea is to use, for a given problem, neighbourhood structures that explicitly take into account the particular characteristics of the problem constraints. Then, during the local search the neighbourhood moves are chosen according to the constraints that are not satisfied in that moment. The adaptation of neighbourhood search heuristics has also been explored by Burke et al. in the

context of multi-objective optimisation where different heuristics are targeted to different objectives (for more details see [11]).

The progress or success rate of different operators can be assessed during the search. Then, their application can be adapted accordingly to the conditions of the search process. For example, Basseur et al. used a scheme to measure the progress of various mutation operators when tackling multi-objective flow-shop scheduling problems [3]. They implemented various mutation operators which are applied with the same probability at the beginning of the search. As the search progresses, the decision as to which mutation operator to use is made dynamically. The generated solutions are evaluated before and after the application of each mutation operator. Depending on the success of the operator, they calculate an average growth value which is used to dynamically adjust the probability of each mutation operator. More specifically:

1. When a mutation operator M is applied, a solution $M(x)$ is generated from a solution x.
2. The progress of the mutation operator M when applied to solution x is 1 if x is dominated by $M(x)$, 0 if x dominates $M(x)$ and 0.5 otherwise. A solution x dominates a solution y if x is as good as y in all objectives and better in at least one of them (see [43]).
3. The average $Progress(M(i))$ of each mutation operator M is calculated by summing all the progresses of M and dividing it by the number of solutions to which M was applied.
4. Then, the probability of each mutation operator is adjusted using (1) where η is the number of mutation operators and δ indicates the minimal ratio value permitted for each operator. That is, δ is a parameter that permits to keep each operator even if the progress of the operator is too poor.

$$P_{M(i)} = \frac{Progress(M(i))}{\sum_{j=1}^{\eta} Progress(M(j))} \times (1 - \eta \times \delta) + \delta \ . \tag{1}$$

As discussed here, although some research has been carried out on how to adapt the application of different genetic and local search operators and heuristics (*memes*) throughout the search, in most cases the *memes* have been designed before the search and remain unchanged during the process. The notion of evolution of *memes* instead of evolution of *genes* in the context of timetabling was first suggested by Ross et al. who said: "we suggest that a GA might be better employed in searching for a good algorithm rather than searching for a specific solution to a specific problem" [40]. As also argued by Krasnogor, the evolution of *memes* is an aspect that deserves more attention in order to design more advanced and improved memetic systems [29].

5 Other Ideas for Memetics

5.1 Cooperative Local Search

As discussed in Sect. 2, most memetic algorithms result from the hybridisation of evolutionary algorithms and the incorporation of a variety of specialised *helpers* such as elaborate encodings, local search heuristics, etc., from the knowledge of the problem domain. Another form of hybridising evolutionary methods and local search is by adding some elements of evolutionary algorithms (such as genetic operators, populations of solutions, a common annealing schedule, etc.) into a *cooperative local search* scheme. This form of hybridisation is illustrated in Fig. 4. To describe the difference between most memetic approaches and *cooperative local search*, we also refer to Fig. 3.

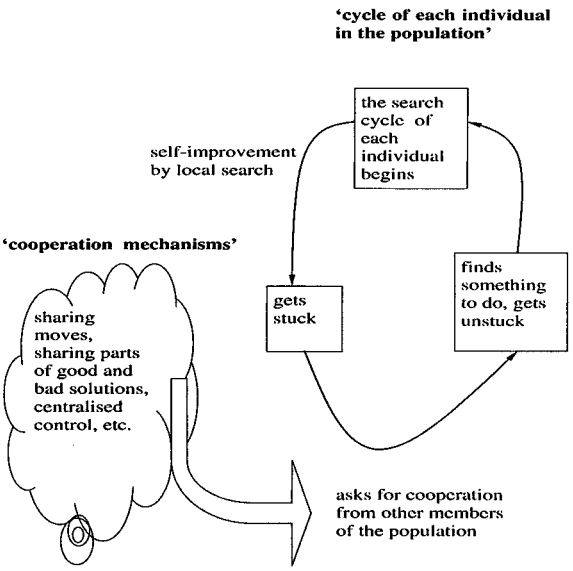


Fig. 4. In a *cooperative local search* scheme, each individual carries out its own local search. When an individual gets stuck it asks for the cooperation of the population in order to find something to do to get unstuck and continue the search from another position in the solution space. The results achieved by each individual may be different at different times and this encourages diversity within the population.

While in most memetic algorithms as depicted in Fig. 3, the structure of the evolutionary algorithm based on generations is maintained, in the *cooperative local search* approach, the self-improving individual cycle is the *driving mechanism* and the *helpers* come from evolutionary methods. This form of hybridisation was proposed in [31] as an alternative strategy to combine the explorative capability of genetic search with the intensification ability of local

searchers. This type of hybrid, can be beneficial in those problems in which the recombination of solutions requires the design of specialised encodings, repairing methods or recombinative operators. This is the case in most of the scheduling and timetabling problems as discussed above. With the *cooperative local search* approach, a population of local searchers evolve mainly by self-improvement. But the individuals also share information during the search process with the rest of the population and hence, a form of recombination can be achieved. In [31] this concept was applied to the space allocation problem and it proved to be very effective. The cooperation between individuals was accomplished by maintaining a pool of good and bad parts of solutions. Then, the cooperation (possible recombination) between individuals in the population is asynchronous as opposed to most memetic algorithms. In a synchronous mechanism (as in memetic algorithms) the cooperation is regulated by generations. In an asynchronous mechanism (as proposed in *cooperative local search*) the individuals cooperate between them at any required time. When an individual gets stuck, it asks for the help of the population and this is implemented by accessing the pool of genes. There are several variants of this approach that can be explored following the terminology of hybrid metaheuristics proposed by Talbi in [44](see also [38]). For example, the cooperation can be synchronous or asynchronous, the explorers can employ the same (homogeneous) or different (heterogeneous) local search procedures and also can search the same or different areas of the solution space (global, partial or functional).

5.2 Teams of Heuristics

Some researchers have investigated the design of search algorithms based upon a collection of simple local search heuristics. The idea is that a set of simple local search heuristics can be applied in a systematic or adaptive way to tackle difficult combinatorial optimization problems. Examples of these approaches are:

Variable neighbourhood search [21]. In variable neighbourhood search a number of different neighbourhood structures are used in a systematic fashion to attempt improvements in the current solution while attempting to avoid getting stuck in poor local optima.

A-teams of heuristics [41]. An A-team of heuristics consists of a set of constructors (to generate solutions), a set of improvers (to perform local search and improve solutions) and a set of destructors (to eliminate poor quality solutions). All these heuristics operate on a population of solutions and all of them behave like independent agents cooperating in an asynchronous fashion.

Hyper-heuristics [10]. A hyper-heuristic can be described as a heuristic that manages the application of a set of heuristics (which can be simple neighbour-

hood heuristics or elaborate meta-heuristics) during the search. For example, a hyper-heuristic might, at each moment during the search, make the decision as to which heuristic to use according to the historical performance of each heuristic. The central idea is that the hyper-heuristic learns and adapts itself dynamically during the search process.

The ideas behind the above approaches can be used to inspire more strategies for designing more advanced memetic algorithms for scheduling, timetabling problems and other combinatorial optimisation problems. Specifically, the self-adaptation of local search heuristics would help to further develop the idea of meme evolution [29].

6 Final Remarks and Future Research

6.1 Scheduling and Timetabling: Interesting Domain for Memetic Algorithms

Scheduling and timetabling problems represent an interesting domain for the application of memetic algorithms. There are already a number of applications reported in the literature (e.g. [1, 2, 3, 5, 6, 9, 12, 13, 16, 19, 22, 26, 36]) but certainly there are still several promising research directions to be explored. We can summarise some of the reasons for which memetic algorithms are a good approach to solve scheduling and timetabling problems as follows:

- The size of the search space in scheduling and timetabling problems is huge for most real-world problems, and a good explorative ability, which memetic approaches have, is required.
- These problems are highly constrained and therefore, most of the times it is easier to self-improve solutions that to recombine them. This highly constrained nature also leads to the design of specialised solution encodings. Memetic algorithms also incorporate specialised encodings and operators for self-improvement of solutions, which are based on the knowledge of the problem domain.
- A complete fitness evaluation function is time consuming in many real-world scheduling and timetabling problems. Using approximate evaluation in memetic algorithms is more robust than in single-solution methods. This is because having a population of new solutions instead of only one new solution, helps to reduce the the effect of the error in the fitness estimation.

6.2 Ideas That Have Been Investigated

In the literature and from the previous sections in this paper, we can summarise some of the ideas that have been investigated with respect to the application of memetic algorithms for scheduling and timetabling problems:

- The design of specialised encodings (e.g. linked lists), local search strategies, genetic operators, and multi-phased methods, all assist the algorithm in dealing with infeasible solutions and have received considerable attention.
- Approximate fitness evaluation has been used in memetic algorithms but to a lesser extent than the above.
- Some work has been carried out on analysing the fitness landscape to inform the design of the memetic algorithm. That is, to select more appropriate operators and to tune the genetic and local search phases. However, this analysis is usually performed prior to the implementation of the memetic approach.
- The balance between the genetic and the local search parts of memetic algorithms has received considerable attention particularly in recent years. The aspects studied here include: the sophistication of operators, the selection of solutions to which apply the operators (including elitist strategies such as archives of solutions), the computing time allocated to the genetic and the local search, the tuning and balance of the parameters (previous to the search), use of population control mechanisms (such as common annealing schedule), etc.

6.3 A Few Thoughts

Designing a memetic algorithm is frequently associated with the incorporation of knowledge from the problem domain in the form of *helpers* to evolutionary algorithms. We should be careful because almost every new piece of specific knowledge that is added to a memetic algorithm can potentially produce improved results. Then, we can many times keep designing a 'new' or an 'improved' version of the memetic algorithm, i.e. an incremental design of algorithms. We should concentrate on the main ideas and strategies without getting lost in the details of the different implementations.

Krasnogor proposes in [29] a grammar to formulate a wide range of memetic algorithms. He also expresses that the grammar can help to envisage many more different implementations of memetic algorithms that have not been investigated. It would be interesting to investigate such variants. But we should also be careful and focus on the main strategies for designing memetic algorithms and not necessarily on the many ways in which they are combined.

6.4 Suggested Future Research

We argue here that by studying in detail the problem domain, there is always room to create a 'new' memetic approach. If memetic algorithms simulate the evolution of ideas, should we not take for granted that many different ideas would exist so we should concentrate more on the mechanism to evolve these ideas instead of manufacturing these ideas by hand before the search process? The research carried out by researchers on the design of memetic algorithms

for scheduling and timetabling has contributed enormously to our understanding of specialised encodings, operators, heuristics, evaluation routines, etc. and their inter-relationships. But as suggested in [29, 33], to learn more about the memetic paradigm, we should concentrate now on using this knowledge for designing approaches to evolve *genes* and *memes* during the search and also automatically select *memes* given the problem domain and also the particular instance characteristics. This was also proposed by Ishibuchi et al. in [24] where they suggested to investigate the dynamic adaptation of the balance between local and genetic search. We also believe that an important challenge in this area is to investigate the design of self-adaptive memetic systems, i.e. the *evolution of genes and memes.*

References

1. Aickelin U., Dowsland K.A. (2000). Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem. Journal of scheduling, 3(3), 139–153.
2. Alkan A., Ozcan E. (2003). Memetic algorithms for timetabling. Proceedings of the 2003 congress on evolutionary computation (CEC 2003), 1796–1802, IEEE press.
3. Basseur M., Seynhaeve F., Talbi E.G. (2002). Design of multi-objective evolutionary algorithms to the flow-shop scheduling problem. Proceedings of the 2002 congress on evolutionary computation (CEC 2002), IEEE press, 1151–1156.
4. Blazewicz J., Domschke W., Pesch E. (1996). The job shop scheduling problem: conventional and new solution techniques. European journal of operational research, 93, 1–33.
5. Burke E.K., Newall J.P. (1999). A multi-stage evolutionary algorithm for the timetable problem. IEEE transactions on evolutionary computation, 3(1), 1085–1092.
6. Burke E.K., Newall J.P., Weare R.F. (1996). A memetic algorithm for university exam timetabling. The practice and theory of automated timetabling: Selected papers from the 1st international conference on the practice and theory of automated timetabling (PATAT 1995), Lecture notes in computer science, 1153, 241–250, Springer.
7. Burke E.K., Cowling P., Landa Silva J.D. (2001). Hybrid population-based metaheuristic approaches for the space allocation problem. Proceedings of the 2001 congress on evolutionary computation (CEC 2001), IEEE press, 232–239.
8. Burke E.K., Cowling P., Landa Silva J.D., Petrovic S. (2001). Combining hybrid metaheuristics and populations for the multiobjective optimisation of space allocation problems. Proceedings of the 2001 genetic and evolutionary computation conference (GECCO 2001), Morgan kaufmann, 1252–1259.
9. Burke E., Cowling P., De Causmaecker P., Vanden Berghe G. (2001). A memetic approach to the nurse rostering problem. Applied intelligence, 15(3), 199–214.
10. Burke E.K., Hart E., Kendall G., Newall J., Ross P., Schulemburg S. (2003). Hyper-heuristics: an emerging direction in modern search technology. In: Glover F.W., Kochenberger G.A. (eds.), Handbook of metaheuristics, Kluwer academic publishers, 2003.

11. Burke E.K., Landa Silva J.D., Soubeiga E. (2003). Hyperheuristic approaches for multiobjective optimisation. Proceedings of the 5th metaheuristics international conference (MIC 2003), Kyoto Japan. Extended version available from the authors.
12. Burke E.K., Smith A. (1999). A memetic algorithm to schedule planned maintenance for the national grid. ACM Journal of experimental algorithmics, 4(1), 1084–1096.
13. Burke E.K., Smith A. (2000) Hybrid evolutionary techniques for the maintenance scheduling problem, IEEE transactions on power systems, 15(1), 122–128.
14. Colorni A., Dorigo M., Maniezzo V. (1998). Metaheuristics for high school timetabling, Computational optimization and applications, 9, 275–298.
15. Corne D., Dorigo M., Glover F. (eds.) (1999). New ideas in optimisation. McGraw Hill.
16. Deris S., Omatu S., Ohta H., Saad P. (1999). Incorporating constraint propagation in genetic algorithm for university timetable planning. Engineering applications of artificial intelligence, 12, 241–253.
17. Erben Wilhelm (2001). A grouping genetic algorithm for graph colouring and exam timetabling. The practice and theory of automated timetabling III: Selected papers from the 3rd international conference on the practice and theory of automated timetabling (PATAT 2000), Lecture notes in computer science, 2079, 132-156, Springer.
18. Erben W., Keppler J. (1996). A genetic algorithm solving a weekly course-timetabling problem. The practice and theory of automated timetabling: Selected papers from the 1st international conference on the practice and theory of automated timetabling (PATAT 1995), Lecture notes in computer science, 1153, 198–211, Springer.
19. Ernst A.T., Jiang H., Krishnamoorthy M., Sier D. (2004). Staff scheduling and rostering: a review of applications, methods and models. European journal of operational research, 153, 3–27.
20. Grefenstette J.J., Fitzpatrick M.J. (1985). Genetic search with approximate function evaluation. Genetic algorithms and their applications: Proceedings of the first international conference on genetic algorithms, 112–120.
21. Mlandenovic N., Hansen P. (1997). Variable neighbourhood search. Computers and operations research, 24(11), 1097–1100.
22. Ishibuchi H., Murata T., Tomioka S. (1997). Effectiveness of genetic local search algorithms, Proceedings of the seventh international conference on genetic algorithms, 505–512.
23. Ishibuchi H., Yoshida T., Murata T. (2002b). Selection of initial solutions for local search in multiobjective genetic local search. Proceedings of the 2002 congress on evolutionary computation (CEC 2002), 950–955, IEEE press.
24. Ishibuchi H., Yoshida T., Murata T. (2003). Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling. IEEE transactions on evolutionary computation, 7(2), 204–223.
25. Jaszkiewicz A. (2002). Genetic local search for multi-objective combinatorial optimization. European journal of operational research, 137(1), 50–71.
26. Kawanaka H., Yamamoto K., Toshikawa T., Shinogi T., Tsuruoka S. (2001). Genetic algorithm with the constraints for nurse scheduling problem. Proceedings of the 2001 congress on evolutionary computation (CEC 2001), 1123–1130, IEEE press.

27. Knowles J.D., Corne D.W. (2000). M-PAES a memetic algorithm for multiobjective optimization. Proceedings of the 2000 congress on evolutionary computation (CEC 2000), 325–332, IEEE press.
28. Knowles J.D., Corne D.W. (2002). Towards landscape analyses to inform the design of a hybrid local search for the multiobjective quadratic assignment problem. In: Soft computing systems: design, management and applications, 271–279, IOS Press.
29. Krasnogor N. (2002). Studies on the theory and design space of memetic algorithms. PhD thesis, Faculty of computing, engineering and mathematical sciences, University of the West of England, UK.
30. Krasnogor N., Smith J. (2000). A memetic algorithm with self-adaptive local search: TSP as a case study. Proceedings of the 2000 genetic and evolutionary computation conference (GECCO 2000), 987–994, Morgan kaufmann.
31. Landa-Silva J.D. (2003). Metaheuristic and multiobjective approaches for space allocation. PhD Thesis, School of Computer Science and Information Technology, University of Nottingham.
32. Merz P, Freisleben B. (1999). Fitness landscape and memetic algorithm design. In: Corne D., Dorigo M., Glover F. (eds.), New ideas in optimisation, McGraw Hill, 245–260.
33. Moscato P. (1999). Memetic algorithms: a short introduction. In: Corne D., Dorigo M., Glover F. (eds.), New Ideas in Optimisation, 219–234, McGraw Hill, 1999.
34. Moscato P. (2002). Memetic algorithms' home page. Online, available at http://www.densis.fee.unicamp.br/moscato/memetichome.html.
35. Murata T., Ishibuchi H., Gen M. (2000). Cellular genetic local search for multiobjective optimization. Proceedings of the 2000 genetic and evolutionary computation conference (GECCO 2000), Morgan kaufmann, 307–314.
36. Paechter B., Cumming A., Norman M.G., Luchiam H. (1996). Extensions to a memetic timetabling system. The practice and theory of automated timetabling: Selected papers from the 1st international conference on the practice and theory of automated timetabling (PATAT 1995), Lecture notes in computer science, 1153, 251–265, Springer.
37. Pinedo Michael (1995). Scheduling, theory, algorithms, and systems. Prentice-hall.
38. Preux Ph., Talbi E.G. (1999). Towards hybrid evolutionary algorithms. International transactions in operational research, 6, 557–570.
39. Randall M., Abramson D. (2001). A general meta-heuristic based solver for combinatorial optimisation problems. Computational optimization and applications, 20, 185–210.
40. Ross P., Hart E., Corne D. (1998). Some Observations about GA-based exam timetabling. The practice and theory of automated timetabling II: Selected papers from the 2nd international conference on the practice and theory of automated timetabling (PATAT 1997), Lecture notes in computer science, 1408, 115–129, Springer.
41. Salman F.S., Kalagnaman J.R., Murthy S., Davenport A. (2002). Cooperative strategies for solving bicriteria sparse multiple knapsack problem. Journal of heuristics, 8, 215–239.
42. Schaerf A. (1999). A Survey of automated timetabling. Artificial intelligence review, 13, 87–127.

43. Steuer Ralph E. (1986). Multiple criteria optimization: theory, computation and application. Wiley.
44. Talbi E.G. (2002). A Taxonomy of hybrid metaheuristics. Journal of heuristics, 8, 541–564.
45. Viana A., Pinho de Sousa J., Matos M.A. (2003). GRASP with constraint neighbourhoods: an application to the unit commitment problem. Proceedings of the 5th metaheuristics international conference (MIC 2003), 2003.
46. Watson J.P., Rana S., Whitley L.D., Howe A.E. (1999). The impact of approximate evaluation on the performance of search algorithms for warehouse scheduling. Journal of scheduling, 2, 79–98.
47. Wren A. (1996). Scheduling, timetabling and rostering, a special relationship?. The practice and theory of automated timetabling: Selected papers from the 1st international conference on the practice and theory of automated timetabling (PATAT 1995), Lecture notes in computer science, 1153, 46–75, Springer.

Memetic Algorithms for Multiobjective Optimization: Issues, Methods and Prospects

Joshua Knowles[1] and David Corne[2]

[1]Dept of Chemistry, UMIST, PO Box 88, Sackville St, Manchester M60 1QD
[2]Dept of Computer Science, Harrison Building, University of Exeter EX4 4QF

Summary. The concept of optimization—finding the extrema of a function that maps candidate 'solutions' to scalar values of 'quality'—is an extremely general and useful idea that can be, and is, applied to innumerable problems in science, industry, and commerce. However, the vast majority of 'real' optimization problems, whatever their origins, comprise more than one objective; that is to say, 'quality' is actually a vector, which may be composed of such distinct attributes as cost, performance, profit, environmental impact, and so forth, which are often in mutual conflict. Until relatively recently this uncomfortable truth has been (wilfully?) overlooked in the sciences dealing with optimization, but now, increasingly, the idea of *multiobjective optimization* is taking over the centre ground. Multiobjective optimization takes seriously the fact that in most problems the different components that describe the quality of a candidate solution cannot be lumped together into one representative, overall measure, at least not easily, and not before some understanding of the possible 'tradeoffs' available has been established. Hence a multiobjective optimization *algorithm* is one which deals directly with a vector objective function and seeks to find multiple solutions offering different, optimal tradeoffs of the multiple objectives. This approach raises several unique issues in optimization algorithm design, which we consider in this article, with a particular focus on memetic algorithms (MAs). We summarize much of the relevant literature, attempting to be inclusive of relatively unexplored ideas, highlight the most important considerations for the design of multiobjective MAs, and finally outline our visions for future research in this exciting area.

1 Introduction

Many important problems arising in science, industry and commerce fall very neatly into the ready-made category of *optimization problems*; that is to say, these problems are solved if we can simply find a 'solution' that maximizes or minimizes some important and measurable property or attribute, such as cost or profit. For example, we might want to find the set of routes that minimizes the distance travelled by a fleet of delivery lorries; or to find the tertiary structure of a trans-membrane protein that minimizes its free energy;

or to find the portfolio of stocks that maximizes the expected profit over the forthcoming year. Of course, solving these problems exactly might be very difficult or impossible in practice, but by applying one of numerous optimization algorithms—memetic algorithms (MAs) being one very flexible and successful possibility—very good solutions can often be found.

However, there is a caveat: maximizing or minimizing a single, lone attribute can, in many cases, be a very bad thing to do. Consider designing a car with the single objective of minimizing its weight: other desirable attributes like safety, comfort, and capacity would be severely compromised as a result. And so it is in many other generic problems: maximizing profit often leads to compromises in environmental impact or customer satisfaction; minimizing production costs often leads to decreased reliability; and minimizing planned time to completion of a project often leads to soaring costs for overrunning. Thus, it is easy to appreciate that most 'real' optimization problems involve optimizing, simultaneously, more than one single attribute.

Now, given that most problems are as we've described—'multiobjective' in nature—, what are the options for tackling them? There are basically three: (1) ignore some of the attributes entirely and just optimize one that looks most important; (2) lump all the objectives together by just adding them up or multiplying them together and then optimize the resulting function; or (3) apply a multiobjective algorithm that seeks to find *all* the solutions that are *nondominated* (we define this later but, roughly speaking, nondominated solutions are those that are optimal under *some/any* reasonable way of combining the different objective functions into a single one). The first and second options are very common and the third less so. However, (3) is rapidly gaining popularity as it is more and more understood that (1) and (2) can be very damaging in practice—solving the problem might be very satisfying to the algorithm or MA practitioner, but the resulting solution may be far from optimal when it is applied back in the real world. Thus, in this chapter, we will argue that option (3)—seeking *multiple, distinct solutions* to a problem, conferring different tradeoffs of the objectives,—is the essence of true multiobjective optimization (MOO).

Doing true multiobjective optimization with memetic algorithms requires a few salient adaptations to the normal design principles. Clearly, since we need to find multiple, distinct solutions, the design of multiobjective MAs will be heavily affected by the need to encourage and preserve *diversity*. Indeed, much of the research in evolutionary algorithms (EAs) for MOO has concerned itself primarily with this issue, but with MAs the use of local search introduces further complications for achieving diversity that must be resolved.

The goal of finding multiple solutions also dictates that the MA incorporate some means of storing the best solutions discovered. While MAs are already endowed with a population, some research in EAs for MOO has found that methods that exploit secondary populations, or *archives*, seem to perform better than single-population approaches, and *elitism* based on archives appears to be particularly effective in improving search capability. Thus, ques-

tions about how to control and use multiple populations (or non-fixed size populations) are somewhat more relevant and pressing in MOO than they are in 'normal' optimization.

A second key distinction of MOO, closely related to the need for multiple solutions, is the inherent *partial* ordering of solutions in terms of their overall quality, which characterises MOO. This impacts on many aspects of search and how it should be conducted. In particular, the simple comparison of two solutions is fraught with difficulties. Local search, which relies upon such comparisons being made, must be re-defined in some way, and there are several competing possibilities.

There are also innumerable possibilities concerning the overall organization of the search—how the set of tradeoff solutions (the nondominated set) is to be built up, over time. Very coarsely, should we try to sweep across the objective space from one 'edge' to the other, i.e. improving one combination of objectives at a time, or should we more try to push the entire 'surface' down in parallel, improving the whole currently nondominated set at once? In either case, what is the best way to exploit the population(s) and the different local searchers at our disposal?

In the remainder of this article, we will try to fill the reader in on the core issues we have but sketched here, mapping out the little that is known and has been tried so far, and speculating about where further research may be most fruitful. In section 2, some MOO applications are outlined to give some idea of their range and differing characteristics. The mathematical definition of the MOO problem is then given, and Pareto optimization is described. Section 3 visits a large number of metaheuristics for MOO and identifies concepts and strategies that, we suggest, may be useful as components in a memetic algorithm. In section 4, we elaborate on other issues in MOO research that may impact on the design and application of multiobjective MAs, including how to measure performance, how multiple populations can be used, and available test functions. Section 5 provides a focused review of existing MAs for MOO, while section 6 proposes some principles for designing more advanced MAs. The last section considers future research directions and gives some recommendations for immediate investigation.

2 A Brief Introduction to MOO

2.1 MAs and MOO: a good combination

The impressive record of memetic algorithms in producing high quality solutions in combinatorial optimization and in real-world applications (e.g. see page 220 [18]) is sometimes cited as a testament to their inherent effectiveness or robustness as black-box searchers. However, since the advent of the No Free Lunch theorems [109, 19, 21], we know that MAs, like any other search algorithm, are only really good to the extent to which they can be 'aligned'

to the specific features of an optimization problem. Nonetheless, MAs, like their forebears, EAs, do have one unassailable advantage over other more traditional search techniques: that is their *flexibility*. EAs and MAs impose no requirement for a problem to be formulated in a particular constraint language, and do not ask for the function to be differentiable, continuous, linear, separable, or of any particular data-type. Rather, they can be applied to *any* problem for which the following are available: (1) some (almost) any way to encode a candidate solution to the problem, and (2) some means of computing the quality of any such encoded solution—the so-called objective function.

This flexibility has important advantages. As has been observed in [83], there are basically two ways to solve optimization problems: one is to choose some traditional technique and then simplify or otherwise alter the problem formulation to allow the problem to be tackled using the chosen technique; the other is to leave the problem formulation in its original form and use an EA, MA, or other metaheuristic. Clearly, the latter is preferable because the answer one arrives at is (at least) to the right question, not to a question which may have been distorted (perhaps so much so as to be irrelevant to the real question), simply to fit in with the requirements of a chosen search method.

In [19], the advantages of 'leaving the problem alone' (and applying a flexible search technique) was reiterated and used to make a further, compelling point. How often are optimization problems in the real world (or from real-world origins) squeezed and stretched into the strait-jacket of a single-objective formulation, when their natural formulation is to have *multiple* objectives? Doesn't the same observation of [83] apply in this case, too? What is the effect of throwing away objectives or of combining them together as a weighted, linear sum, as is so often done? If we are to believe the EA/MA mantra about tackling problems in their original formulation, shouldn't we be tackling *multiobjective* problems in the same way?

Of course, the answer is that we should. And there are two reasons: (1) simplifying a problem *does* change it irrevocably and make it irrelevant in many cases, and (2) with EAs, including MAs, we have the capability to tackle multiobjective problems in their native form and indeed the cost of doing so is demonstrably not high.

2.2 Some example MOO problems

One could argue that engineering is the art of finding the good compromise; and indeed many problems encountered in engineering do have multiple and distinct objectives. Fortunately, we are now gradually seeing that the optimization problems being formulated in various engineering sub-disciplines are respecting the multiobjective nature of the underlying problem. For example, civil engineering tasks such as designing water networks are being seen as multiobjective optimization problems [48, 13, 14, 15], as is power

distribution [3, 4, 6, 79], and various telecommunications network optimization tasks [73, 72]. And, at the other end of the engineering spectrum, the design of various types of controllers has been aided by such an approach [2, 8, 104, 24, 38, 41] for some years now.

Scheduling and *timetabling* are two huge classes of *planning problem* that can involve a multitude of different objectives. In scheduling, problems tackled in the academic literature often consider only one objective: minimizing the makespan—the total time needed to complete all jobs. However, the reality of scheduling in factories, space programmes, engineering projects and so forth is far more complex. Reducing the makespan is undoubtedly one objective but other important ones are mean and total tardiness, mean flow time, mean waiting time, and the mean and total completion time. In addition to these objectives there are often a number of constraints. If all these constraints are modelled as 'hard', the resulting schedules can be brittle and sub-optimal. By softening some of these constraints (those which are not really inviolable) and treating them as further objectives, great gains can sometimes be made for minute sacrifices elsewhere. Frequently, the robustness of a schedule to unforeseen changes, such as late arrival times of materials, machine failures and so forth, should also be modelled. Making robustness an objective enables planners to consider fully the tradeoffs between allowing some slack, versus 'risking it' and going for the absolutely optimal schedule.

Much the same can be said for timetabling, particularly with regard to constraints. More often than not, timetabling problems are tackled as constraint satisfaction problems in which hard constraints must be satisfied and soft constraint violations should be minimized. However, the latter are usually just added together, leading to absurd situations, where, for example, the optimization algorithm 'chooses' that nineteen students having consecutive exams is better than 14 having to get up early one morning, together with 6 invigilators working through their lunch break! Fortunately, the recognition that these problems are multiobjective, and need to be tackled as such, is leading to more research in this vein: e.g. [46, 51, 59, 62] in scheduling, and [91, 12] in timetabling.

There are a whole host of other varied MOO applications emerging on a more and more frequent basis: from the training of neural networks [1, 11, 111, 93], to various design applications [92, 95, 5], to dealing with the challenges of dynamic optimization [110, 35]. The short survey presented here scratches but the surface, and the reader is directed to [32] and [16] for more comprehensive reviews.

Basic MOO definitions

An unconstrained multiobjective optimization problem can be formulated as

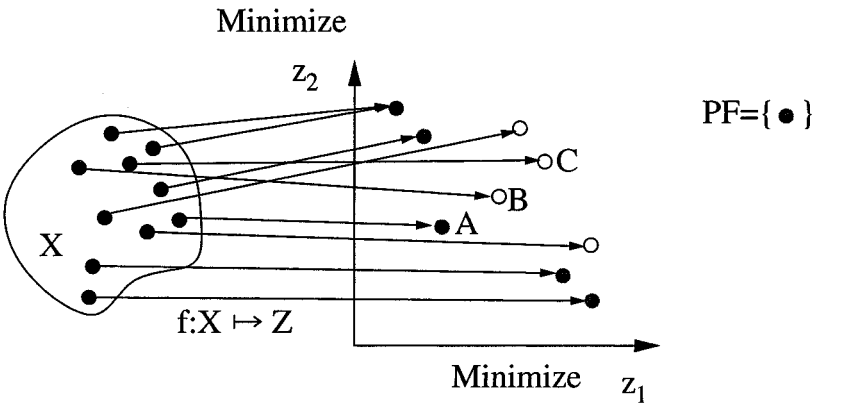


Fig. 1. An illustration of a multiobjective optimization problem with a search space X, a vector fitness function $\mathbf{f}$ that maps solutions in X to objective vectors made up of two component 'costs' z_1 and z_2 to be minimized. The solid objective vectors are nondominated and comprise the Pareto front. The solutions corresponding to these points are Pareto optimal. The relation between the three objective vectors A, B, and C is $A < B < C$

$$\begin{aligned} &\text{"minimize"} \quad \mathbf{z} = \mathbf{f}(\mathbf{x}) \\ &\text{where} \qquad \mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_k(\mathbf{x})) \\ &\text{subject to} \quad \mathbf{x} \in X \end{aligned} \tag{1}$$

involving $k \geq 2$ different *objective functions* $f_i : \Re^n \mapsto \Re$ to be minimized simultaneously. Note that if f_i is to be maximized, it is equivalent to minimize $-f_i$.

The term "minimize" appears in quotes in (1) to emphasise that the exact meaning of the vector minimization must be specified before optimization can be performed. That is, we need to specify a binary relation on objective vectors in order to form a (partial) ordering of them. Although different possibilities exist, in this chapter we will be concerned only with the component-wise order relation, which forms the basis for Pareto optimization as defined below (also see figure 1).

Definition 1 *The component-wise order relation* $<$ *is defined as* $\mathbf{z}^r < \mathbf{z}^s \Leftrightarrow z_i^r \leq z_i^s, i = 1..k \wedge \mathbf{z}^r \neq \mathbf{z}^s$.

Definition 2 *A solution* $\mathbf{x}^* \in X$ *is called* **Pareto optimal** *if there is no* $\mathbf{x} \in X$ *such that* $\mathbf{f}(\mathbf{x}) < \mathbf{f}(\mathbf{x}^*)$. *If* $\mathbf{x}^*$ *is Pareto optimal,* $\mathbf{z}^* = \mathbf{f}(\mathbf{x}^*)$ *is called (globally)* **nondominated**. *The set of all Pareto optima is called the* **Pareto optimal set**, *and the set of all nondominated objective vectors is called the* **Pareto front (PF)**. *Finding an approximation to either the Pareto optimal set or the Pareto front is called* **Pareto optimization**.

More generally, Miettinen [84] defines solving a multiobjective problem as finding a Pareto optimal solution to (1) that also satisfies a *decision maker* (DM), who knows or understands something more about the problem. Such a definition brings into play the science of multi-criteria decision making (MCDM), where methods are used to model the preferences of decision makers in order to aid them in comparing and choosing solutions. Thus, according to this definition, solving a multiobjective problem, involves both search and *decision making*, and to accomplish this, one of three general approaches is normally taken:

1. *A priori* optimization
2. *A posteriori* optimization
3. Interactive optimization

In *a priori* optimization, the decision maker is consulted before search and a mathematical model of her preferences is constructed (following one of several regimes for this), and used in the search to evaluate all solutions. The best solution found, according to the model, is returned and represents the outcome of the optimization process with no further input from the DM. The drawback with such methods is obvious: decision makers find it very hard to give adequate models determining which solutions they prefer, without knowing or having any idea what it is possible to attain, and how much one objective may have to be sacrificed with respect to others. Furthermore, notice that this method, in a sense, places all the additional work associated with MOO, firmly with the DM, and leaves the search problem as seen by a search algorithm, in much the same form as for normal optimization, i.e. one solution must be found and all solutions are comparable (using the DM's *a priori* preference model). For this reason, we do not consider *a priori* optimization any further in this article, as standard MAs could be used (or trivially adapted) to this case.

A posteriori optimization approaches the multiobjective problem from the reverse angle. First, search is conducted to find the Pareto optimal set (or an approximation/representation thereof) and the DM will then choose between these alternatives by inspection (with or without using some mathematical decision-making aid). The disadvantage (according to [84]) of this approach is the difficulty DMs may have in visualizing the different alternatives and choosing between them, particularly if a large number have been generated. Nonetheless, the problem of decision-making is in our opinion definitely aided by knowing something about what solutions are possible. Thus, *a posteriori* methods move at least some of the work from the DM to the search algorithm, which now is given the task of searching for multiple different solutions. Exactly what solutions the search algorithm finds will depend upon how, internally, it evaluates solutions, but it should be oriented towards finding Pareto optima. And in order to give the DM what she needs—real *alternatives*—the Pareto optima should not be all in one region of the objective space, but should be spread as far and wide as possible. (Being more precise than this is

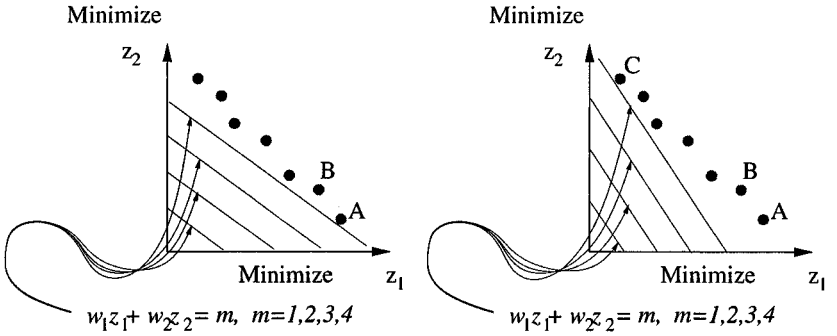


Fig. 2. Illustration of the drawbacks of scalarizing objectives using the weighted sum approach. The figures show a Pareto front and lines of equal cost under a weighted sum. In the left figure, A is the optimal solution. A slight change to the weights, slightly altering the angle of the isocost lines, as shown in the figure on the right, makes C the optimal solution. The nondominated solution B is 'non-supported' – not on the convex hull of the Pareto front. Therefore it is not optimal under any linear combination of the objectives

problematic as seen in section 4.1 where we will discuss how to evaluate different approximations to Pareto fronts). In any case, *a posteriori* optimization is the method we advocate in this article, in preference to *a priori* methods, and we assume in the remainder of the article that finding a 'good' approximation to the whole Pareto front is the goal of multiobjective optimization, leaving decision-making as a separate issue.

The interactive methods of search combine *a priori* and *a posteriori* methods in an iterative funnelling of goals, preferences and solutions discovered. These methods are probably preferable to *a posteriori* methods, since they limit the choices shown to a DM at any instant, and focus the search on a smaller area. However, we do not make more than a passing reference to them in what follows, for two reasons. First, because, so far, relatively little research in the EA community has been directed to this general approach, so it is difficult to make judgments or recommendations. And more importantly, because effectively, from a search point of view, the problem is still one of finding a *set* of alternatives, albeit reduced in size, and so we can regard it as a special case of *a posteriori* optimization.

2.3 An overview of methods for generating a Pareto front

What methods can we use to build up an approximation to the true Pareto front (our goal as outlined above)? Leaving aside, for the moment, the finer details of the overall algorithm design, the initial question is simply: how can any solution be evaluated so that some form of heuristic search can be effected?

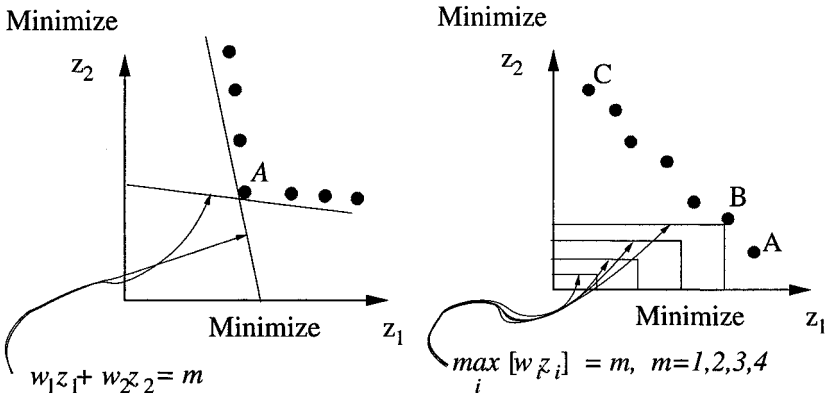


Fig. 3. The figure on the left shows a Pareto front where even a large change in the weights of a weighted sum scalarization would result in finding the same solution. On the right, the weighted Tchebycheff problem (equation 3) can find non-supported Pareto optima, as shown. Here, the reference point is taken as the origin

There are a great variety of answers possible. One large family of methods is to replace (1) with some parameterized single *scalarizing* function to minimize, such as a weighted sum of the objectives:

$$\text{minimize } \sum_{i=1}^{k} w_i.f_i(\mathbf{x}) \tag{2}$$

where we usually specify $\sum_k w_i = 1$ and $w_i \geq 0$, for $i \in 1..k$. Then, by varying the weighting parameters w_i in some systematic way, a representation of the PF can be built up. The weighted sum is only one possible method in this family of *scalarizing methods* and has some serious drawbacks. Only supported solutions—those on the convex hull of the PF—will be generated by minimizing the weighted sum. Furthermore, a small change in the weights can cause big changes in the objective vectors (see figure 2); while, on the other hand, very big changes in the weights may lead to the same or very similar vectors (figure 3, left). Other methods in this family that can generate the non-supported solutions are possible, e.g. the weighted Tchebycheff problem:

$$\text{minimize } max_{i \in 1..k}[\, w_i | f_i(\mathbf{x}) - z_i^* | \,] \tag{3}$$

where $\mathbf{z}^*$ is a reference point beyond the *ideal point*, i.e. each of its components is less than the minimum value possible on the corresponding objective. With such a reference point correctly specified, every Pareto optimal solution minimizes the function for some particular value of the weights. However, as with the weighted sum, in order to achieve an 'even sampling' of the Pareto front, care must be taken with how the weights are adjusted.

Other parameterized scalarizing methods include the epsilon-constraint method and achievement scalarizing functions: see [84] for further details.

Notice that these methods are suitable for exact algorithms, local searchers and so forth, since they effectively transform the problem back into a single-objective problem temporarily. So, for MAs, they may well be used as part of the overall algorithm.

With many metaheuristics, particularly traditional EAs, however, it is not necessary to have an explicit function to minimize, but only some means of estimating *relative* fitness (as in EA populations) or accepting/rejecting neighbour solutions (as in e.g., simulated annealing and tabu search). This opens the door to at least two other distinct approaches. One is to consider *alternately* one objective function then another; and there are various ways this could be organized (see section 3.5). The other approach is to use some form of relative ranking of solutions in terms of Pareto *dominance* (section 3.1 and 3.2). The latter is the most favoured approach in the EA community because it naturally suits population-based algorithms and avoids the necessity of specifying weights, normalizing objectives, and setting reference points.

In the last section, we discussed the reasons why we will restrict our working definition of MOO to be the problem of generating an approximation to the entire PF, ignoring methods that seek only a single solution. Following this, we went on to outline three general ways in which solutions could be evaluated in a search algorithm in order to effect optimization. In this section, we will expand greatly on this outline as we tour a host of metaheuristics for MOO. In addition, we will begin to appreciate two other related issues: how to build up the Pareto front during search (i.e. how to ensure a spread of solutions across it); and how memory of these solutions is organized to exploit them during search and/or to store them for presentation at the termination of the search process.

In the following we attempt a fairly broad survey of MOO algorithms in order to furnish the reader with a library of 'components' from which MAs could be constructed. We cluster different algorithms together in ad-hoc categories, as we review them.

2.4 Non-elitist EAs using dominance ranking

Goldberg in a short discussion in [44] suggested that multiple objectives could be handled in an EA using a ranking procedure to assign relative fitness to the individuals in a population, based on their relative Pareto dominance. The procedure, known as nondominated sorting, has become one of the bedrocks of the whole EMOO field. It is described and depicted in Figure 4. Although Goldberg did not implement it himself, it was not long before it gave rise to the popular NSGA [100]. The contemporaneous MOGA, of Fonseca and Fleming, [39] uses a slightly different ranking procedure based on counting the number of individuals that dominate each member of the population but otherwise the idea is very much the same.

Both NSGA and MOGA also employ *fitness sharing* [45], a procedure that reduces the effective fitness of an individual in relation to the number of other

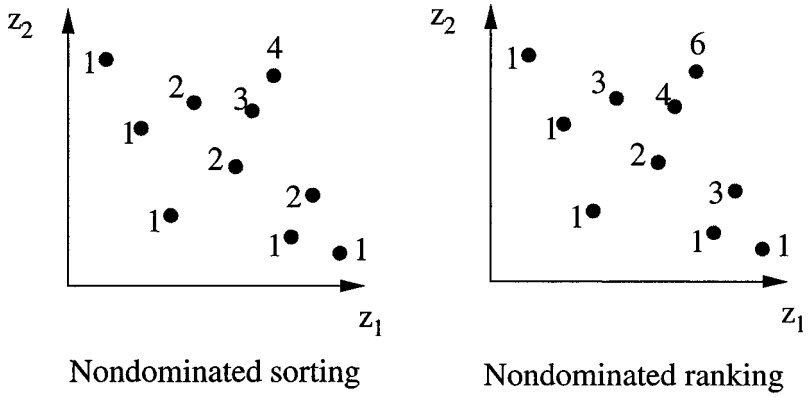


Fig. 4. On the left, individuals in a population are assigned dummy fitness values using Goldberg's nondominated sorting scheme. In this, successive iterations of the sorting procedure identify, and remove from further consideration, the nondominated set of solutions. A dummy fitness of 1 is assigned to the first set of solutions removed, and then fitness 2, and so on, 'peeling off' layers of mutually nondominated solutions. On the right, individuals in the same population are assigned fitness values using MOGA-style ranking, where fitness is 1+ the number of dominating solutions. Note, in both schemes, lower values are associated with greater fitness in the sense of reproductive opportunity or survival chances

individuals that occupy the same 'niche'. In MOO, the niche is often defined by the 'distance' of solutions to one another in the objective space, though parameter space niching may also be used. Sharing and other methods of niching have to be used in dominance-ranking MOEAs in order to encourage a spread of solutions in the objective space. Some objective-space niching methods are depicted schematically in Figure 5. Both NSGA and MOGA use similar methods to convert the shared fitness value to actual reproductive opportunity: a ranking-based selection.

The niched Pareto GA (NPGA) of Horn and Nafpliotis [53] uses, instead, tournament selection. In addition to the two individuals competing in each tournament, a sample of other individuals is used to estimate the dominance rank of the two individuals. In the case of a tie, again, fitness sharing was applied.

These EAs, NSGA, MOGA and NPGA, represent a trio that were tested and applied to more problems than any preceding algorithms for MOO, and pushed forward immensely the popularity and development of the evolutionary multiobjective optimization (EMOO) field. Most MOEAs today still use some form of dominance ranking of solutions, albeit often combined with *elitism*, and some form of niching to encourage diversity.

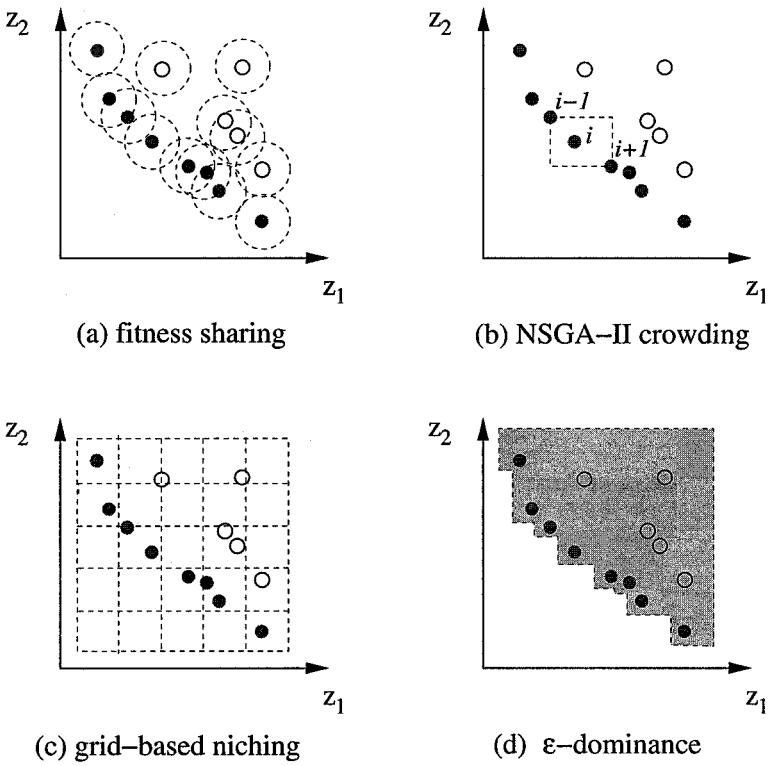


Fig. 5. Schematics depicting the different forms of niching used in various MOEAs to encourage diversity in the objective space; nondominated solutions are shown solid, and dominated ones are in outline. (a) *fitness sharing* (as used in NSGA and MOGA) reduces the fitness of an individual falling within another's niche (dashed circles), the radius being defined explicitly by a parameter. (b) NSGA-II *crowding* ranks solutions by measuring the distance of it's nearest nondominated neighbours, in each objective. (c) a grid is used in PAES, PESA and PESA-II, to estimate crowding: individuals in crowded grid regions have reduced chances of selection. (d) in ϵ-dominance archiving, a solution dominates a region just beyond itself, specified by the ϵ parameter so that the shaded region is forbidden — thus new nondominated solutions very nearby to those shown would not enter the archive

2.5 Elitist EAs using dominance ranking

Elitism in the EA terminology means the retention of good parents in the population from one generation to the next, to allow them to take part in selection and reproduction more than once and across generations.

The first multiobjective evolutionary algorithms employing elitism seem to have appeared at approximately the same time as MOGA, NSGA, and NPGA were put forward, around 1993-4 as reviewed in detail in [52]. In some elitist MOEAs, the strategy of elitism is combined with the maintenance of an 'external population' of solutions that are nondominated among all those found so far. Several early schemes are discussed in [112] but the first elitist MOEA paper to be published in the mainstream evolutionary computation literature was [94]. In this work, Parks and Miller describe a MOEA that maintains an 'archive' of nondominated solutions, similar to a store of all nondominated solutions evaluated, but limited in size: members of the main population only enter the archive if they are sufficiently dissimilar from any already stored. Reproductive selection takes parents from both the main population and the archive. The authors investigate the effects of different degrees of selection from each, and also different strategies for selecting from amongst the archive, including how long individuals have remained there.

At around the same time Zitzler and Thiele proposed what is to date one of the most popular of all MOEAs: the strength Pareto EA (SPEA) [113]. It uses two populations: an internal population, and an external population consisting of a limited number of nondominated solutions. In each generation, the external population is updated by two processes: addition of new nondominated individuals coming from the internal population (with removal of any solutions that consequently become dominated); and removal of solutions by objective-space clustering, to maintain a bound on the population's size. The new internal population is then generated by selection from the union of the two populations, and then by applying variation operators. The novelty, and perhaps the efficacy, of SPEA derives from the way the internal and external population interact in the fitness assignment step. In this, each external population member is first awarded a *strength*, proportional to the number of internal population members it dominates. Then each internal population member is assigned a dummy fitness based on the sum of the strengths of the external population members that dominate it. Binary tournament selection with replacement is used based on the dummy fitness/strengths of the combined populations. This fitness assignment strategy is a co-evolutionary approach between two distinct populations and its purpose is to bias selection towards individuals with a lower dominance rank *and* that inhabit relatively unpopulated 'niches'. The niches in SPEA are governed by the position of the nondominated individuals, and these are clustered so should themselves be well-distributed.

Numerous other elitist MOEAs exist in the literature, offering slightly differing ways of assigning fitness, choosing from a main population and an archive, and encouraging or preserving diversity. Regarding the latter, a trend towards self-adaptive niching (see Figure 5) has established itself with SPEA, PAES [73], NSGA-II [26], and PESA [20], amongst others, to avoid the necessity of setting niche sizes in the objective space, a problem with early algorithms such as MOGA and NSGA. Control of the degree of elitism has

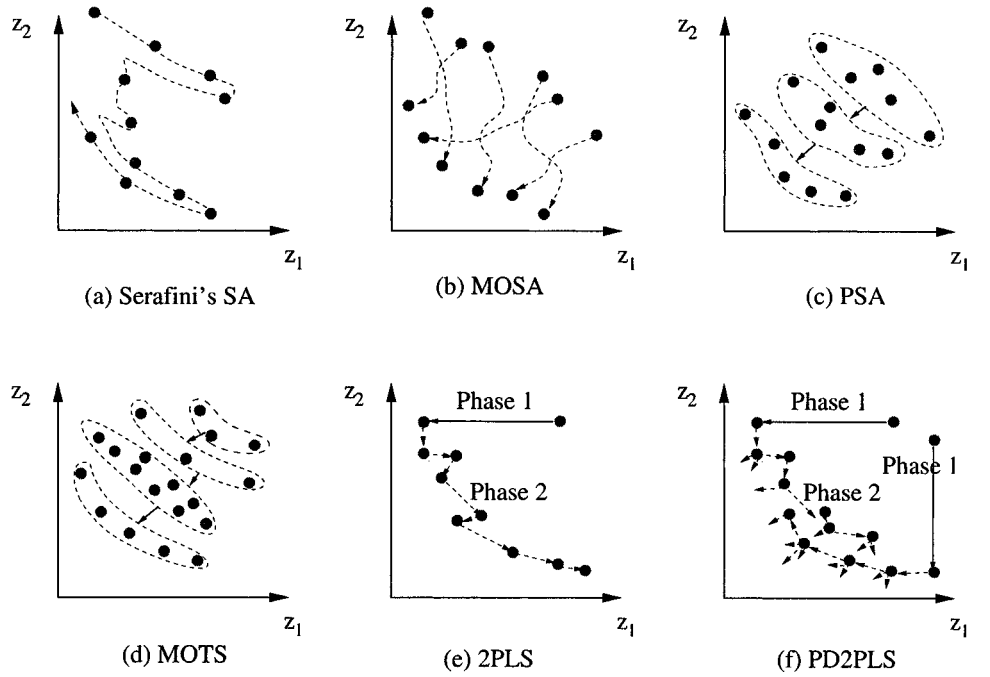


Fig. 6. Schematics depicting the different strategies employed by different local search metaheuristics, as described in section 3.3

also been investigated, e.g. in [27], and there has also been a trend towards lower computationally complexity, as evidenced by PAES, NSGA-II and the micro-GA [17]. More efficient data structures for ranking and niching available now [64] should make the current breed of elitist MOEAs a good starting point for designing good MAs for MOO.

2.6 Local search algorithms using scalarizing functions

One of the earliest papers on local-search metaheuristics for MOO is [99], which proposes and investigates modifications to simulated annealing in order to tackle the multiobjective case. A number of alternative acceptance criteria are considered, including those based on Pareto dominance, but the preferred strategy combines two weight-based scalarizing functions. In order to sample different Pareto optima during one run of the algorithm, the weights for each objective are slowly modified, at each fixed temperature, using a purely random (non-adaptive) scheme.

The MOSA method [107] follows Serafini regarding the modification of the SA acceptance function, but uses a different approach to building up the

approximation to the Pareto set. Where Serafini's approach varied the weights of the scalarizing function as the cooling occurred, MOSA method works by executing (effectively) separate runs of SA, each run using its own unique weight vector, and archiving all of the nondominated solutions found.

A population-based version of Serafini's SA is proposed and tested in [23]. The Pareto simulated annealing algorithm, PSA, performs each step of the SA algorithm 'in parallel' on each independent member of a population (N.B. the members are not in competition: there is no selection step), and each member carries with it its own weighting vector. Of particular note is the fact that the members of the population co-operate through an innovative adaptive scheme for setting their individual objective weights, in order to achieve a good distribution of solutions in the objective space. In this scheme, each member of the population continually adjusts its own weight vector to encourage it to move away from the nearest neighbour solution in the objective space.

These three SA algorithms, Serafini's SA, MOSA method, and PSA, illustrate three different ways to organize the building up of a Pareto front, respectively: (1) use a single solution and improve it, letting it drift up and down the PF via the use of randomly changing scalarizing weights; (2) use separate, independent runs and improve a single solution towards the PF, each run using a unique direction; (3) use a population of solutions and try to improve them all in parallel, at the same time encouraging them to spread out in the objective space. These alternatives are illustrated respectively in Figure 6 (a),(b),(c).

The idea of adaptively setting the weight vectors of individuals in a population, as used in PSA, is also used and extended in a tabu search algorithm, called MOTS [49]. In this, an initial population of points is improved in parallel, much as in PSA, but using a tabu search acceptance criterion. MOTS has another notable feature of particular relevance to MA design: it uses an adaptive population size based on the current nondominance rank of each member of the population. When the average of this rank is very low, it indicates that the members of the population are already well-spread (since few dominate each other), so the population size is increased in order to be able to cover more of the Pareto front. If the rank becomes too high this indicates that solutions are overlapping each other in objective space, and hence the population size is decreased—see figure 6(d).

Most recently, [90] describes a generic local search-based procedure for bi-objective problems, the two-phase local search (2PLS). In this approach the so-called 'first phase' applies local search to the problem, considering only one objective in isolation. When a good local optimum has been found, the 'second phase' begins. It uses the previous good solution as a starting solution for a new local search based on a scalarizing of the two objectives. Once a good solution has been found, the weights of scalarization are adjusted and the LS is again applied, again using the previous solution as a starting solution. Thus, a 'chain' of LS runs is applied, until a specified number of weights has been completed and the algorithm terminates (figure 6(e)). Depending

on the problem, the weights may be adjusted gradually or randomly. For the multiobjective TSP it is shown that gradual changes in the weights leads to good performance. In a slight variation to the algorithm, called the Pareto double two phase local search (PD2PLS), two first phases are used, one for each objective, and subsequently the best solution returned by each LS run is augmented using a search for nondominated solutions in its neighbourhood (figure 6(f)). This increases the number of nondominated solutions found by the algorithm with little overhead in time. Overall, the 2PLS and PD2PLS algorithms exhibit high performance on benchmark multiobjective combinatorial optimization problems, and are thus worthy contenders as subroutines for use within an MA for MOO, although versions for more than two objectives are needed.

2.7 Model-based searchers using dominance ranking

Model-based search is a name for a class of algorithms that employ some kind of statistical model of the distribution of remembered good solutions in order to generate new solutions. They can be seen as a development of EAs, in which recombination is replaced by a more statistically unbiased way of sampling from the components of known good solutions. Examples of model-based search algorithms are population-based incremental learning (PBIL), univariate distribution algorithms (UDAs), ant-colony optimization (ACO), Bayesian optimization algorithms (BOAs), and linkage-learning EAs. Recently a number of attempts at extending model-based search to the multiobjective case have been made, and like most MOEAs, they use the dominance ranking (see figure 4) to evaluate solution quality.

Straddling the middle-ground between a standard EA and a model-based search, the *messy genetic algorithm*, which attempts to learn explicit 'building blocks' for crossover to operate with, has been extended to the MOO case with the MOMGA and MOMGA-II algorithms [108, 115].

A step further away from standard, recognisable EAs, are algorithms that replace recombination altogether by using instead an explicit probability distribution over solution components, in order to generate new solutions. Several different attempts have been made at adapting Bayesian optimization algorithms (BOAs) and similar variants, to the multiobjective case [65, 80, 98, 106]. In the models proposed in [106], it is found that a factorization based on clusters in the objective space is necessary to obtain a good spread across the Pareto front. This results in an algorithm that is quite similar to the population-based ACO [47], described below, except that here the model is based only on the current population and not on a selection from a store of all nondominated solutions. The approach of [80] is a little different: instead of a mixture of clustered univariate distributions, a binary decision tree is used to model the conditional probabilities of good solution components. In order to encourage this model to generate sufficient diversity in the objective space, the selection step is based on ϵ-dominance [82] (see figure 5),

whereby solutions that are very similar tend to ϵ-dominate each other and will not be selected.

Ant colony optimization [30], is an agent-based search paradigm, particularly suited for constrained combinatorial optimization. Briefly, in this approach, candidate solutions are constructed 'component by component' by the choices made by 'ants' as they walk over a solution construction graph. At each step of a solution construction, the components available for the ants to select have associated with them a particular desirability, which biases the selection. This bias is mediated through the concentration of pheromone on the nodes or edges of the construction graph. In the usual implementations of ACO, the initially random pheromone levels change gradually via two processes: depositing of pheromone on the components making up a very good solution whenever one is found, and evaporation of pheromone, as a forgetting mechanism to remove the influence of older solutions. In population-based ACO, no evaporation is used, and instead a population of good solutions is always stored. Whenever a solution in the population is replaced by a new one, the pheromone trails associated with the old one are entirely removed from the construction graph, and the new member of the population deposits its pheromone instead. In [47], population-based ACO is adapted to the multiobjective case. This is achieved by making use of a store of all nondominated solutions found, and periodically choosing a subset of this to act as a temporary population. Promotion of diversity in the objective space is achieved in two ways: (1) the members of a temporary population are selected from the nondominated set based on their proximity to one another in the objective space (so there is a kind of restricted-mating or island-model effect); and (2) each objective has its own pheromone and the selection of components is governed by a weighted sum over the different pheromone levels—the weights being determined by the location, in objective space, of the current temporary population, relative to the entire nondominated set.

2.8 Algorithms using alternating objective functions

Schaffer is widely regarded as having started the field of evolutionary multiobjective optimization with his seminal paper on the vector evaluated genetic algorithm (VEGA) [97]. This was a true attempt at the evolution of multiple nondominated solutions concurrently in a single EA run, and the strategy was aimed at treating possibly non-commensurable objectives. Thus, aggregation of objectives was ruled out in favour of a selection procedure that treated each objective separately and alternately. As explained in [40], the approach is, however, *effectively* assigning reproduction opportunities (fitness) as a weighted linear function of the objective functions, albeit it implicitly adapts the weighting to favour the objective which is 'lagging' behind. This behaviour means that on problems with concave Pareto fronts, 'speciation' occurs, meaning that only solutions which do well on a single objective are found, while compromise or middling solutions do not tend to survive. Another early

approach, this time using evolution strategies (ESs) as the basis, was proposed by Kursawe [78]. The paper included some interesting early ideas about how to deal with non-commensurable objectives but the algorithm proposed has not been tested thoroughly to date.

Nearly ten years younger than the latter, [102], describes one of the first distributed EAs for MOO. It employs three separate but interacting populations: a main population and two islands, with the main population accepting immigrants from the islands. The performance of three strategies were compared. One strategy is to use homogeneous populations, each evolving individuals using the dominance ranking for fitness assignment. The second is to use heterogeneous islands, each evolving individuals to optimize a different objective, while the main population is still evolved using dominance ranking. The third is the same as the second but restarts are additionally used in the island populations. Testing on a number of scheduling problems revealed the latter to be consistently and significantly the most effective and efficient of the three strategies.

Gambardella*et al.* use a similar kind of heterogeneous, co-operative approach in their ant-colony optimization algorithm for a vehicle routing problem [42]. The problem tackled has two objectives: to minimize the number of vehicles needed to visit a set of customers with particular time window constraints; and to minimize the total time to complete the visits. To achieve this, two separate ant colonies work pseudo-independently and in parallel. Starting from a heuristically generated feasible solution, one colony attempts to minimize the number of constraint violations when one fewer vehicle is used than in the current best solution, while the other colony attempts to reduce the total time, given the current best number of vehicles. Feasible improvements made by either colony are used to update the current best solution (which is used by both colonies to direct construction of candidate solutions). In the case that the colony using one fewer vehicles finds a feasible solution, both ant colonies are restarted from scratch, with the reduced number of vehicles.

2.9 Other approaches

One MOO approach which stands very much on its own is a method proposed in [37]. The originality of the approach lies in the way the whole multiobjective optimization problem is viewed. In every other approach outlined above, whether it be population-based, model-based, or a local search, it is individual solutions that are evaluated, and the fitter ones somehow utilised. By contrast, [37] proposes evaluating the whole current population of solutions *in toto* and using this scalar quantity in an acceptance function. For example, simulated annealing in this scheme would work by applying some measure (and Fleischer proposes the Lebesgue integral of the dominated region – see figure 7) over a population of current solutions. When a neighbour solution of one of the population is generated, it is accepted modulo the change in the Lebesgue measure of the *whole* population. Fleischer points out that since the

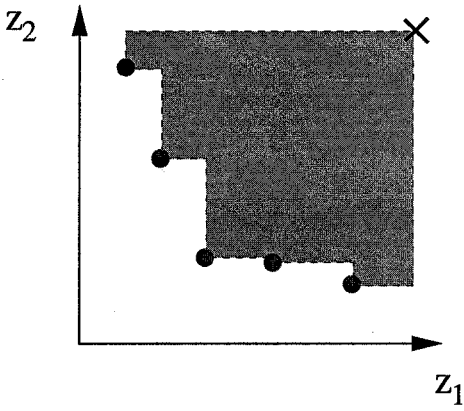


Fig. 7. The Lebesgue measure (or $\mathcal{S}$ metric) of a nondominated approximation set is a measure of the hypervolume dominated by it (shaded region), with respect to some bounding point (here shown by an X). The maximum of the Lebesgue measure corresponds to the Pareto front

maximum of the Lebesgue integral is the Pareto optimal set (provided the number of solutions is large enough), a simulated annealing (for example) optimizing this measure provably converges in probability to the Pareto optimal set.

3 Going Further: Issues and Methods

We have seen in the last section a variety of metaheuristic approaches to MOO, illustrating some of the basic principles of how to assign fitness and maintain diverse 'populations' of solutions. These are the basic pre-requisites for MAs for MOO, however a number of further issues present themselves. In this section we briefly discuss the current thinking on some of these other issues.

3.1 Performance measures in MOO

If one is developing or using an algorithm for optimization it almost goes without saying that there should be some way to measure its performance. Indeed, if we are to compare algorithms and improve them we really must first be able to define some means of assessing them. In single-objective optimization it is a relatively simple case of measuring the quality of solution obtained in fixed time, or alternatively the time taken to obtain fixed quality

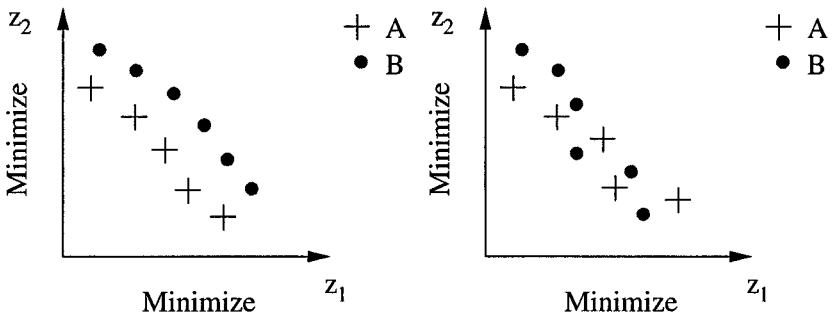


Fig. 8. On the left, two sets A and B, where A outperforms B, since every vector in B is dominated by at least one in A. On the right, two sets that are incomparable—neither is better under the minimal assumptions of Pareto optimization

– and quality and time can themselves be defined unequivocally in some convenient way. In MOO the situation is the same regarding the time aspect of performance assessment but the quality aspect is clearly more difficult. Recall that the standard goal of MOO (as far as we are concerned) is to approximate the true Pareto optimal set, and hence the outcome of the search is not one best solution, but a set of solutions, each of which has not one, but multiple dimensions of quality. We call these approximation sets, and it is clear that approximation sets cannot be *totally* ordered by quality, (see figure 8), if we remain loyal to the minimal assumptions of Pareto optimization. Nonetheless, a *partial* order of all approximation sets does exist, so it is possible to say that one set is better than another for some pairs, while others are *incomparable.*

The partial ordering of approximation sets is sometimes unsatisfying because it, of itself, does not enable an approximation set to be evaluated in isolation. For this reason, practitioners sometimes (often implicitly) adopt an ad hoc definition of a 'good approximation set' as one exhibiting one or more of: *proximity* to the true PF; *extent* in the objective space; and a good or even *distribution*—and use measures for evaluating these properties. The problem with such an approach (if not done with great care and thought) is that these measures can conflict utterly with the stated goal of approximating the PF. This problem is illustrated in Figure 9.

If one wants to really do Pareto optimization, and needs a unary measure of approximation set quality, the fact that there is a true partial ordering of all approximation sets (under Pareto optimization assumptions) demands that good or reliable measures of quality respect this ordering in some way. Using this fact, it is possible to assess how useful and reliable are different potential measures of approximation set quality. If a measure can judge an approximation set A to be better than B, when the converse is true, for some pair of sets A and B, then the measure is, in a sense, unreliable and fairly useless. On the other hand if a measure never states that A is better

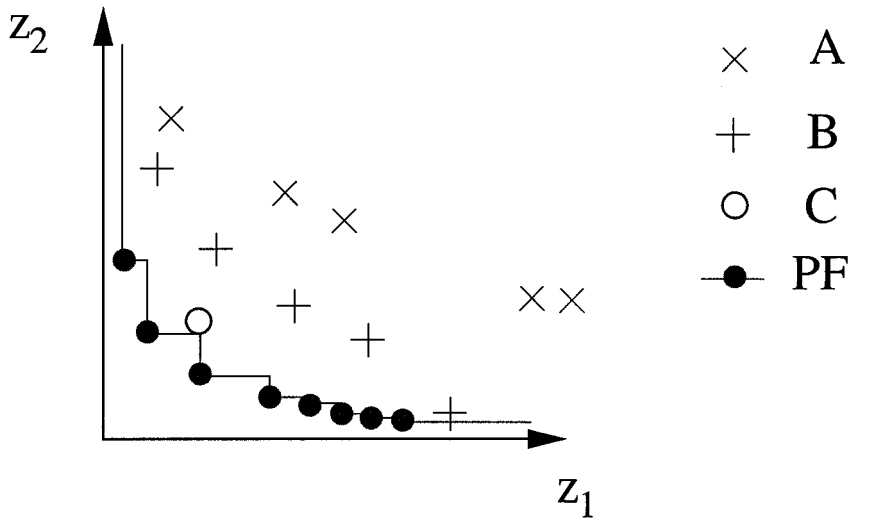


Fig. 9. A Pareto front and three approximation sets, A, B, and C. Depending on the measure used A, B, or C might be considered the best, and even better than the PF! If a measure of 'well-distributednes' is used B is best, better even than the PF. If a measure of proximity to the PF is used, C is best, even though B is just as close in parts, and if extent in objective space is measured, A is best

than B when the reverse is true, then it may be of some use, even if it does not detect all positive cases. More useful still, is if it always detects positive cases correctly but sometimes judges one set better when they are, in fact, incomparable. The ideal situation is when a measure always detects A better than B when it is the case, and never gives a false positive, when it is not.

A plethora of different measures for overall or specific aspects of MOO performance are described in the EMOO literature but it is not until relatively recently that some researchers have begun to critically assess them. Most notably, Zitzler *et al.* [114] give an extensive treatment of performance measures in MOO, using a framework that formalizes the notion of respecting the true partial ordering of approximation sets, as described above. A key result of [114] is that no unary measure (i.e. one taking just one approximation set as input, and returning a number or vector of numbers) whatever, including any finite combination of unary measures, can detect reliably when one approximation set is better than another, without giving false positives. Such results underline the necessity of thinking very hard before selecting measures to evaluate performance. For earlier work on the same issue, see also [50, 68, 72].

3.2 Archiving, multi-populations, and convergence

In the tour of metaheuristics of section 3 we saw some examples of algorithms using secondary populations, archives, and/or populations of non-fixed size. The use of such mechanisms seems to be a necessary element of more advanced methods for MOO, which aim to build up and store a good approximation to the PF. Some of the options for incorporating these elements within existing algorithms are summarised below:

- Use a main population only: the population is the store (NPGA [53], NSGA [100], NSGA-II [26])
- Use a single-point local search, but keep separately a bounded-sized archive of nondominated points found (PAES [73])
- Use a main and a secondary population - both of fixed size (SPEA [113], PESA [20])
- Use a main population of fixed size and an archive of unbounded size [36] (RD-MOGLS [60])
- Use multiple populations as in an island model [66]
- Have a dynamic main population size [105] (MOTS [50])

With the use of an (unbounded) archive of solutions, an algorithm can potentially converge to the (entire) Pareto front. Thus, convergence proofs for MOEAs now exist in the literature: Rudolph [96] proved convergence to the Pareto front (that is at least one Pareto optimal solution) for some simple multiobjective EAs, and it is possible to prove that the entire Pareto front can be enumerated provided an unbounded archive is available. More realistically, archives should be bounded in some way. A number of more recent papers have been written regarding what can be proved with respect to the convergence properties of such bounded archives [71, 70, 74, 82, 81]. Research in this area is still needed and the issue of which solutions to store during the search in order to converge towards a 'good approximation set' remains an open question.

3.3 Test functions, problems, and landscapes

Early test functions in the EMOO literature comprised a number of ad hoc, low-dimensional functions, enabling a proof-of-concept for early MOEAs, but nothing more. A large step toward a more scientific approach was taken with the introduction, by Deb [25], of a framework for constructing functions with identifiable features such as concave, discontinuous, and non-uniform PFs, local optima, and deception. A suite of six 2-objective functions derived from these became popular for some time, despite some drawbacks. Deb later extended the framework in [29] to allow scalable functions of any number of objectives to be generated and this is becoming more popular for testing now.

There is still a lack of understanding of the relationships between the properties of problems, their 'landscapes', and strategies for search. A couple of studies that have begun investigating this with respect to combinatorial

problems are [9] and [69]. [9] investigates 'global convexity' in multiobjective TSP problems and finds that optima nearby to each other on the PF are quite similar, a finding that suggests restricted mating in MOEAs and strategies for chaining local searches along the PF, like the 2PLS [90], would be effective on this problem. [69] introduces a tunable multiobjective version of the QAP problem and proposes some techniques for characterising the landscapes of instances of this problem. The latter is ongoing work.

3.4 Not quite Pareto

In much of the above discussion we have explicitly stated that finding a good approximation to the entire Pareto front is the goal of MOO, as far as we are concerned. Nonetheless, several situations arise when finding the whole Pareto front may not be desirable, and yet finding a single solution, as would be obtained by transforming the problem into a single objective, would not be adequate either. In particular, when the number of objectives is much above two or three, the size of the Pareto optimal set may be very large, necessitating a more restrictive notion of optimal. In these cases, some kind of 'middle ground' may be the best option, in which some Pareto optima may be treated as more desirable than others. One of the seminal papers on non-Pareto approaches is [7], which proposes a number of alternative ranking policies for use in EA selection. Other more recent policies are described in [10, 34], and in [31] where the concept of the order relation, 'favour', is introduced. Where more explicit preferences of a decision maker are available more advanced methods may be used, as in, for example, [22].

4 MAs for MOO: the fossil record

The extensive array of existing metaheuristics, issues and methods reviewed in the sections above gives a richer basis from which to design new MAs than do the existing MAs for MOO themselves. Nonetheless, before outlining some principles and ideas for new MAs, it is worth reviewing the few multiobjective MAs described in the current literature.

Arguably, it is just three separate groups of authors that are responsible for much of the small multiobjective MA literature, each group having written several papers. A small number of others have published more isolated works and these tend to be application-based rather than aiming at developing general algorithms. The three main groups are: Ishibuchi and Murata, who proposed a 'multiobjective genetic local search' (MOGLS) [54] algorithm in 1996; Jaszkiewicz, who proposed an algorithm initially called RD-MOGLS for 'random directions' MOGLS [60], and a slight variant called the Pareto memetic algorithm (PMA) [61]; and Knowles and Corne, who developed an algorithm called M-PAES [67]. In all of these algorithms, the basic idea is simple: a local search is applied to every new offspring generated (by crossover

or mutation), and the improved offspring then competes with the population for survival to the next generation (Lamarckianism). In all cases, only one local search operator is available and there has been no work on mechanisms for deciding whether or not to apply a local search to an offspring. The algorithms of Ishibuchi and Murata and Jaszkiewicz are quite similar in other respects too: both use randomly-drawn scalarizing functions to assign fitness for parent selection and in the local search. The algorithm of Jaszkiewicz uses an unbounded 'current set' of solutions, CS, and from this selects a small 'temporary population', TP that comprises the best solutions on the incumbent scalarizing function. It is then TP that is used to generate offspring by crossover. Some results put forward by Jaszkiewicz suggest that scalarizing functions are particularly better at encouraging diversity than dominance ranking methods used in most EAs. Ishibuchi and Murata have also made a number of interesting studies on their algorithm over the years, investigating restricted mating and other innovations, and have tested it on several problems [55, 86, 87, 88, 58, 57, 56, 89]. Knowles and Corne's M-PAES algorithm is quite different in at least one respect from the other two: it does not use scalarizing functions at all, either in the local search or the parental selection, employing instead a form of Pareto ranking based on comparing solutions to an archive of nondominated solutions. This may perhaps make it slower when very fast local search heuristics are available because the comparison of solutions takes longer to operate than applying a scalar acceptance function. On the other hand, whereas the MOGLS algorithms will discard newly generated nondominated solutions if they are poor on the incumbent scalarizing function, this will not happen in M-PAES, making it potentially more parsimonious of function evaluations—an advantage when these are more costly.

Of the more isolated papers, a few stand out for their interesting ideas or applications. In [43] the idea of using supported solutions (figure 9) to seed an EA is proposed. That is, on problems where some exact algorithm for computing supported Pareto optima is available, [43] proposes a two-phase hybrid approach where the exact algorithm is applied first, then an EA is used to search for the non-supported Pareto optima, which cannot be found using the exact heuristic.

Another kind of two-phase approach is described in [103]. The proposed procedure is as follows: run an MOEA for a fixed number of generations; then for each Pareto optimal solution, compute the neighbourhood and store any nondominated solutions found; update the list of PO solutions and again recompute all the neighbourhoods; iterate the procedure until no improvement occurs.

Similarly to [103], [27] proposes to run an EA (NSGA-II) and then apply local search afterwards to improve the Pareto optimal set. To do this, the authors apply a local search using a weighted sum of objectives. The weights used are computed for each solution based on its location in the Pareto front such that the direction of improvement is roughly in the direction perpendic-

ular to the PF. Nondominated solutions are then identified and clustering is finally applied to reduce the number of solutions returned.

Finally, worth mentioning because the results on a well-known application problem are apparently good, is [1]. This paper introduces a hybrid of differential evolution [101] and backpropagation learning in order to evolve both the architecture and weights of an artificial neural network. Two objectives are minimized, the summed squared error in training, and the number of neural units in the network. Abbass reports good reports on the Australian Credit Card and Diabetes Data sets.

5 Recommendations for MA design and practice

In the previous sections we have reviewed current MOO practices: we revisited a swathe of metaheuristics, considered some of the most salient issues and results, and looked briefly at some existing MAs. We now consider how we should draw on this background to build a more 'memetic' MA for MOO.

In recent years, Moscato and Krasnogor have provided a guiding manifesto for putting the 'memetic' back in memetic algorithms [77, 85] advocating, in particular, the use of multiple memes: memeplexes. These are collections of ways of learning or adapting which can be transmitted at different levels and through different processes. For example, multiple local searches, multiple recombination operators, and so on could co-exist in a single algorithm, that then learns, at both the individual and the population level, which operators to use, and when, depending on the monitoring of internal processes at the level of the individual or population. The MAs that we have seen in the MOO literature to date are relatively poor images of these 'fully-fledged' MAs.

In Algorithm 1, we put forward a simple framework that could serve as a guide for making a more memetic MA for MOO. In line 1, a population P of solutions is initialized. As usual, this procedure may be simply random or it may employ some heuristic(s). Line 2 sets the archive A to the nondominated solutions from P. Thereafter, the main loop of the MA begins. Line 4 sets up an inner loop in which a stagnation criterion is checked. This should be based on some memeplex which monitors progress in diversity, proximity, and/or some other criteria. Lines 5–9 give a very high level description of the update of the population and archive. Five different 'schedulers' are employed, basically corresponding to mating selection, reproduction, lifetime learning, survival selection, and update of the archive, respectively. Each scheduler chooses from a memeplex of operators, based on estimates of the current success of those operators. E.g., in line 5, SelectFrom is the operation of mating selection, the domain of which is the union of the population and archive, and co-domain is a child population C; the selection is controlled by the scheduler, sel_sched, which uses a success measure, succ, to choose one operator from the set, *SEL*, of currently available operators for selection. Notice that P and A are potentially of variable size, in this scheme. In line 11, the population P is

```
Multi-Objective MA():
Begin
  P := Initialize(P);
  A := Nondom(P);
  While ( stop_criterion not satisfied ) Do
    While ( stagnation_criterion not satisfied ) Do
      C := SelectFrom(P ∪ A, sel_sched(succ(SEL)));
      C' := Vary(C, var_sched(succ(VAR)));
      C'' := LocalSearch(C', ls_sched(succ(LS)));
      P := Replace(P ∪ C'', rep_sched(succ(REP)));
      A := Reduce(Nondom(A ∪ P), red_sched(succ(RED)));
    endDo
    P := RandomImmigrants(P, imm_sched(succ(IMM)));
  endDo
  return (A);
End.
```

Fig. 10. Candidate MA framework for MOO

updated using some immigration policy to release it from stagnation. The archive of nondominated solutions is returned in line 13.

The framework proposed is rather broad and actually instantiating it requires us to consider how we should resolve many choices, including those considered in the following sections, at the very least. Table 1 summarises some of the MA elements/configuration choices to consider.

5.1 Desired outcomes and prevailing conditions

As in any other optimization scenario, we should know at the outset what is a desirable outcome, how this can be measured, and what are the prevailing conditions under which the search is going to take place.

One important factor in MOO is knowing how many solutions are desired. The answer could be as many as possible, an exact number, or could be expressed in terms of some resolution at which the PF is sampled. These considerations might affect different options for storing the nondominated solutions (see Table 1, question 1).

The dimensionality of the objective space is another important factor and how this is going to be dealt with. If there are only two or three objectives then there is some evidence that dominance-ranking-based selection methods may be the most appropriate, assuming a good approximation to the Pareto front is desired, with no particular preference for either diversity or proximity. On the other hand, if the number of objectives is high, Pareto selection may be problematic, because many solutions will be incomparable. There

Table 1. Some suggestions for configuring an MA design

Question	Answer	Choices
1. How many solutions are desired?		
	precisely N:	Use Lebesgue archiving [75] or adaptive grid archiving [70]
	as many as possible:	Use an unbounded archive [36]
	require ϵ-approx set:	Use ϵ-Pareto archiving [82]
2. Is diversity more or less important than proximity?		
	More:	Use scalarizing functions; optimize diversity only
	Less:	Use strong elitist selection; dominance ranking approach;
	No preference:	Combine Pareto approach with scalarizing methods; monitor progress using an overall unary measure like the $\mathcal{S}$ metric
3. What is the dimensionality of the objective space?		
	1-d:	Consider 'multi-objectivizing' [76, 63]
	2-d or 3-d:	Use Pareto-ranking approaches
	4-d+:	Use the order relation *favour* [31], or preference methods [22], to reduce the number of effective optima; consider aggregating correlated objectives
4. How long does function evaluation take?		
	Minutes-to-days:	Use Bayesian approach [80] , or other computationally intensive model-based methods
	Seconds:	Use self-adaptation, other medium-overhead methods
	Microseconds:	Rely on fast LS strategies [90]
5. Is the true Pareto front known?		
	Yes:	Use epsilon-measure to compute progress/measure overall performance [114]
	No:	Use $\mathcal{S}$ measure to compute progress/measure overall performance
6. Are supported solutions available?		
	Yes:	Seed the MA with them and try to find the non-supported solutions [43]

are various alternatives to consider: using the relation *favour* [31], instead of component-wise order; using some other aggregating methods as proposed in [7]; or, actually aggregating some of the objectives together following a correlation/mutual information analysis.

5.2 Methods for monitoring progress

The MA framework proposed above requires that operators and procedures be selected based on their current success rates. These, in turn, must be estimated by some notion of progress. How should this progress be measured? Deb has proposed a number of running time metrics in [28] and Zitzler has advocated using the $\mathcal{S}$ metric [112] (see figure 7) to detect convergence. We have not seen much in the way of statistics for detecting or measuring the success of particular operators so far but these could be adapted from similar measures used in EAs.

5.3 System- and self-adaptation

Adaptation of mutation rates, crossover probabilities, and so forth is a topic that has received significant attention in the EAs literature over the years (see [33] for an extensive review). By comparison, the topic of self-adaptation in MOO, is surprisingly under-developed. Where it has been used, as we saw earlier, is in the control of the search 'direction' in the objective space: i.e. to direct the search towards sparsely populated areas of the Pareto front. Thus, it is usually some kind of weighting vector adaptation. The potential in MOO for self-adaptation is large, however, and should be part of any 'real' memetic approach. The adaptation of selection pressure/elitism maybe of particular importance, since we would expect that the different stages of building up a Pareto front might demand more or less aggressive searches. Getting to a local Pareto front quickly may demand aggression, whereas stagnation there might suggest decreasing the selection pressure in order to spread out along it, or hopefully find a route to a better front.

Some attempts have been made at adapting population size to the size of the Pareto front – whether that be the archive population or the main one. E.g., as we saw in MOTS, the mutual dominance of the population was used to adapt the size of the population. In the work of Laumanns, it is by setting a desired level of approximation that the archive's size is controlled, so that an appropriate number of solutions is maintained. These methods seem to be going in the right direction, as the use of a fixed population size, when trying to search efficiently a multi-dimensional objective space would seem to be too restrictive.

5.4 Controlling the overall search

Let us assume that we are interested in maximizing the rate at which the S measure increases – that is our gold standard of progress. Then, we could have

a number of overall search strategies competing with each other in some form of bidding mechanism, where they each have wealth in proportion to a record of the prior rates of progress they achieved when in control of the search. We could have, for example, the following two search strategies:

1. A PLS-like strategy that applies a local search repeatedly in one direction, using an aggregation function, until some convergence criterion is fulfilled. After this, a nearby weight is chosen and the same solution is once again improved. All nondominated solutions are stored in an archive.
2. A PESA-like strategy in which a whole front of nondominated solutions is used to generate new solutions, generation by generation, via recombination and variation, with selection based on crowding.

One strategy could be chosen at random to start with. After each 'generation' the strategy of choice could be reviewed. However, changing a strategy could be tabu for some time immediately after a change, in order to give it a chance and for decent statistics on it to be collected. Noticeable drops in progress rate could invoke a change in the current strategy in use.

Much work is needed to investigate if advanced 'multi-meme' approaches like this illustrative example really could provide robustness over different types of landscapes arising from different problems, or indeed within a single problem. It is not clear, even from the existing single objective literature, that this kind of high-level adaptation is really beneficial, but the time is perhaps overdue for us to try and find out.

6 Future Prospects

What does the future of multiobjective MAs hold and what are the most promising avenues to investigate now? In this article we have tried to distil a rich soup of ideas from the ever-growing literature on multiobjective metaheuristics, and a little on MAs, in order to provide some basis for the generation of new, more advanced algorithms. Many of the basics will probably remain the same: solutions will be evaluated by a combination of Pareto ranking-type methods and scalarizing methods; diversity will be encouraged using niching and crowding in parameter and objective space, and by the controlled use of different weights in the scalarizing functions. However, there is great scope for building more advanced and more memetic algorithms. In particular, it seems that the need, unique to MOO, to obtain and maintain a diverse pool of different solutions, suggests that such things as adaptive population sizes, multi-populations, and combinations of local and global search are especially relevant.

Expanding on this, and looking into the near future, we see that there is potential for more investigation as to the effects of restricted mating schemes, and how the success of these relates to features of the underlying problem

and/or multiobjective landscapes. For problems with a large number of objectives, new non-Pareto methods for ranking solutions need more investigation, as do methods for analysing correlations between objectives, perhaps to combine some objectives together; and conversely, we have seen some evidence in the recent literature [76, 63] that even single-objective problems may be tackled more effectively using multiobjective methods – work which merits further attention.

We have also provided in this article a glimpse of the different possible routes to building up a Pareto front employed by different multiobjective algorithms and have hinted at ways that these different overall strategies could be combined together in self-adaptive strategies that are sensitive to the progress being made in the search. This area, we think, is most promising.

New, advanced data structures for the storage and retrieval of Pareto optima [64] may offer increased speed of MAs and EAs which will, if developed further, enable exact solutions to be found even in relatively large solution spaces, provided fast local searches and evaluation functions are available.

And at the other end of the spectrum, where the evaluation of a solution takes a relatively long time, the recent advanced methods in model-based search promise a more principled way of sampling the search space. We have yet to see how these could be combined with local searches and other heuristics to build advanced MAs for these tough problems but the future is certainly exciting.

References

1. Hussein A. Abbass. A Memetic Pareto Evolutionary Approach to Artificial Neural Networks. In *The Australian Joint Conference on Artificial Intelligence*, pages 1–12, Adelaide, Australia, December 2001. Springer. Lecture Notes in Artificial Intelligence Vol. 2256.
2. M. F. Abbod, D. A. Linkens, and M. Mahfouf. Multi-Objective Genetic Optimization for Self-Organizing Fuzzy Logic Control. In *Proceedings of UKACC Control'98*, pages 1575–1580, University of Wales Swansea, UK, september 1998. IEE.
3. M.A. Abido. A new multiobjective evolutionary algorithm for environmental/economic power dispatch. In *Power Engineering Society Summer Meeting*, volume 2, pages 1263–1268. IEEE, 2001.
4. Antonino Augugliaro, Luigi Dusonchet, and Eleonora Riva Sanseverino. Evolving non-dominated solutions in multiobjective service restoration for automated distribution networks. *Electric Power Systems Research*, 59(3):185–195, October 2001.
5. Richard Balling and Scott Wilson. The Maximim Fitness Function for Multiobjective Evolutionary Computation: Application to City Planning. In Lee Spector, Erik D. Goodman, Annie Wu, W.B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Com-*

putation Conference (GECCO'2001), pages 1079–1084, San Francisco, California, 2001. Morgan Kaufmann Publishers.
6. Benjamín Barán, José Vallejos, Rodrigo Ramos, and Ubaldo Fernández. Reactive Power Compensation using A Multi-objective Evolutionary Algorithm. In *IEEE Porto Power Tech Proceedings*, volume 2, pages 6–11, Porto, Portugal, September 2001. IEEE.
7. P. J. Bentley and J. P. Wakefield. Finding Acceptable Solutions in the Pareto-Optimal Range using Multiobjective Genetic Algorithms. In P. K. Chawdhry, R. Roy, and R. K. Pant, editors, *Soft Computing in Engineering Design and Manufacturing*, Part 5, pages 231–240, London, June 1997. Springer Verlag London Limited. (Presented at the 2nd On-line World Conference on Soft Computing in Design and Manufacturing (WSC2)).
8. To Thanh Binh and Urlich Korn. Multicriteria control system design using an intelligent evolution strategy with dynamical constraints boundaries. In *Proceedings of the Conference for Control of Industrial Systems (CIS'97)*, volume 2, pages 242–247, Belfort, France, 1997.
9. Pedro Castro Borges and Michael Pilegaard Hansen. A basis for future successes in multiobjective combinatorial optimization. Technical Report IMM-REP-1998-8, Institute of Mathematical Modelling, Technical University of Denmark, March 1998.
10. Jürgen Branke, Thomas Kaußler, and Harmut Schmeck. Guidance in Evolutionary Multi-Objective Optimization. *Advances in Engineering Software*, 32:499–507, 2001.
11. Larry Bull and Matt Studley. Considerations of Multiple Objectives in Neural Learning Classifier Systems. In Juan Julián Merelo Guervós, Panagiotis Adamidis, Hans-Georg Beyer, José-Luis Fernández-Villaca nas, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature—PPSN VII*, pages 549–557, Granada, Spain, September 2002. Springer-Verlag. Lecture Notes in Computer Science No. 2439.
12. Edmund K. Burke, Patrick De Causmaecker, Sanja Petrovic, and Greet Vanden Berghe. A Multi Criteria Meta-heuristic Approach to Nurse Rostering. In *Congress on Evolutionary Computation (CEC'2002)*, volume 2, pages 1197–1202, Piscataway, New Jersey, May 2002. IEEE Service Center.
13. Donald H. Burn and Jeanne S. Yullanti. Waste-Load Allocation using Genetic Algorithms. *Journal of Water Resources Planning and Management*, 127(2):121–129, March-April 2001.
14. H. W. Chen and Ni-Bin Chang. Water pollution control in the river basin by fuzzy genetic algorithm-based multiobjective programming modeling. *Water Science and Technology*, 37(8):55–63, 1998.
15. Scott E. Cieniawski, J. W. Eheart, and S. Ranjithan. Using Genetic Algorithms to Solve a Multiobjective Groundwater Monitoring Problem. *Water Resources Research*, 31(2):399–409, February 1995.
16. Carlos A. Coello Coello and Carlos E. Mariano Romero. Evolutionary Algorithms and Multiple Objective Optimization. In Matthias Ehrgott and Xavier Gandibleux, editors, *Multiple Criteria Optimization: State of the Art Annotated Bibliographic Surveys*, pages 277–331. Kluwer Academic Publishers, Boston, 2002.
17. Carlos A. Coello Coello and Gregorio Toscano Pulido. A Micro-Genetic Algorithm for Multiobjective Optimization. In Eckart Zitzler, Kalyanmoy Deb,

Lothar Thiele, Carlos A. Coello Coello, and David Corne, editors, *First International Conference on Evolutionary Multi-Criterion Optimization*, pages 126–140. Springer-Verlag. Lecture Notes in Computer Science No. 1993, 2001.
18. D. Corne, M. Dorigo, and F. Glover. *New Ideas in Optimization*. McGraw-Hill, London, UK, 1999.
19. David W. Corne, Kalyanmoy Deb, Peter J. Fleming, and Joshua D. Knowles. The Good of the Many Outweights the Good of the One: Evolutionary Multi-Objective Optimization. *Connections. The Newsletter of the IEEE Neural Networks Society*, 1(1):9–13, February 2003.
20. David W. Corne and Joshua D. Knowles. The Pareto-envelope based selection algorithm for multiobjective optimization. In *Proceedings of the Sixth International Conference on Parallel Problem Solving from Nature (PPSN VI)*, pages 839–848, Berlin, 2000. Springer-Verlag.
21. David W. Corne and Joshua D. Knowles. No Free Lunch and Free Leftovers Theorems for Multiobjective Optimisation Problems. In Carlos M. Fonseca, Peter J. Fleming, Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele, editors, *Evolutionary Multi-Criterion Optimization. Second International Conference, EMO 2003*, pages 327–341, Faro, Portugal, April 2003. Springer. Lecture Notes in Computer Science. Volume 2632.
22. Dragan Cvetković and Ian C. Parmee. Use of Preferences for GA–based Multi–objective Optimisation. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *GECCO–99: Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1504–1509, Orlando, Florida, USA, 1999. Morgan Kaufmann Publishers.
23. P. Czyzak and A. Jaszkiewicz. Pareto simulated annealing—a metaheuristic technique for multiple-objective combinatorial optimization. *Journal of Multi-Criteria Decision Analysis*, 7:34–47, 1998.
24. N. V. Dakev, A. J. Chipperfield, J. F. Whidborne, and P. J. Fleming. An evolutionary algorithm approach for solving optimal control problems. In *Proceedings of the 13th International Federation of Automatic Control (IFAC) World Congress*, San Francisco, California, 1996.
25. Kalyanmoy Deb. Multi-Objective Genetic Algorithms: Problem Difficulties and Construction of Test Problems. *Evolutionary Computation*, 7(3):205–230, Fall 1999.
26. Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T Meyarivan. Fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *Parallel Problem Solving from Nature - PPSN VI : 6th International Conference Proceedings*, number 1917 in LNCS, pages 849–858. Springer, 2000.
27. Kalyanmoy Deb and Tushar Goel. Controlled Elitist Non-dominated Sorting Genetic Algorithms for Better Convergence. In Eckart Zitzler, Kalyanmoy Deb, Lothar Thiele, Carlos A. Coello Coello, and David Corne, editors, *First International Conference on Evolutionary Multi-Criterion Optimization*, pages 67–81. Springer-Verlag. Lecture Notes in Computer Science No. 1993, 2001.
28. Kalyanmoy Deb and Sachin Jain. Running performance metrics for evolutionary multi-objective optimization. Technical report, KANGAL, IIT Kanpur, India, May 2002.
29. Kalyanmoy Deb, Lothar Thiele, Marco Laumanns, and Eckart Zitzler. Scalable Multi-Objective Optimization Test Problems. In *Congress on Evolutionary*

Computation (CEC'2002), volume 1, pages 825–830, Piscataway, New Jersey, May 2002. IEEE Service Center.

30. M. Dorigo and G. Di Caro. The Ant Colony Optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*. McGraw-Hill, 1999.
31. Nicole Drechsler, Rolf Drechsler, and Bernd Becker. Multi-objective Optimisation Based on Relation *favour*. In Eckart Zitzler, Kalyanmoy Deb, Lothar Thiele, Carlos A. Coello Coello, and David Corne, editors, *First International Conference on Evolutionary Multi-Criterion Optimization*, pages 154–166. Springer-Verlag. Lecture Notes in Computer Science No. 1993, 2001.
32. Matthias Ehrgott and Xavier Gandibleux. A Survey and Annotated Bibliography of Multiobjective Combinatorial Optimization. *OR Spektrum*, 22:425–460, 2000.
33. Ágoston E. Eiben, Robert Hinterding, and Zbigniew Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3:124–141, 1999.
34. M. Farina and P. Amato. Fuzzy Optimality and Evolutionary Multiobjective Optimization. In Carlos M. Fonseca, Peter J. Fleming, Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele, editors, *Evolutionary Multi-Criterion Optimization. Second International Conference, EMO 2003*, pages 58–72, Faro, Portugal, April 2003. Springer. Lecture Notes in Computer Science. Volume 2632.
35. M. Farina, K. Deb, and P. Amato. Dynamic Multiobjective Optimization Problems: Test Cases, Approximation, and Applications. In Carlos M. Fonseca, Peter J. Fleming, Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele, editors, *Evolutionary Multi-Criterion Optimization. Second International Conference, EMO 2003*, pages 311–326, Faro, Portugal, April 2003. Springer. Lecture Notes in Computer Science. Volume 2632.
36. Jonathan E. Fieldsend, Richard M. Everson, and Sameer Singh. Using Unconstrained Elite Archives for Multiobjective Optimization. *IEEE Transactions on Evolutionary Computation*, 7(3):305–323, June 2003.
37. M. Fleischer. The Measure of Pareto Optima. Applications to Multi-objective Metaheuristics. In Carlos M. Fonseca, Peter J. Fleming, Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele, editors, *Evolutionary Multi-Criterion Optimization. Second International Conference, EMO 2003*, pages 519–533, Faro, Portugal, April 2003. Springer. Lecture Notes in Computer Science. Volume 2632.
38. Peter Fleming. Designing Control Systems with Multiple Objectives. In *IEE Colloquium on Advances in Control Technology*, pages 4/1–4/4, 1999.
39. Carlos M. Fonseca and Peter J. Fleming. Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 416–423, San Mateo, California, 1993. University of Illinois at Urbana-Champaign, Morgan Kauffman Publishers.
40. Carlos M. Fonseca and Peter J. Fleming. An Overview of Evolutionary Algorithms in Multiobjective Optimization. *Evolutionary Computation*, 3(1):1–16, Spring 1995.
41. C.M. Fonseca and P.J. Fleming. Multiobjective optimal controller design with genetic algorithms. In *International Conference on Control*, volume 1, pages 745–749, 1994.

42. Luca Maria Gambardella, Éric Taillard, and Giovanni Agazzi. MACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows. In David Corne, Marco Dorigo, and Fred Glover, editors, *New Ideas in Optimization*, pages 63–76. McGraw-Hill, 1999.
43. Xavier Gandibleux, Hiroyuki Morita, and Naoki Katoh. The Supported Solutions Used as a Genetic Information in a Population Heuristic. In Eckart Zitzler, Kalyanmoy Deb, Lothar Thiele, Carlos A. Coello Coello, and David Corne, editors, *First International Conference on Evolutionary Multi-Criterion Optimization*, pages 429–442. Springer-Verlag. Lecture Notes in Computer Science No. 1993, 2001.
44. David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1989.
45. D.E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms and their Applications*, pages 41–49. Lawrence Erlbaum, 1987.
46. Marc Gravel, Wilson L. Price, and Caroline Gagné. Scheduling Continuous Casting of Aluminum Using a Multiple-Objective Ant Colony Optimization Metaheuristic. Technical Report 2001–004, Faculté des Sciences de L'Administration, Université Laval, Québec, Canada, April 2001. Available at `http://www.fsa.ulaval.ca/rd`.
47. Michael Guntsch and Martin Middendorf. Solving Multi-criteria Optimization Problems with Population-Based ACO. In Carlos M. Fonseca, Peter J. Fleming, Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele, editors, *Evolutionary Multi-Criterion Optimization. Second International Conference, EMO 2003*, pages 464–478, Faro, Portugal, April 2003. Springer. Lecture Notes in Computer Science. Volume 2632.
48. D. Halhal, G. A. Walters, D. Ouazar, and D. A. Savic. Multi-objective improvement of water distribution systems using a structure messy genetic algorithm approach. *Journal of Water Resources Planning and Management ASCE*, 123(3):137–146, 1997.
49. Michael Pilegaard Hansen. Tabu Search in Multiobjective Optimisation : MOTS. In *Proceedings of the 13th International Conference on Multiple Criteria Decision Making (MCDM'97)*, Cape Town, South Africa, January 1997.
50. Michael Pilegaard Hansen and Andrzej Jaszkiewicz. Evaluating the quality of approximations to the non-dominated set. Technical Report IMM-REP-1998-7, Technical University of Denmark, March 1998.
51. M. Hapke, A. Jaszkiewicz, and R. Slowinski. Fuzzy multi-mode resource-constrained project scheduling with multiple objectives. In J. Weglarz, editor, *Recent Advances in Project Scheduling*, chapter 16, pages 355–382. Kluwer Academic Publishers, 1998.
52. Jeffrey Horn. Multicriterion Decision Making. In Thomas Bäck, David Fogel, and Zbigniew Michalewicz, editors, *Handbook of Evolutionary Computation*, volume 1, pages F1.9:1 – F1.9:15. IOP Publishing Ltd. and Oxford University Press, 1997.
53. Jeffrey Horn and Nicholas Nafpliotis. Multiobjective Optimization using the Niched Pareto Genetic Algorithm. Technical Report IlliGAl Report 93005, University of Illinois at Urbana-Champaign, Urbana, Illinois, USA, 1993.

54. Hisao Ishibuchi and Tadahiko Murata. Multi-Objective Genetic Local Search Algorithm. In Toshio Fukuda and Takeshi Furuhashi, editors, *Proceedings of the 1996 International Conference on Evolutionary Computation*, pages 119–124, Nagoya, Japan, 1996. IEEE.
55. Hisao Ishibuchi and Tadahiko Murata. Multi-Objective Genetic Local Search Algorithm and Its Application to Flowshop Scheduling. *IEEE Transactions on Systems, Man and Cybernetics*, 28(3):392–403, August 1998.
56. Hisao Ishibuchi and Youhei Shibata. An Empirical Study on the Effect of Mating Restriction on the Search Ability of EMO Algorithms. In Carlos M. Fonseca, Peter J. Fleming, Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele, editors, *Evolutionary Multi-Criterion Optimization. Second International Conference, EMO 2003*, pages 433–477, Faro, Portugal, April 2003. Springer. Lecture Notes in Computer Science. Volume 2632.
57. Hisao Ishibuchi and Tadashi Yoshida. Hybid Evolutionary Multi-Objective Optimization Algorithms. In A. Abraham, J. Ruiz del Solar, and M. Köppen, editors, *Soft Computing Systems: Design, Management and Applications (Frontiers in Artificial Intelligence and Applications, Volume 87)*, pages 163–172. IOS Press, ISBN 1-58603-297-6, 2002.
58. Hisao Ishibuchi, Tadashi Yoshida, and Tadahiko Murata. Selection of Initial Solutions for Local Search in Multiobjective Genetic Local Search. In *Congress on Evolutionary Computation (CEC'2002)*, volume 1, pages 950–955, Piscataway, New Jersey, May 2002. IEEE Service Center.
59. Hisao Ishibuchi, Tadashi Yoshida, and Tadahiko Murata. Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling. *IEEE Transactions on Evolutionary Computation*, 7(2):204–223, April 2003.
60. Andrzej Jaszkiewicz. Genetic local search for multiple objective combinatorial optimization. Technical Report RA-014/98, Institute of Computing Science, Poznan University of Technology, 1998.
61. Andrzej Jaszkiewicz. Do multiple-objective metaheuristics deliver on their promises? a computational experiment on the set-covering problem. *IEEE Transactions on Evolutionary Computation*, 7(2):133–143, April 2003.
62. Mikkel T. Jensen. *Robust and Flexible Scheduling with Evolutionary Computation.* PhD thesis, Department of Computer Science. University of Aarhus, Aarhus, Denmark, October 2001.
63. Mikkel T. Jensen. Guiding single-objective optimization using multi-objective methods. In *Applications of Evolutionary Computation*, volume 2611 of *LNCS*, pages 268–279. Springer, 2003.
64. Mikkel T. Jensen. Reducing the run-time complexity of multi-objective eas: The nsga-ii and other algorithms. *IEEE Transactions on Evolutionary Computation*, 7(5):502–515, 2003.
65. Nazan Khan, David E. Goldberg, and Martin Pelikan. Multi-Objective Bayesian Optimization Algorithm. In W.B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M.A. Potter, A.C. Schultz, J.F. Miller, E. Burke, and N. Jonoska, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2002)*, page 684, San Francisco, California, July 2002. Morgan Kaufmann Publishers.

66. Michael Kirley. MEA: A metapopulation evolutionary algorithm for multi-objective optimisation problems. In *Proceedings of the Congress on Evolutionary Computation 2001 (CEC'2001)*, volume 2, pages 949–956, Piscataway, New Jersey, May 2001. IEEE Service Center.
67. Joshua Knowles and David Corne. M-PAES: A Memetic Algorithm for Multiobjective Optimization. In *2000 Congress on Evolutionary Computation*, volume 1, pages 325–332, Piscataway, New Jersey, July 2000. IEEE Service Center.
68. Joshua Knowles and David Corne. On Metrics for Comparing Nondominated Sets. In *Congress on Evolutionary Computation (CEC'2002)*, volume 1, pages 711–716, Piscataway, New Jersey, May 2002. IEEE Service Center.
69. Joshua Knowles and David Corne. Towards Landscape Analyses to Inform the Design of Hybrid Local Search for the Multiobjective Quadratic Assignment Problem. In A. Abraham, J. Ruiz del Solar, and M. Köppen, editors, *Soft Computing Systems: Design, Management and Applications*, pages 271–279, Amsterdam, 2002. IOS Press. ISBN 1-58603-297-6.
70. Joshua Knowles and David Corne. Properties of an Adaptive Archiving Algorithm for Storing Nondominated Vectors. *IEEE Transactions on Evolutionary Computation*, 7(2):100–116, April 2003.
71. Joshua Knowles and David Corne. Bounded Pareto archiving: Theory and practice. In X. Gandibleux, M. Sevaux, K. Sörensen, and V. T'Kindt, editors, *Metaheuristics for Multiobjective Optimisation*, volume 535 of *Lecture Notes in Economics and Mathematical Systems*. Springer, January 2004. To appear.
72. Joshua D. Knowles. *Local-Search and Hybrid Evolutionary Algorithms for Pareto Optimization*. PhD thesis, University of Reading, UK, 2002.
73. Joshua D. Knowles and David W. Corne. Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy. *Evolutionary Computation*, 8(2):149–172, 2000.
74. Joshua D. Knowles, David W. Corne, and Mark Fleischer. Bounded archiving using the Lebesgue measure. In *Proceedings of the IEEE Congress on Evolutionary Computation*. IEEE Press, 2003. To appear.
75. Joshua D. Knowles, David W. Corne, and Mark Fleischer. Bounded archiving using the lebesgue measure. In *Proceedings of the IEEE Congress on Evolutionary Computation*. IEEE Press, 2003. (in press).
76. Joshua D. Knowles, Richard A. Watson, and David W. Corne. Reducing local optima in single-objective problems by multi-objectivization. In Eckart Zitzler, Kalyanmoy Deb, Lothar Thiele, Carlos A. Coello Coello, and David Corne, editors, *Evolutionary Multi-Criterion Optimization : first international conference; proceedings / EMO 2001*, volume 1993 of *LNCS*, pages 269–283. Springer, 2001.
77. Natalio Krasnogor. *Studies on the Theory and Design Space of Memetic Algorithms*. PhD thesis, University of Nottingham, 2002.
78. Frank Kursawe. A Variant of Evolution Strategies for Vector Optimization. In H. P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature. 1st Workshop, PPSN I*, volume 496 of *Lecture Notes in Computer Science Vol. 496*, pages 193–197, Berlin, Germany, October 1991. Springer-Verlag.
79. W. B. Langdon and P. C. Treleaven. Scheduling Maintenance of Electrical Power Transmission Networks Using Genetic Programming. In Kevin Warwick, Arthur Ekwue, and Raj Aggarwal, editors, *Artificial Intelligence Techniques in Power Systems*, chapter 10, pages 220–237. IEE, 1997.

80. Marco Laumanns and Jiri Ocenasek. Bayesian Optimization Algorithms for Multi-objective Optimization. In Juan Julián Merelo Guervós, Panagiotis Adamidis, Hans-Georg Beyer, José-Luis Fernández-Villaca nas, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature—PPSN VII*, pages 298–307, Granada, Spain, September 2002. Springer-Verlag. Lecture Notes in Computer Science No. 2439.
81. Marco Laumanns, Lothar Thiele, Kalyanmoy Deb, and Eckart Zitzler. Combining convergence and diversity in evolutionary multi-objective optimization. *Evolutionary Computation*, 10(3):263–282, Fall 2002.
82. Marco Laumanns, Lothar Thiele, Eckart Zitzler, and Kalyanmoy Deb. Archiving with Guaranteed Convergence and Diversity in Multi-Objective Optimization. In W.B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M.A. Potter, A.C. Schultz, J.F. Miller, E. Burke, and N. Jonoska, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2002)*, pages 439–447, San Francisco, California, July 2002. Morgan Kaufmann Publishers.
83. Z. Michalewicz and D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer-Verlag, 2000.
84. Kaisa Miettinen. Some methods for nonlinear multi-objective optimization. In E. Zitzler, K. Deb, L. Thiele, C. A. Coello Coello, and D. Corne, editors, *Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization (EMO 2001)*, volume 1993 of *Lecture Notes in Computer Science*, pages 1–20, Berlin, 2001. Springer-Verlag.
85. Pablo Moscato. *New Ideas in Optimization*, chapter Memetic Algorithms: A Short Introduction, pages 219–234. McGraw Hill, 1999.
86. Tadahiko Murata and Hisao Ishibuchi. Constructing Multi-Objective Genetic Local Search Algorithms for Multi-Objective Flowshop Scheduling Problems. In *Proceedings of the 1998 Japan-USA Symposium on Flexible Automation*, pages 1353–1356, Ohtsu, Japan, July 1998.
87. Tadahiko Murata, Hisao Ishibuchi, and Mitsuo Gen. Cellular Genetic Local Search for Multi-Objective Optimization. In Darrell Whitley, David Goldberg, Erick Cantú-Paz, Lee Spector, Ian Parmee, and Hans-Georg Beyer, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2000)*, pages 307–314, San Francisco, California, 2000. Morgan Kaufmann.
88. Tadahiko Murata, Hisao Ishibuchi, and Mitsuo Gen. Specification of Genetic Search Directions in Cellular Multi-objective Genetic Algorithms. In Eckart Zitzler, Kalyanmoy Deb, Lothar Thiele, Carlos A. Coello Coello, and David Corne, editors, *First International Conference on Evolutionary Multi-Criterion Optimization*, pages 82–95. Springer-Verlag. Lecture Notes in Computer Science No. 1993, 2001.
89. Tadahiko Murata, Shiori Kaige, and Hisao Ishibuchi. Generalization of Dominance Relation-Based Replacement Rules for Memetic EMO Algorithms. In Erick Cantú-Paz et al., editor, *Genetic and Evolutionary Computation—GECCO 2003. Proceedings, Part I*, pages 1234–1245. Springer. Lecture Notes in Computer Science Vol. 2723, July 2003.
90. Luis Paquete and Thomas Stützle. A Two-Phase Local Search for the Biobjective Traveling Salesman Problem. In Carlos M. Fonseca, Peter J. Fleming, Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele, editors, *Evolutionary*

Multi-Criterion Optimization. Second International Conference, EMO 2003, pages 479–493, Faro, Portugal, April 2003. Springer. Lecture Notes in Computer Science. Volume 2632.

91. Luís F. Paquete and Carlos M. Fonseca. A Study of Examination Timetabling with Multiobjective Evolutionary Algorithms. In Jorge Pinho de Sousa, editor, *Proceedings of the 4th Metaheuristics International Conference (MIC'2001)*, pages 149–153. Program Operational Ciencia, Tecnologia, Inovaçao do Quadro Comunitário de Apoio III de Fundaçao para a Ciencia e Tecnologia, Porto, Portugal, July 16–20 2001.
92. Kwang-Wook Park and Donald E. Grierson. Pareto-Optimal Conceptual Design of the Structural Layout of Buildings Using a Multicriteria Genetic Algorithm. *Computer-Aided Civil and Infrastructure Engineering*, 14(3):163–170, May 1999.
93. Sangbong Park, Dongkyung Nam, and Cheol Hoon Park. Design of a neural controller using multiobjective optimization for nonminimum phase systems. In *1999 IEEE International Fuzzy Systems Conference Proceedings*, volume 1, pages 533–537. IEEE, 1999.
94. Geoffrey T. Parks and I. Miller. Selective Breeding in a Multiobjective Genetic Algorithm. In A. E. Eiben, M. Schoenauer, and H.-P. Schwefel, editors, *Parallel Problem Solving From Nature — PPSN V*, pages 250–259, Amsterdam, Holland, 1998. Springer-Verlag.
95. Ian C. Parmee, Dragan Cvetković, Andrew H. Watson, and Christopher R. Bonham. Multiobjective Satisfaction within an Interactive Evolutionary Design Environment. *Evolutionary Computation*, 8(2):197–222, Summer 2000.
96. Günter Rudolph. Evolutionary Search for Minimal Elements in Partially Ordered Finite Sets. In V.W. Porto, N. Saravanan, D. Waagen, and A.E. Eiben, editors, *Evolutionary Programming VII, Proceedings of the 7th Annual Conference on Evolutionary Programming*, pages 345–353, Berlin, 1998. Springer.
97. J. David Schaffer. *Multiple Objective Optimization with Vector Evaluated Genetic Algorithms.* PhD thesis, Vanderbilt University, 1984.
98. Josef Schwarz and Jiri Ocenasek. Evolutionary Multiobjective Bayesian Optimization Algorithm: Experimental Study. In *Proceedings of the 35th Spring International Conference: Modelling and Simulation of Systems (MOSIS'01)*, pages 101–108, Czech Republic, 2001. MARQ, Hradec and Moravici.
99. Paolo Serafini. Simulated Annealing for Multiple Objective Optimization Problems. In G.H. Tzeng, H.F. Wang, U.P. Wen, and P.L. Yu, editors, *Proceedings of the Tenth International Conference on Multiple Criteria Decision Making: Expand and Enrich the Domains of Thinking and Application*, volume 1, pages 283–294, Berlin, 1994. Springer-Verlag.
100. N. Srinivas and Kalyanmoy Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. Technical report, Department of Mechanical Engineering, Indian Institute of Technology, Kanpur, India, 1993.
101. Rainer Storn and Kenneth Price. Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, International Computer Science Institute, Berkeley, CA, 1995.
102. Ricardo Szmit and Amnon Barak. Evolution Strategies for a Parallel Multi-Objective Genetic Algorithm. In Darrell Whitley, David Goldberg, Erick Cantú-Paz, Lee Spector, Ian Parmee, and Hans-Georg Beyer, editors, *Proceed-*

ings of the Genetic and Evolutionary Computation Conference (GECCO'2000), pages 227–234, San Francisco, California, 2000. Morgan Kaufmann.

103. El-Ghazali Talbi, Malek Rahoual, Mohamed Hakim Mabed, and Clarisse Dhaenens. A Hybrid Evolutionary Approach for Multicriteria Optimization Problems: Application to the Flow Shop. In Eckart Zitzler, Kalyanmoy Deb, Lothar Thiele, Carlos A. Coello Coello, and David Corne, editors, *First International Conference on Evolutionary Multi-Criterion Optimization*, pages 416–428. Springer-Verlag. Lecture Notes in Computer Science No. 1993, 2001.
104. Kay Chen Tan and Yun Li. Multi-Objective Genetic Algorithm Based Time and Frequency Domain Design Unification of Linear Control Systems. In *Proceedings of the IFAC/IEEE International Symposium on Artificial Intelligence in Real-Time Control*, pages 61–66, Kuala Lumpur, Malaysia, September 1997.
105. K.C. Tan, T.H. Lee, and E.F. Khor. Evolutionary Algorithms with Dynamic Population Size and Local Exploration for Multiobjective Optimization. *IEEE Transactions on Evolutionary Computation*, 5(6):565–588, December 2001.
106. Dirk Thierens and Peter A.N. Bosman. Multi-Objective Mixture-based Iterated Density Estimation Evolutionary Algorithms. In Lee Spector, Erik D. Goodman, Annie Wu, W.B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2001)*, pages 663–670, San Francisco, California, 2001. Morgan Kaufmann Publishers.
107. E.L. Ulungu, J. Teghem, Ph. Fortemps, and D. Tuyttens. MOSA Method: A Tool for Solving Multiobjective Combinatorial Optimization Problems. *Journal of Multi-Criteria Decision Analysis*, 8(4):221–236, 1999.
108. David A. Van Veldhuizen and Gary B. Lamont. Multiobjective Optimization with Messy Genetic Algorithms. In *Proceedings of the 2000 ACM Symposium on Applied Computing*, pages 470–476, Villa Olmo, Como, Italy, 2000. ACM.
109. David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1:67–82, 1997.
110. Kazuo Yamasaki. Dynamic Pareto Optimum GA against the changing environments. In *2001 Genetic and Evolutionary Computation Conference. Workshop Program*, pages 47–50, San Francisco, California, July 2001.
111. Gary G. Yen and Haiming Lu. Hierarchical Rank Density Genetic Algorithm for Radial-Basis Function Neural Network Design. In *Congress on Evolutionary Computation (CEC'2002)*, volume 1, pages 25–30, Piscataway, New Jersey, May 2002. IEEE Service Center.
112. Eckart Zitzler. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications.* PhD thesis, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, November 1999.
113. Eckart Zitzler and Lothar Thiele. An Evolutionary Algorithm for Multiobjective Optimization: The Strength Pareto Approach. Technical Report 43, Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, May 1998.
114. Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M. Fonseca, and Viviane Grunert da Fonseca. Performance Assessment of Multiobjective Optimizers: An Analysis and Review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, April 2003.
115. Jesse B. Zydallis, David A. Van Veldhuizen, and Gary B. Lamont. A Statistical Comparison of Multiobjective Evolutionary Algorithms Including

the MOMGA–II. In Eckart Zitzler, Kalyanmoy Deb, Lothar Thiele, Carlos A. Coello Coello, and David Corne, editors, *First International Conference on Evolutionary Multi-Criterion Optimization*, pages 226–240. Springer-Verlag. Lecture Notes in Computer Science No. 1993, 2001.

Part IV

Related Search Technologies

A Memetic Learning Classifier System for Describing Continuous-Valued Problem Spaces

David Wyatt[1] and Larry Bull[2]

[1] University of the West of England, Frenchay Campus, Bristol BS16 1QY
david2.wyatt@uwe.ac.uk
[2] University of the West of England, Frenchay Campus, Bristol BS16 1QY
larry.bull@uwe.ac.uk

Summary. Learning Classifier Systems have previously been shown to have some application in deducing the characteristics of complex multi-modal test environments to a suitable level of accuracy. In this study, an accuracy-based Learning Classifier System, XCS, is used. The system has the capability of inducing a set of general rules from a sample of data points using a combination of Reinforcement Learning and a Genetic Algorithm. The investigation presented here builds on earlier work in this area by considering the application of a memetic approach during learning. The motivation for this investigation is identify if any increases in learning speed and classification performance can be made. The type of memetic learning used is based on Lamarckian Evolution but has several subtle differences from the standard approach. In particular, the Learning Classifier System is based on a Reinforcement Learning paradigm that has a dynamic effect on the fitness landscape. And, the form of lifetime learning used is based on a Widrow-Hoff delta rule update procedure in which changes to an individual's genotypic description are based upon some distance measure between the individual and a "focal rule"' (analogous to a local optima in a standard MA). In addition, no distinction is made between genotype and phenotype. Initial investigations focus on the effects on performance for three different learning rates and three different "focal rule" identification options for two different test environments - a two-dimensional and a decomposable six-dimensional test environment. Results show that improvements can be made over a non-memetic approach. The study also considers the use of a self-adaptive learning mechanism. Self-adaptation has been suggested as beneficial for Genetic Algorithms where the technique is usually used for adapting the mutation rate in a time-dependant and decentralised way. However, the investigation of a self-adaptive learning mechanism presented here focuses on the benefits of adjusting the Widrow-Hoff learning rate used within the memetic-learning component of the system. The mechanism was applied to both test environments. Results show that the mechanism can provide a more robust learning system both in terms of reduction in the number of system parameters and increased generalisation and solution convergence. Further detailed analysis of experimental results for the decomposable six-dimensional test function is also performed. This would otherwise be non-trivial for a non-decomposable six-dimensional function. The classification accuracy of several different versions of the system in-

cluding those systems with and without memetic or self-adaptive memetic learning are analysed region by region showing the effects of the new learning approach at a much greater level of detail. Analysis shows that the self-adaptive memetic version of the classifier system outperforms the non-adaptive and non-memetic versions in some of the regions.

1 Introduction

There have been several published studies demonstrating the capabilities of XCS [30] for data-mining through rule-induction. XCS is a Learning Classifier System (LCS) [11] that is capable of inducing a set of general rules from a sample of data points using a combination of Reinforcement Learning [14] and a Genetic Algorithm [10]. In [4], Bernado et al. describe an experimental comparison of XCS with seven other learning schemes, including C4.5, Naive Bayes and Support Vector Machines. Fifteen UCI repository data-sets [5] were used in that study each having a mixture of attribute types and differing numbers of classes and data-set sizes. The XCS system was shown to be highly competitive when compared with the other learning schemes. Wilson [32][33] has also demonstrated the capabilities of an interval based encoding when used to induce rules describing the Wisconsin Breast Cancer data-set. In fact, XCS was shown to improve on the best known performance for that data-set. That is, the XCS classifier system can be cast as an induction engine that is trained using a reinforcement learning approach, i.e., an external agent provides a reward for each classified data instance. Once the system has completed its training, new unseen data are presented and a measure of classification accuracy made.

Initial investigations in [8] show that the XCSR system [31] (an extension of the binary-input XCS to real-inputs) is able to identify high performance regions from a continuous multi-variable search space using a sample of training data points. Parmee [25] introduced the concept of the identification of high performance regions of complex preliminary design spaces rather than the identification of single optimal design solutions. A region of high performance is any contiguous set of points in a given design space which are considered to be exceptional solutions to a particular set of possibly conflicting design criteria. The solution provided by XCSR is a complete set of simple classification rules that define orthogonal regions of the solution space with attached classification labels. Investigations continued using a new *Simplified Learning Scheme* with the aim of improving XCSR performance with respect to learning speed and ability to respond to changes in the underlying test environment (such as class relabelling). When using the *Simplified Learning Scheme* newly created rules have their expected payoff value set to that of the first training instance they experience. This value remains constant. The new system was termed sXCSR and results showed that improvements can be made under the new learning scheme. The work presented clearly demonstrated the capability of XCSR to evolve real-valued pairs to describe interval

bounds for each variable in the multi-variable problem and thereby define a set of simple classification rules for the high performance regions.

The investigation was extended in [34] by applying XCSR and sXCSR to progressively more complex multi-modal test environments each with typical search space characteristics, convex/non-convex regions of high performance and complex interplay between variables. In particular, two test environments were used to investigate the effects of different degrees of feature sampling, parameter sensitivity, training set size and rule subsumption. Both test environments are constructed using a combination of functions allowing for the simple generation of training and test points. Each sample point can be represented by a vector of continuous values and a continuous performance measure which may be discretised as appropriate. Both test environments are also used in this study. Fixed size training data-sets were used in an effort to provide some consistency in experimentation with those design problems for which the cost of an on-line evaluation per sample point is high or for which data-sets are constructed from other off-line data sources.

The study is arranged as follows: Sect. 2 presents a basic overview of memetic algorithms, and in particular, how this relates to the approach presented here; Sect. 3 describes the XCSR system used throughout; Sect. 4 describes the experimental details for this study; Sect. 5 describes and presents results for a two-dimensional test environment; Sect. 6 describes and presents results for a six-dimensional test environment; Sect. 7 presents the results from using a self-adaptive approach to tuning the memetic learning rate for both the two and six-dimensional test environments; Sect. 8 shows a more detailed analysis of a selection of six-dimensional test environment results by decomposing the test data-set into eighteen distinct regions of high performance and finally, all findings are discussed in Sect. 9. Section 10 defines several related equations that can be used to define the two and six dimensional test environments used in this study.

2 Memetic Algorithms

In [23], Moscato termed a Memetic Algorithm (MA) as "a marriage between a population-based global search and the heuristic local search made by each of the individuals" and made it clear that the global search need not be constrained to a genetic representation. However, for the purposes of this investigation, a more constrained definition provided by Krasnogor and Smith [18] is used, that is, "MAs are extensions of Evolutionary Algorithms (EAs) that apply separate local search processes to refine individuals". It should also be made clear that the local search processes used to refine individuals may include constructive and exact methods as well as iterative improvement techniques. The key feature of the memetic approach to learning is that individuals are permitted to learn during their lifetime. This type of learning is applied using the phenotypic description of an individual rather than the

genotypic description. However, for the evolutionary process to take advantage of this lifetime learning, the effects of any improvements need to be felt in succeeding generations of the genetic search. There are two basic models of evolution that can be used for this purpose. These are the Baldwin Effect [2][22][24] and Lamarckian Evolution [29].

The Baldwin Effect was discovered independently by Baldwin [2], Morgan [22] and Osborn [24] in 1896 and has since become known as the Baldwin Effect due to Baldwin's dedicated research efforts. The Baldwin Effect allows an individual's fitness to be determined as a result of the local search processes but without any resulting changes in phenotypic description being reflected in the individual's genotypic description. According to Whitley et al. [29], this technique has the effect of changing, or smoothing out, the fitness landscape while retaining the advantages of the evolutionary process. Here, it is not characteristics acquired during an individual's lifetime that are inherited, but its ability to acquire those characteristics.

Larmarckian Evolution, introduced by Jean Baptiste Larmarck in 1809, is based on an assumption that characteristics acquired during an individual's lifetime are inherited, that is, any changes to the phenotypic description will be reflected in the individual's genotypic description. One of the drawbacks of using this technique for real-world problem solving is the requirement for an inverse mapping from phenotype and environment to genotype. There is no such requirement when using the Baldwin Effect.

Despite the issues raised above, the Larmarckian approach is used as the basis for memetic learning in this investigation. However, there are several subtle differences between the standard approach to Larmarckian Evolution and that employed here. Firstly, the underlying Evolutionary Algorithm used is based on a Reinforcement Learning paradigm that has a dynamic effect on the fitness landscape. Secondly, the form of lifetime learning used is based on a Widrow-Hoff update procedure in which changes to an individual's genotypic description are based upon some distance measure between the individual and a "focal rule". Here, the "focal rule" is analogous to a local optima in a standard MA. It should be noted that no distinction is made between genotype and phenotype in this study.

In particular, this involves identifying a "focal rule" together with any members of the current *Action Set* that qualify for update. In order to qualify for update, a rule must have been a member of a sufficient number of previous *Action Sets*, that is, the rule is expected to have been updated enough times to show its true potential. This threshold is termed as the update qualification threshold and is denoted by the symbol ξ. The "focal rule" is a single rule used in a Widrow-Hoff update procedure applied to all qualifying *Action Set* members. In this investigation, three different approaches are used to identify the "focal rule": *Best Fitness*, *Most Numerate* and *Accurate*. Each approach involves sorting the *Action Set* members in descending order with the first member in the sorted list being defined as the "focal rule". It is possible in each of these approaches, and likely in the case of *Accurate*, for several

members of a given *Action Set* to be competing for the title of "focal rule", in which case, one of them is selected at random. Once the "focal rule" and members qualifying for update have been identified, the following update rule is applied, $x_i^j = x_i^j + \eta\left[F_i - x_i^j\right], \forall i, j$, where represents gene i of qualifying member j, F_i represents gene i of the "focal rule" F, and η is a learning rate between 0 and 1 applied to the update. The new system is termed mXCSR. Initial investigations focused on the effects on performance for three different learning rates and three different "focal rule" identification options over those for the standard non-memetic approach.

3 XCSR

In [31], Wilson presents a version of XCS [30] for problems which can be defined by a vector of bounded continuous real-coded variables – XCSR. In that system, each rule in the classifier system population consists of the following parameters: $< condition >$: $< action >$: prediction (p) : prediction error (ε) : fitness (F) : experience (exp) : time-stamp (ts) : action set size (as) : numerosity (n). Given that XCSR is an accuracy-based classifier system, the three parameters p, ε and F are used to assess the accuracy of the rule's prediction in relation to its experiences over time, that is, how accurately the rule predicts the actual reward or payoff from its use for a given environment input. The other parameters, exp, ts, as and n, are used by the classifier system to maintain the internal dynamics of the system, such as balancing resources across environmental niches, genetic algorithm invocation and computational issues.

Figure 1 shows a schematic illustration of the architecture of a single-step version of the XCSR system with a particular emphasis on the data-mining capabilities of the system. The following description of the process actions of the XCSR system can be found in algorithmic form in [9]. A sample point is selected at random from the database and is presented to the system as an *input vector*. The system defines a subset of the *Population*, called the *Match Set*, from those rules whose $< condition >$ *matches* the *input vector*, where each rule predicts one of n actions ($n = 2$ for this study). If there are no matching rules, the system generates, or *covers*, a rule for each possible action using the *input vector* as a template. The *Prediction Array* is *calculated* as a sum of the fitness-weighted prediction of each rule in the *Match Set*, that is, π represents the sum of fitness-weighted prediction for all rules advocating action 1 and ϕ for those advocating action 2. The action with the highest sum represents the systems "best guess" at the classification for the given *input vector*. There are two *action selection* regimes, explore and exploit. Assuming the system exploits its knowledge, the system defines a subset of the *Match Set*, called the *Action Set*, from those rules advocating the *selected action*. The *predicted class* is compared with the *actual class* for the given *input vector* and a *reward* is received, 1000 for correct and 0 for incorrect. The system *reinforces*

those rules in the *Action Set* using the *reward*. If the average number of time-steps since the last invocation of the *Genetic Algorithm* component is greater than some pre-defined threshold, the *Genetic Algorithm* is permitted to act upon the members of the *Action Set*.

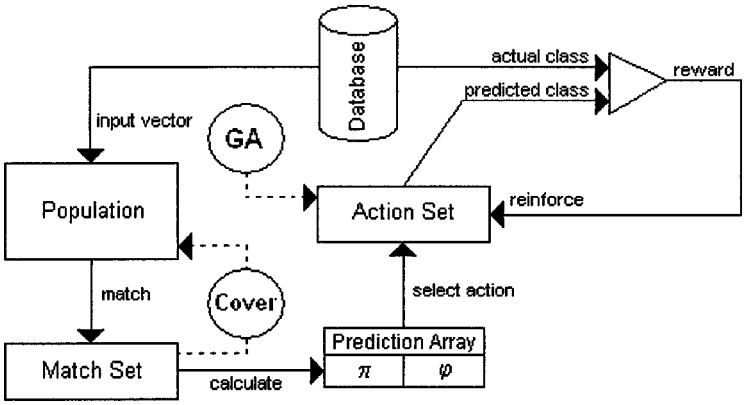


Fig. 1. The XCSR Classifier System Schematic Illustration for Single Step Data-mining Tasks

Another operator that acts upon the *Action Set* is the *subsumption* operator. One rule may subsume another if every interval predicate in the subsumee's $<condition>$ can be subsumed by the corresponding predicate in the subsumer, that is, for real-coded $<condition>$'s the subsumee's lower bound must be greater and its upper bound must be lesser than the corresponding subsuming predicate. In fact, XCSR implements two different forms of subsumption, *Action Set Subsumption* and *Genetic Algorithm Subsumption*. In the first form, a single rule is defined as the most general in a given *Action Set* and is permitted to subsume any other rule in the *Action Set* providing it is sufficiently experienced and accurate enough. In the second form of subsumption, a newly generated offspring rule may be subsumed if either of its parents are more general than it, sufficiently experienced and accurate enough.

In [31], Wilson defines a $<condition>$ as consisting of interval predicates of the form $\{\{c_1, s_1\}, \ldots, \{c_n, s_n\}\}$, where c is the interval's range "centre" and s is the "spread" from that centre – termed here as the *Centre-Spread* encoding. Each interval predicate's upper and lower bounds are calculated as follows : $[c_n - s_n, c_n + s_n]$. If an interval predicate goes outside the variable's defined bounds, it is truncated. In order for a rule to match the environmental stimulus, each input vector value must sit within the interval predicate defined for that variable.

In [32], Wilson describes another version of XCS which could also be used for such multi-variable problems in which a vector of integer-coded interval predicates is used in the form $\{[l_1, u_1], \ldots, [l_n, u_n]\}$, where l and u are the intervals' lower and upper bounds, respectively – termed here as the *Interval*

encoding. It is clear that a real-coded version of the integer bounded interval predicates would be trivial to implement. For both the *Centre-Spread* and *Interval* encoded versions, mutation is implemented via a random step (range $-0.1 \leq x \leq 0.1$) and cover produces rules centred on the input value with a "spread" of s_0.

It is important to note in the case of the *Interval* encoding, a potential problem may arise through the action of the mutation operator such that it is possible for a variable predicate's upper bound to become smaller than its lower bound. There are two ways to deal with this problem, termed here as *Ordered Interval* and *Unordered Interval* [26]. The first way uses a repair operator to enforce an ordering restriction on the predicates by swapping the offending values to ensure that all interval predicates in the $< condition >$ remain feasible, i.e., in the form $\{[l_1, u_1], \ldots, [l_n, u_n]\}$. The second way lifts the ordering restriction such that an interval $[l_n, u_n]$ is equivalent to $[u_n, l_n]$. The reader is referred to [26] for a discussion of the issues related to the differences between *Interval* encodings.

4 Experimental Details

The investigation presented in [34] compared the three different real-coded interval encodings described in Sect. 3 and showed that there was little or no difference between the encodings for the two and six-dimensional test environments used. Given that these same test environments are used in this study, the choice of encoding becomes an arbitrary one. In fact, the *Unordered Interval* encoding is used throughout.

The investigation also showed results for different sized training data-sets, that is, 500 and 2000 sample points for the two-dimensional environment and 6000 and 12000 sample points for the six-dimensional environment. In both cases, the larger training data-set led to better performance. For this reason, the 2000 and 12000 sample point data-sets are used in this study. Results presented in [34] for different population sizes show that a population of 8000 rules for the two-dimensional test environment and a population of 2000 rules for the six-dimensional test environment led to the best performance.

To clarify, all experiments in this study use a system based on the *Unordered Interval* encoding. Experiments using the two-dimensional environment run for a total of 200000 trials with a population size of $N = 8000$ while those using the six-dimensional test environment run for 250000 trials with a population size of $N = 2000$. XCSR's other parameters are defined as: $\beta = 0.2$, $\alpha = 0.1$, $\varepsilon_0 = 10$, $\nu = 5$, $\theta_{GA} = 12$, $\chi = 0.8$, $\mu = 0.04$, $\theta_{del} = 20$, $\delta = 0.1$, $p_I = 10$, $\varepsilon_I = 0$, $F_I = 0.01$, $\theta_{mna} = 2$, $\theta_{sub} = 20$, plus $m = \pm 10\%$ and $s_0 = 2\%$ for the two-dimensional environment while $m = \pm 10\%$ and $s_0 = 25\%$ for the six-dimensional environment.

An approach is required to enable comparisons of performance to be made between different parameter sets. One common way to define the performance

of a classification system is to use a confusion matrix [16] of size $L \times L$, where L is the number of different classifications. The matrix contains information about the actual and predicted classifications resulting from the classification task and provides a simple format to record and analyse system performance. Figure 2 gives an example of a 2×2 confusion matrix with the two possible

		Predicted Class	
		Low	High
Actual Class	Low	**a**	**b**
	High	**c**	**d**

Fig. 2. An Example 2×2 Confusion Matrix with *High* and *Low* Classifications

Several measures of classification accuracy based on confusion matrices were developed to overcome problems associated with analysis where the number of examples in each classification is significantly different. These include Lewis and Gale's F-measure [20], the geometric mean as defined by Kubat et al. in [19], using ROC graphs to examine classifier performance [27] and Kononenko and Bratko's information-based evaluation criterion [17].

Given that all the test data-sets used in this study have been manipulated such that the number of examples per classification are nearly equal, a simple accuracy measure will suffice for basic analysis. For a two-class classification problem, the accuracy measure is defined as the number of examples correctly classified as *High* plus the number correctly classified as *Low* divided by the total number of examples classified, that is, $(a + d)/(a + b + c + d)$ according to Fig. 2. Unclassified test examples are not included in this measure, that is, the denominator may not always equal the number of sample points in the test data-set. The percentage of *High* and *Low* points correctly classified are traditionally known as sensitivity and specificity, respectively. These terms frequently appear in the medical literature and are mainly used to describe the result of medical trials for disease prevention, but have come to be used in many non-medical classification tasks including information retrieval. A similar set of performance metrics were introduced for the EpiCS [12] system.

An important aspect of the experimental method identified in [34] was the class imbalance problem [13][28] which can be defined as a problem encountered by any inductive learning system in domains for which one class is under-represented and which assume a balanced class distribution in the

training data. For a two-class problem, the class defined by the smaller set of examples is referred to as the minority class while the other class is referred to as the majority class. Initial experiments for the six-dimensional test environment used in [34], and here, showed that without applying some form of rebalancing of the class distribution, the six-dimensional test problem could not be described to an acceptable level of accuracy. The solution used in [34] for the six-dimensional test environment was an approach suggested by Ling and Li [21] that makes use of both minority over-sampling and majority under-sampling. In particular, the minority class is re-sampled with replacement until some pre-defined multiple, n, of the original sample size is achieved. The majority class is re-sampled without replacement until a number of examples equal to those sampled from the minority class have been defined. The Ling and Li re-sampling solution is restricted to the six-dimensional test environment as was the case in [34].

For all experiments presented in this study, the new mXCSR system is trained using a single training data-set and tested using a different test data-set generated from a uniform random distribution. The test data-sets have been manipulated in such a way as to provide an equal number of test points per classification. In particular, n points are sampled from a uniform random distribution and evaluated according to the given environment. The sample points are sorted in descending order of performance and the top $2m$ points are used to define the test data-set, where m equals the total number of *High* points generated. All data-sets used have two defined classes, *High* and *Low*. The training data-sets are generated from a Halton Sequence Leaped (HSL) sequence [15], where the HSL is a quasi-random sequence that provides a set of real numbers whose degree of uniformity is high. By manipulating the test data-sets to include sample points from both classifications near to the classification decision boundaries, it is hoped that clear evidence of the classifier system's capability to evolve rules that define those boundaries will be gathered.

The results for each parameter setting of the mXCSR system are averaged over ten independent runs and presented together with a standard deviation for that sample. Any conclusions made are based on the application of Mann-Whitney Rank Sum Test which makes no assumptions about the distribution of population from which the runs where sampled.

5 A Two-dimensional Test Environment

The two-dimensional test environment used in this paper is the multi-modal modified Himmelblau function [3]. The equation for the modified Himmelblau function, which is used to evaluate each sample point, is given in Sect. 10.1. There are four optima of approximately equal magnitude. This function is used to define a two-class classification task to investigate the effects of using a memetic approach to learning within the XCSR classifier system,

that is, mXCSR. In particular, an exact threshold value of $\psi = 184$, where $\psi \in [-1986, 200]$, is used to define *High/Low* class decision boundaries. Figure 3 shows a contour plot of the function, clearly indicating the four regions of high performance as defined by the threshold value given above.

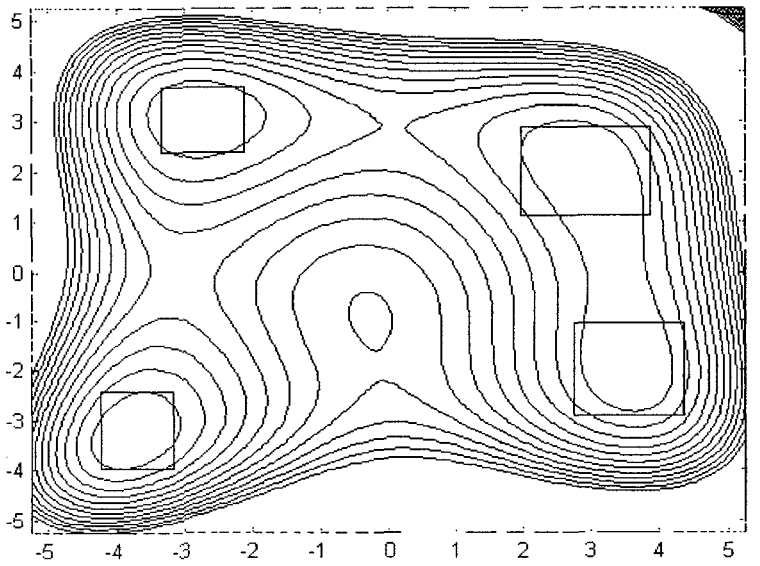


Fig. 3. The Modified Himmelblau Function Contour with Four High Performance Regions

The new mXCSR system was trained using a single training data-set, Fig. 4, generated from a HSL sequence with 2000 sample points and was tested using a different data-set generated from a uniform random distribution. This test data-set has 2116 sample points of which 1073 points are defined as *High* – shown as faint dots in Fig. 5.

Table 1 shows a single performance measure (ten run average with standard deviation) for each parameter combination using the Uniform Random test data-set. The parameter combinations used include running the new mXCSR system both with and without *Action Set Subsumption* and *Simplified Learning Scheme* using three different approaches to "focal rule" identification together with three different learning rates for each approach.

It is clear from Table 1 that the system performed well on the two-dimensional Himmelblau test problem in terms of correct classification of unseen data, between 72.8% and 90.1% depending on "focal rule" identification approach, learning rate, subsumption type and whether or not the *Simplified Learning Scheme* was used. The performance gain for mXCSR when *Action Set Subsumption* is turned off is remarkably clear in Table 1. In fact, the difference between mXCSR with and without *Action Set Subsumption* is statistically significant ($> 99.9\%$) for all parameter settings shown. Although, a

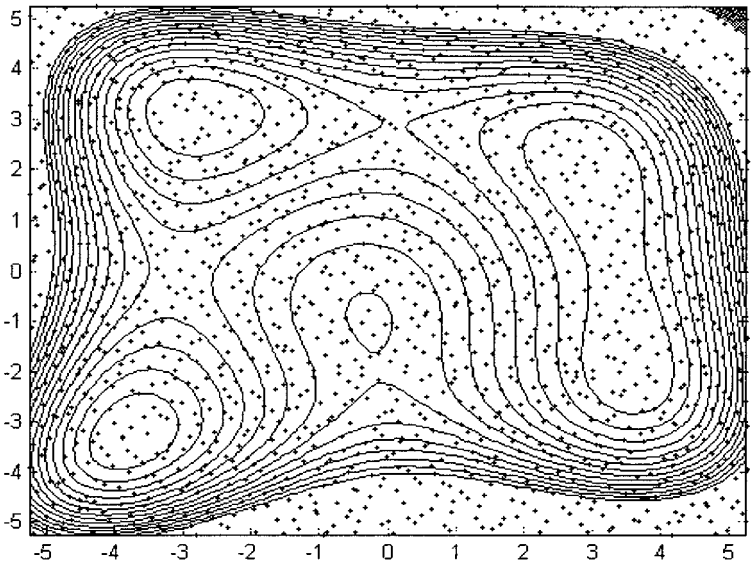


Fig. 4. Training Data-set with 2000 HSL-generated Sample Points

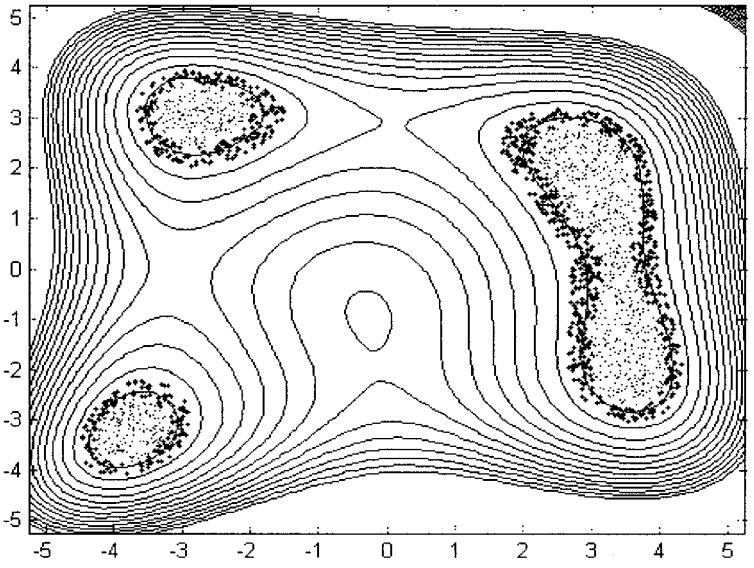


Fig. 5. Test Data-set with 2116 Sample Points Generated from a Uniform Random Distribution

Table 1. Classification Accuracy on Test Data for Several Different Versions of XCSR when applied to a Two-dimensional Test Environment

	With *AS-Sub*		Without *AS-Sub*	
	mXCSR	smXCSR	mXCSR	smXCSR
No Memetic	$74.2_{(6.8)}$	$67.8_{(4.0)}$	$90.9_{(0.6)}$	$90.3_{(0.8)}$
Best Fitness$_{(\eta=0.1)}$	$80.6_{(4.7)}$	$75.7_{(6.3)}$	$89.4_{(1.0)}$	$89.0_{(1.1)}$
Best Fitness$_{(\eta=0.2)}$	$83.0_{(2.6)}$	$77.4_{(3.2)}$	$89.5_{(1.0)}$	$88.7_{(1.4)}$
Best Fitness$_{(\eta=0.4)}$	$80.9_{(4.5)}$	$75.4_{(3.4)}$	$89.5_{(1.2)}$	$88.5_{(1.9)}$
Most Numerate$_{(\eta=0.1)}$	$78.0_{(7.6)}$	$72.8_{(7.4)}$	$89.7_{(1.5)}$	$89.5_{(1.2)}$
Most Numerate$_{(\eta=0.2)}$	$81.3_{(1.8)}$	$78.5_{(4.4)}$	$89.3_{(0.7)}$	$88.6_{(1.2)}$
Most Numerate$_{(\eta=0.4)}$	$75.5_{(4.4)}$	$75.6_{(2.2)}$	$89.4_{(0.9)}$	$89.4_{(0.8)}$
Accurate$_{(\eta=0.1)}$	$85.4_{(1.8)}$	$82.5_{(3.2)}$	$90.1_{(1.4)}$	$88.9_{(1.8)}$
Accurate$_{(\eta=0.2)}$	$83.0_{(3.2)}$	$77.7_{(4.5)}$	$89.1_{(0.8)}$	$88.8_{(1.4)}$
Accurate$_{(\eta=0.4)}$	$82.4_{(4.5)}$	$77.0_{(4.3)}$	$89.6_{(1.2)}$	$88.6_{(1.1)}$

difference has been identified in previous work [34], it is clear from the results shown here that the difference is significantly smaller, that is, the memetic learning approach has help to mitigate some of the problems associated with using *Action Set Subsumption* in the two-dimensional test domains. Figures 6 and 7 show the performance gain for the mXCSR system using the *Accurate*$_{(\eta=0.1)}$ "focal rule" identification approach when *Action Set Subsumption* is turned off.

Regarding different "focal rule" identification approaches, Table 1 shows a clear difference in performance between *Accurate*$_{(\eta=0.1)}$ and the other two approaches, *Most Numerate*$_{(\eta=0.1)}$ and *Best Fitness*$_{(\eta=0.1)}$, when *Action Set Subsumption* is turned on. This effect is a statistically significant ($> 99\%$) improvement in performance. In fact, a learning rate of 0.1 can also be shown to be statistically significant ($> 95\%$) when compared with the other two settings for the Accurate "focal rule" identification approach.

Figures 6 and 8 show a comparison of performance based on the *Simplified Learning Scheme*, that is, Fig. 6 does not use it and Fig. 8 does. Both systems are based on the *Accurate*$_{(\eta=0.1)}$ approach with *Action Set Subsumption* turned on. Results suggest that there may be a slight degradation in performance when the *Simplified Learning Scheme* is used. In fact, this result is statistically significant ($> 97.5\%$). Figures 9 and 10 show the learning speed and system error for the same two systems. These figures show that despite final performance being significantly different, there is no increase in learning speed as demonstrated in [8].

Overall, results suggest that when *Action Set Subsumption* is turned on, the memetic learning helps to improve system performance significantly above

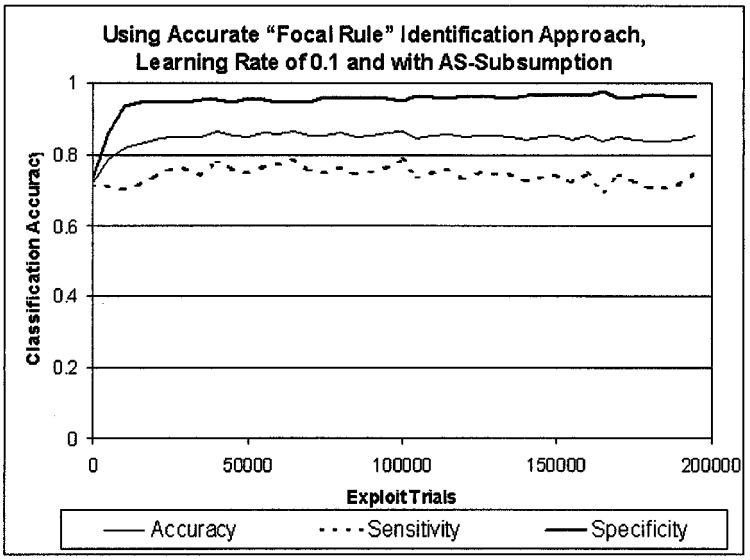


Fig. 6. Classification Accuracy for mXCSR with *Action Set Subsumption* using the $Accurate_{(\eta=0.1)}$ "focal rule" identification approach

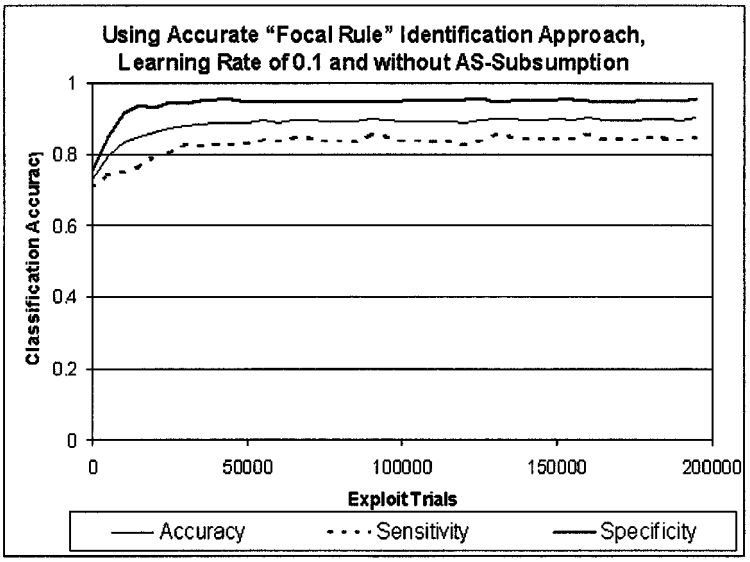


Fig. 7. Classification Accuracy for mXCSR without *Action Set Subsumption* using the $Accurate_{(\eta=0.1)}$ "focal rule" identification approach

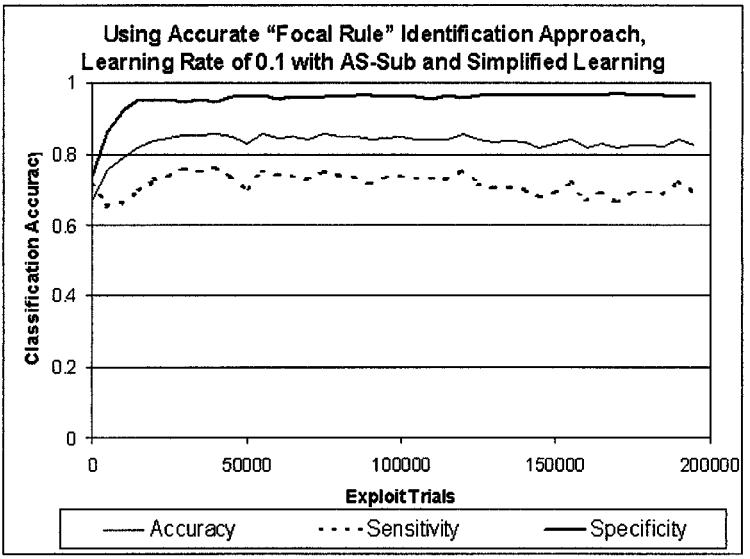


Fig. 8. Classification Accuracy for mXCSR with *Action Set Subsumption* and *Simplified Learning Scheme* using the $Accurate_{(\eta=0.1)}$ "focal rule" identification approach

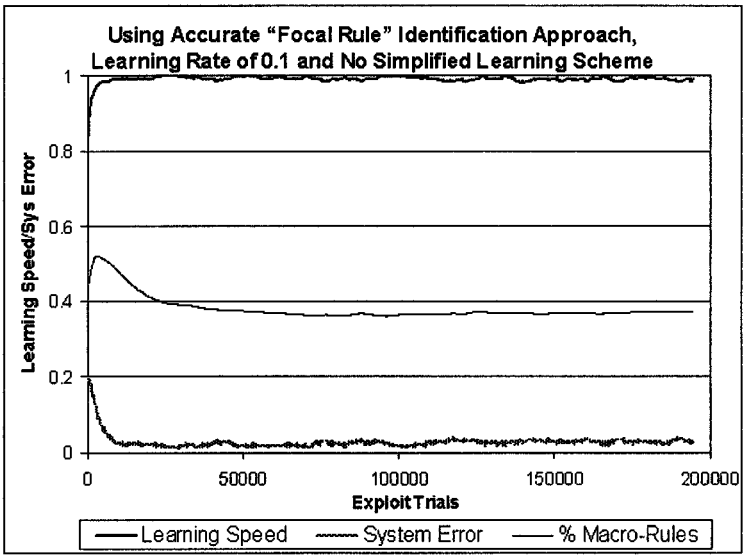


Fig. 9. Learning Speed and System Error for mXCSR with *Action Set Subsumption* and without *Simplified Learning Scheme* using the $Accurate_{(\eta=0.1)}$ "focal rule" identification approach

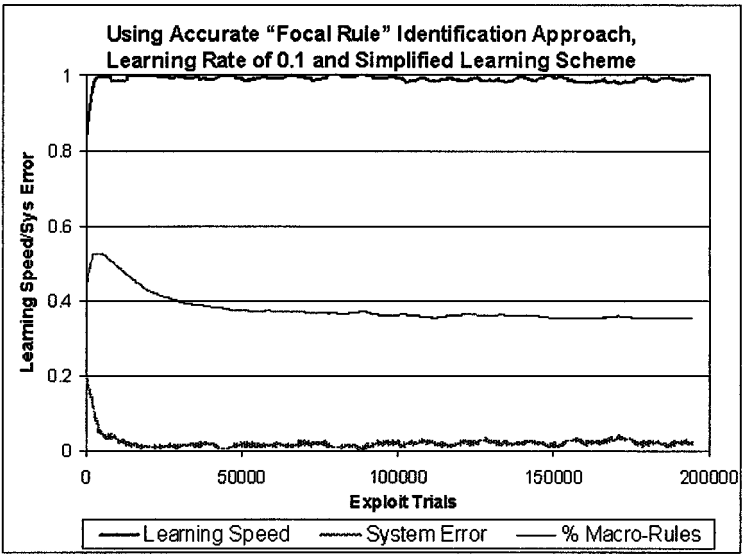


Fig. 10. Learning Speed and System Error for mXCSR with *Action Set Subsumption* and with *Simplified Learning Scheme* using the $Accurate_{(\eta=0.1)}$ "focal rule" identification approach

that of a non-memetic system except for the $Most\ Numerate_{(\eta=0.1)}$ approach. In fact, this improvement is statistically significant at a level of $> 95\%$. However, it also clear that the memetic learning degrades performance for those systems that do not use *Action Set Subsumption*. Although this degradation is very small (around 2%), it is statistically significant ($> 95\%$).

6 A Six-dimensional Test Environment

The six-dimensional test environment used in this paper is a multi-modal function developed by Bonham and Parmee, [6] and [7], and is described in Sect. 10.2. It is defined by the additive effect of three different two-dimensional planes, as shown in contour plot form in Figs. 11–13 . Each plane has an associated "local" fitness value and the "global" fitness value of the six-dimensional function is defined by adding each of these "local" fitness values together, that is, $fitness_{global} = fitness_{plane1} + fitness_{plane2} + fitness_{plane3}$. Each sample point is defined by a six-dimensional vector of the form $\{a, b, c, d, e, f\}$, where $a \ldots f \in [0, 1]$.

In this two-class classification problem, a sample point is classified as either *High* or *Low*. It is classified as *High* only when each "local" fitness value is greater than the exact threshold value $\psi = 0.35$, where $\psi \in [0, 0.5]$, and the "global" fitness value is greater than exact threshold value $\psi_G = 1.20$, where

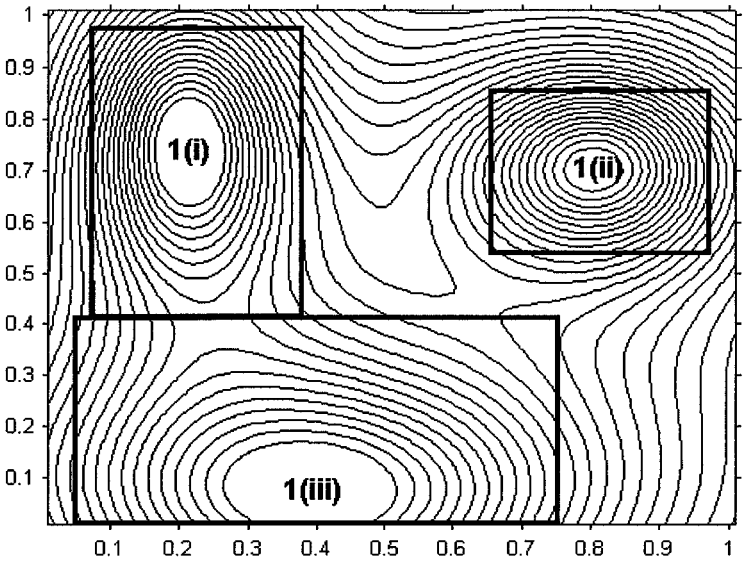


Fig. 11. Plane 1 of Bonham and Parmee's Six-dimensional Function showing Three Local Regions of High Performance

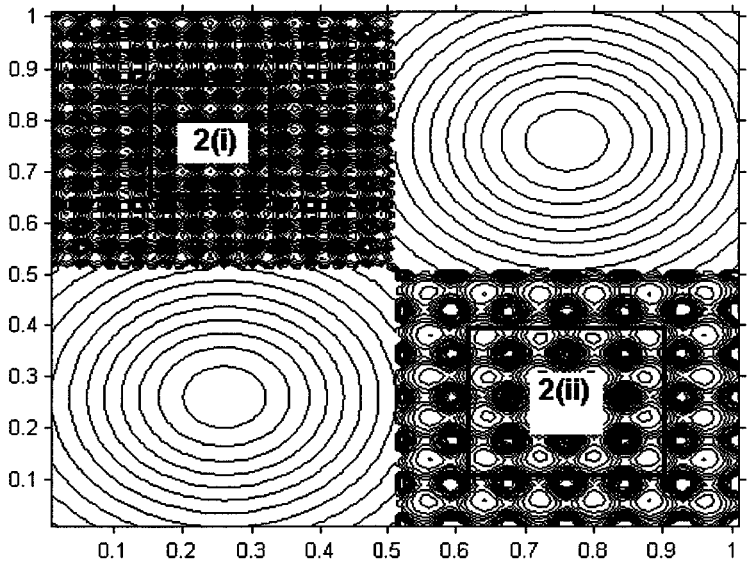


Fig. 12. Plane 2 of Bonham and Parmee's Six-dimensional Function showing Two Local Regions of High Performance

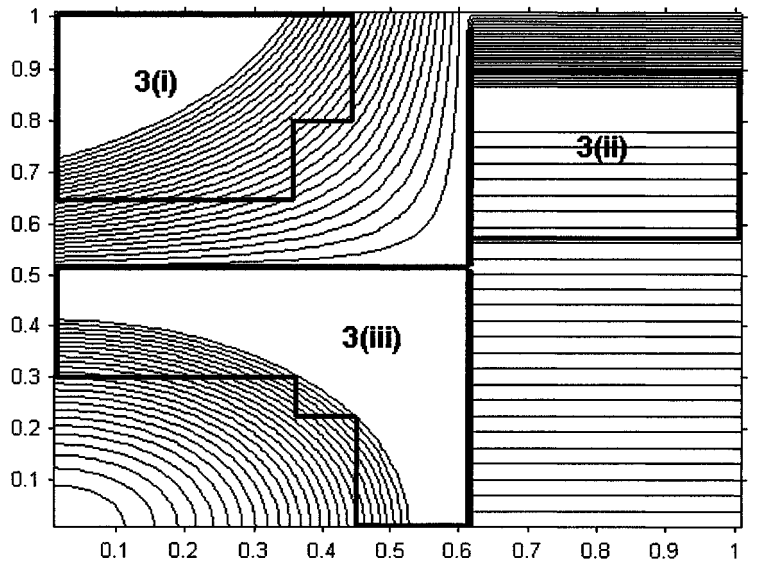


Fig. 13. Plane 3 of Bonham and Parmee's Six-dimensional Function showing Three Local Regions of High Performance

$\psi_G \in [0, 1.5]$, otherwise the point is classified as *Low*. By combining local regions of high performance, as shown in Table 2, an environment of eighteen unique regions of globally high performance are defined, that is, three local high performance regions in Plane 1, two in Plane 2 and three in Plane 3. In fact, Sect. 8 presents an analysis of a selection of results for this environment emphasising its decomposable nature.

The XCSR system was trained using a single 12000 sample point training data-set (not shown) generated from a HSL sequence which is re-balanced using the approach suggested by Ling and Li [21] with a predefined multiple of $n = 32$. The new data-set consists of 11968 sample points, that is, within 3% of the original size. The system was tested using a data-set generated from a uniform random distribution with 1693 sample points of which 813 points are defined as *High*. The test data-set is manipulated to include sample points from both classifications near to the classification decision boundaries as was the case in the two-dimensional test environment.

In general, Table 3 shows no significant difference in performance between "focal rule" identification approaches except where the standard deviation figure is high. However, there is a statistically significant difference (> 97.5%) between $Accurate_{(\eta=0.2)}$ and the other approaches as well as between $Accurate_{(\eta=0.4)}$ and *Most Numerate*. The memetic learning appears to provide little or no advantage in performance over the non-memetic system except for a slight degradation when *Action Set Subsumption* is turned off. This degradation is statistically significant (> 95%). Figures 14 and 15 show a comparison

Table 2. Eighteen Unique Regions of Globally High Performance (HPR_{region}) based on the Six-dimensional Test Function

	Plane 1	Plane 2	Plane 3
HPR_1	(i)	(i)	(i)
HPR_2	(i)	(i)	(ii)
HPR_3	(i)	(i)	(iii)
HPR_4	(i)	(ii)	(i)
HPR_5	(i)	(ii)	(ii)
HPR_6	(i)	(ii)	(iii)
HPR_7	(ii)	(i)	(i)
HPR_8	(ii)	(i)	(ii)
HPR_9	(ii)	(i)	(iii)
HPR_{10}	(ii)	(ii)	(i)
HPR_{11}	(ii)	(ii)	(ii)
HPR_{12}	(ii)	(ii)	(iii)
HPR_{13}	(iii)	(i)	(i)
HPR_{14}	(iii)	(i)	(ii)
HPR_{15}	(iii)	(i)	(iii)
HPR_{16}	(iii)	(ii)	(i)
HPR_{17}	(iii)	(ii)	(ii)
HPR_{18}	(iii)	(ii)	(iii)

of performance between a memetic learning and non-memetic system where both system have *Action Set Subsumption* turned off. It is clear from the figure that there is little difference between these systems for the given parameters settings.

Results also show that there is no clear difference in performance between using and not using the *Simplified Learning Scheme*. However, there seems to be the potential for further increases in learning speed when using the *Simplified Learning Scheme* as shown in Figs. 16 and 17, where *Action Set Subsumption* was also used. It is clear from the figure that the *Simplified Learning Scheme* does provide the same level of performance some 25000–30000 exploit trials quicker than the standard scheme. Given that the speed-up was not apparent in the simpler two-dimensional test environment, this provides some strength to an argument put forth in [8] in which it was hypothesized that the *Simplified Learning Scheme* may have a greater effect on performance as the complexity of the test environment increases.

In order to qualify for update, a rule must have been a member of a sufficient number of previous *Action Sets*, that is, the rule is expected to have been updated enough times to show its true potential. Given the similarities

Table 3. Classification Accuracy on Test Data for Several Different Versions of XCSR when applied to a Six-dimensional Test Environment

	With *AS-Sub*		Without *AS-Sub*	
	mXCSR	smXCSR	mXCSR	smXCSR
No Memetic	$71.3_{(20.9)}$	$80.6_{(1.2)}$	$82.2_{(0.8)}$	$82.4_{(0.6)}$
Best Fitness$_{(\eta=0.1)}$	$77.9_{(9.8)}$	$81.1_{(0.7)}$	$69.3_{(23.9)}$	$80.2_{(1.5)}$
Best Fitness$_{(\eta=0.2)}$	$81.3_{(0.9)}$	$81.0_{(1.1)}$	$81.0_{(1.3)}$	$80.0_{(1.7)}$
Best Fitness$_{(\eta=0.4)}$	$71.3_{(21.0)}$	$81.0_{(0.9)}$	$80.8_{(1.6)}$	$81.4_{(1.2)}$
Most Numerate$_{(\eta=0.1)}$	$80.8_{(1.2)}$	$80.5_{(1.1)}$	$80.4_{(1.5)}$	$79.9_{(1.1)}$
Most Numerate$_{(\eta=0.2)}$	$81.6_{(1.1)}$	$80.9_{(0.8)}$	$75.7_{(15.6)}$	$80.4_{(1.1)}$
Most Numerate$_{(\eta=0.4)}$	$81.7_{(0.9)}$	$81.8_{(1.0)}$	$71.2_{(21.4)}$	$80.6_{(1.3)}$
Accurate$_{(\eta=0.1)}$	$80.6_{(1.2)}$	$79.8_{(1.6)}$	$80.8_{(0.9)}$	$78.8_{(0.7)}$
Accurate$_{(\eta=0.2)}$	$73.9_{(18.1)}$	$78.9_{(1.8)}$	$77.3_{(7.0)}$	$78.8_{(2.2)}$
Accurate$_{(\eta=0.4)}$	$80.0_{(1.5)}$	$79.9_{(1.4)}$	$79.6_{(1.2)}$	$79.6_{(1.3)}$

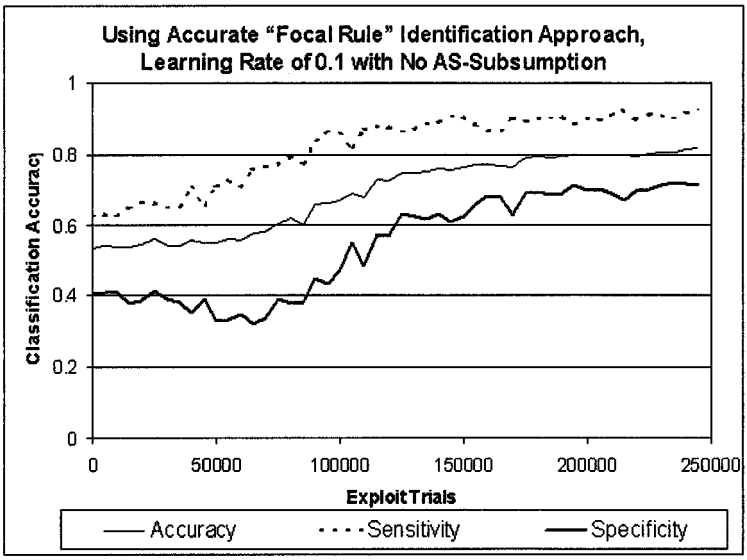


Fig. 14. Classification Accuracy for mXCSR without *Action Set Subsumption* using the *Accurate*$_{(\eta=0.1)}$ "focal rule" identification approach

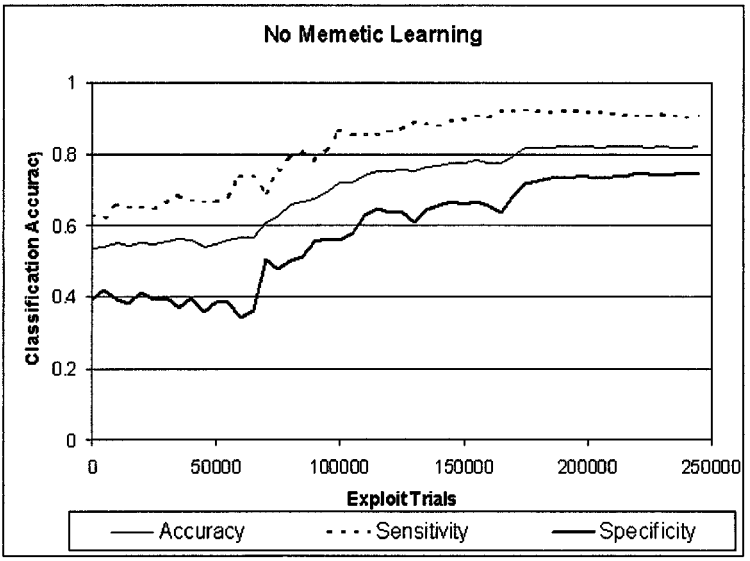


Fig. 15. Classification Accuracy for XCSR without *Action Set Subsumption*

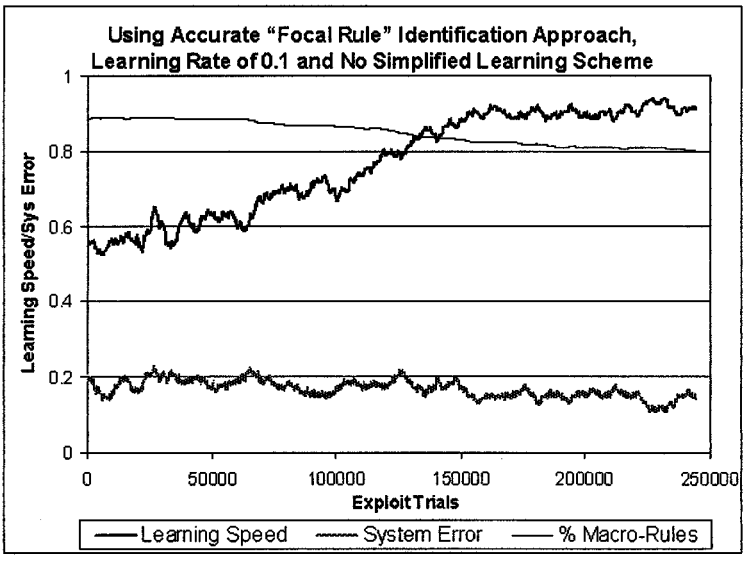


Fig. 16. Learning Speed and System Error for mXCSR with *Action Set Subsumption* and without *Simplified Learning Scheme* using the $Accurate_{(\eta=0.1)}$ "focal rule" identification approach

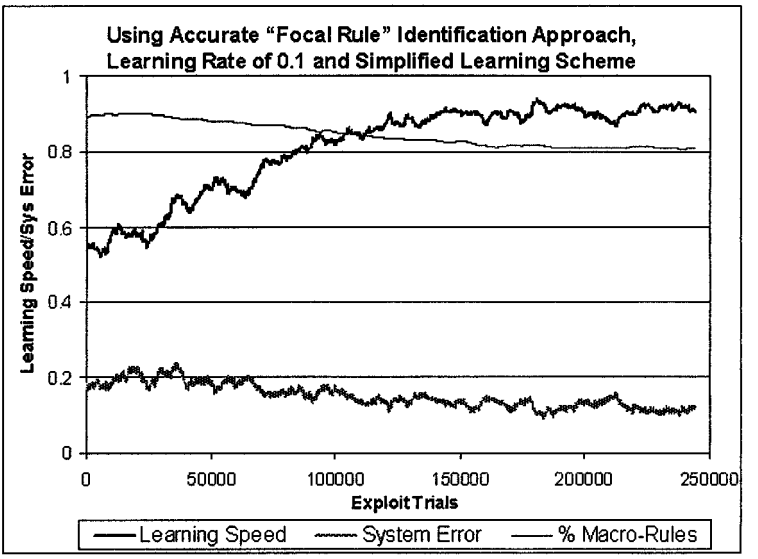


Fig. 17. Learning Speed and System Error for mXCSR with *Action Set Subsumption* and with *Simplified Learning Scheme* using the $Accurate_{(\eta=0.1)}$ "focal rule" identification approach

in nature between the subsumption operator and the memetic learning suggested in this study, the update qualification value, ξ, has been fixed at 20 to this point. However, there was some evidence in [8] and [34] that permitting initially weaker rules enough time to show their true potential by reducing the early domination of more numerate rules in a given *Action Set* can lead to improvements. Based on this insight, another set of experiments was performed using the $Accurate_{(\eta=0.1)}$ "focal rule" identification approach where the threshold for update qualification varies between 20 and 200. The choice of identification approach was somewhat arbitrary given the results in Table 3 above.

It is clear from the result of these experiments, shown in Table 4, that a small improvement can be made in performance when $\xi = 50$ or $\xi = 100$, except for any highly variant results. In fact, this effect is statistically significant at the level of $> 95\%$ for differences between $\xi = 20$ and $\xi = 50$. There is also significant difference ($> 95\%$) in performance between those systems using the *Simplified Learning Scheme* with and without *Action Set Subsumption*. It should be noted that those settings of ξ greater than 100 lead to a performance that is not statistically different from the original settings of $\xi = 20$.

On closer examination, the differences in performance for those systems using the memetic learning but with *Action Set Subsumption* turned off, as shown in Table 3, are mitigated by the alteration of the threshold ξ, as shown

Table 4. Classification Accuracy on Test Data for Several Different Versions of XCSR when applied to a Six-dimensional Test Environment

	With *AS-Sub*		Without *AS-Sub*	
	mXCSR	smXCSR	mXCSR	smXCSR
No Memetic	$71.3_{(20.9)}$	$80.6_{(1.2)}$	$82.2_{(0.8)}$	$82.4_{(0.6)}$
$Accurate_{(\eta=0.1,\xi=20)}$	$80.6_{(1.2)}$	$79.8_{(1.6)}$	$80.8_{(0.9)}$	$78.8_{(0.7)}$
$Accurate_{(\eta=0.1,\xi=50)}$	$76.8_{(18.0)}$	$81.7_{(1.2)}$	$81.4_{(1.8)}$	$82.7_{(0.7)}$
$Accurate_{(\eta=0.1,\xi=100)}$	$81.4_{(1.4)}$	$80.2_{(1.8)}$	$81.8_{(1.2)}$	$82.1_{(0.9)}$
$Accurate_{(\eta=0.1,\xi=200)}$	$80.7_{(1.4)}$	$80.0_{(1.2)}$	$75.5_{(18.9)}$	$82.4_{(1.0)}$

in Table 4. That is to say that when the qualification threshold is between 50 and 100 for systems with *Action Set Subsumption* turned off, performance is comparable with a standard non-memetic system. Figures 18–21 illustrate this effect and also show that the number of rules required to achieve this level of performance is some 20% larger in the case of the memetic learning version of the system. This is likely to be one side effect of performing a Widrow-Hoff update on the qualifying members of a given *Action Set*, that is, there may be several rules in the rule-base with almost identical $<condition>$ parts but which cannot be subsumed in the normal manner.

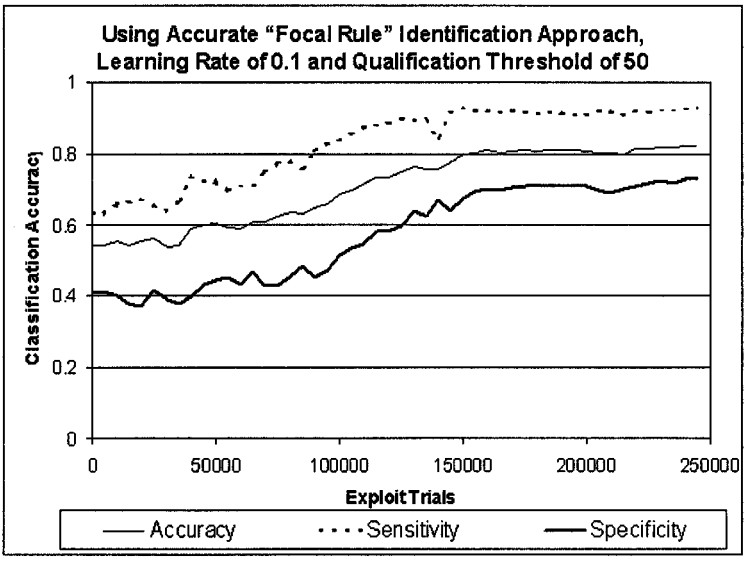


Fig. 18. Classification Accuracy for mXCSR without *Action Set Subsumption* using the $Accurate_{(\eta=0.1,\xi=50)}$ "focal rule" identification approach

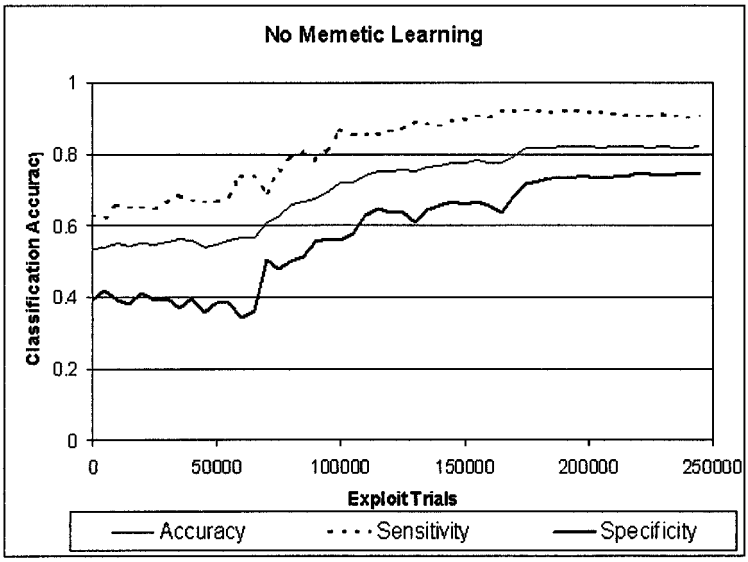


Fig. 19. Classification Accuracy for XCSR without *Action Set Subsumption*

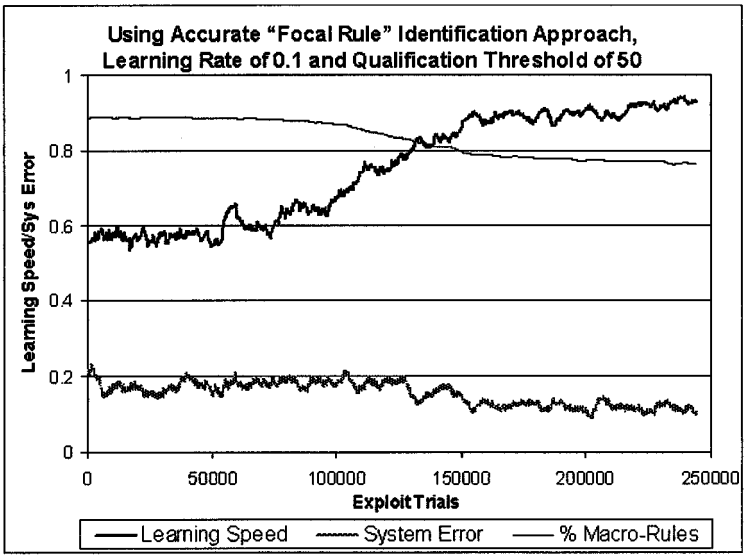


Fig. 20. Learning Speed and System Error for mXCSR without *Action Set Subsumption* using the $Accurate_{(\eta=0.1,\xi=50)}$ "focal rule" identification approach

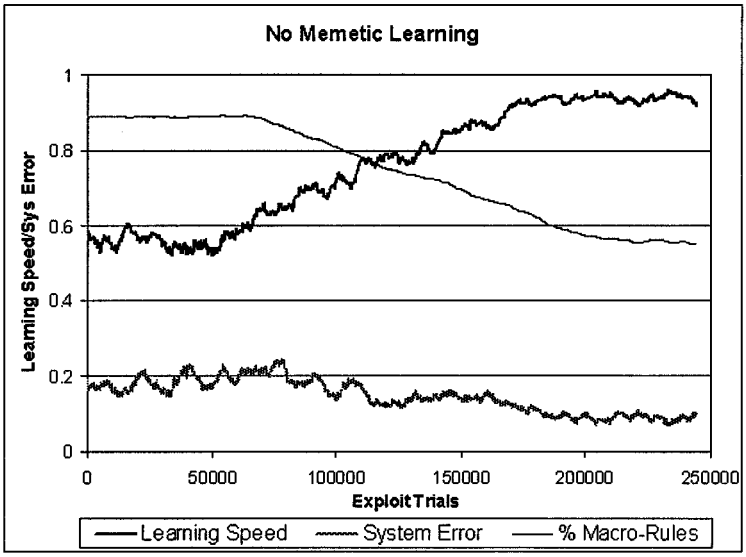


Fig. 21. Learning Speed and System Error for XCSR without *Action Set Subsumption*

7 Self-Adaptive Memetic Learning

Self-adaptation was suggested as beneficial for Genetic Algorithms in [1] where the author investigated the use of a technique for adapting the GA's mutation rate in a time-dependant and decentralised way. One advantage of using this technique is that the external global rate becomes an internal time-dependant rate, that is, there is a reduction in the number of parameters that require careful, possibly problem-dependant, setting by the user. Given this study, the investigation of a self-adaptive learning mechanism presented here focuses on the benefits of using this technique to adjust the Widrow-Hoff learning rate used within the memetic learning component of the new system. It is hoped that providing the classifier system with this capability will allow positive selective pressure toward a learning rate that is in proportion to the degree of solution convergence aiding both learning speed and solution quality in the process.

The self-adaptation is implemented by adding an extra real-coded gene to each rule in the population of the classifier system representing that particular rule's Widrow-Hoff learning rate. The approach used is similar to that presented in [1] except that the gene is affected by the mutation operator in the same way as the other genes, plus it is not included in the Widrow-Hoff update. Learning rates are initially assigned a random value in the range $[0.0, 1.0]$.

This section has been divided into two subsections, that is, results for the two-dimensional and six-dimensional test environments, respectively.

7.1 Two-dimensional Test Environment

The results shown in Sect. 5 suggest that, under certain circumstances, the $Accurate_{(\eta=0.1)}$ "focal rule" identification approach provides the best level of performance and as such it has been used for the following set of experiments. In particular, four self-adaptive versions of the system (with and without *Action Set Subsumption* and *Simplified Learning Scheme*) are compared with the same non-adaptive versions (η fixed at 0.1).

Table 5. Classification Accuracy on Test Data for Several Different Versions of XCSR when applied to a Two-dimensional Test Environment

	With *AS-Sub*		Without *AS-Sub*	
	mXCSR	smXCSR	mXCSR	smXCSR
$Accurate_{(\eta=0.1,\xi=20)}$	$85.4_{(1.8)}$	$82.5_{(3.2)}$	$90.1_{(1.4)}$	$88.9_{(1.8)}$
$Accurate_{(\eta=Adaptive,\xi=20)}$	$68.6_{(4.5)}$	$67.2_{(4.1)}$	$90.8_{(0.7)}$	$90.7_{(0.7)}$

Table 5 shows a clear degradation in performance, of around 16%, for the self-adaptive versions when *Action Set Subsumption* is turned on. In fact, this difference is statistically significant at the level of greater than 99.9%. However, the table also shows that when *Action Set Subsumption* is turned off there is an improvement of between 0.7% and 2.8% for the self-adaptive versions of the system. This difference is statistically significant at the level of greater than 95%. Figures 22 and 23 shows a comparison of performance between self-adaptive and non-adaptive versions of the system where *Action Set Subsumption* is turned off and the *Simplified Learning Scheme* is not used. This comparison provides some evidence of the degree of convergence shown by both versions of the system and, in particular, how that level is greater in the self-adaptive version. Another apparent advantage of using the self-adaptive technique is an increase in generalisation as indicated by fall in % Macro-Rules during the experiment as seen in Figs. 24 and 25.

Finally, Fig. 26 shows the average learning rate for all four self-adaptive versions of the system, that is, with and without both *Action Set Subsumption* and *Simplified Learning Scheme.* This figure clearly shows that average learning rates are very noisy when *Action Set Subsumption* is turned on, but that the average learning rates for the other two systems have much in common with both the falling % Macro-Rules and rising degree of convergence, that is, positive selective pressure toward lower learning rates for systems without *Action Set Subsumption* exists.

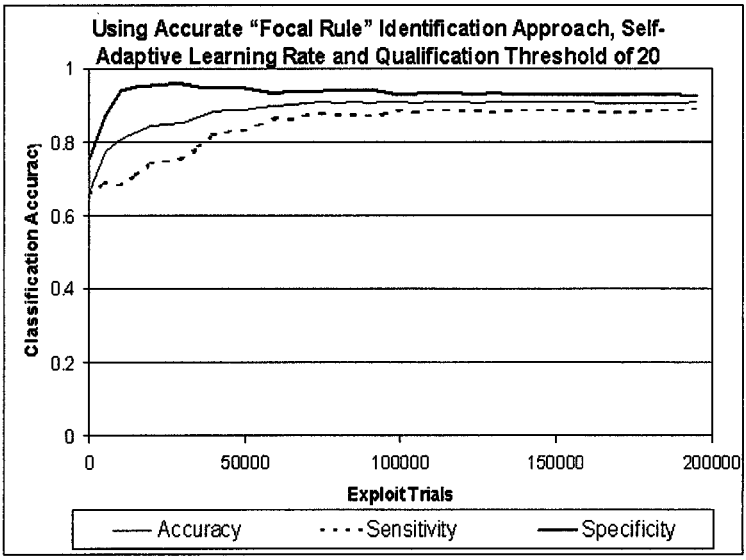


Fig. 22. Classification Accuracy for mXCSR without *Action Set Subsumption* using the *Accurate* "focal rule" identification approach with Self-Adaptive Memetic Learning

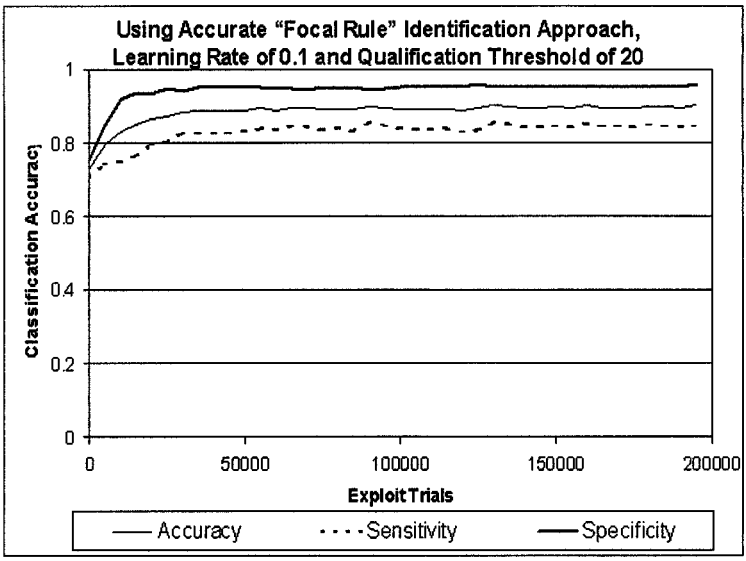


Fig. 23. Classification Accuracy for mXCSR without *Action Set Subsumption* using the $Accurate_{(\eta=0.1)}$ "focal rule" identification approach

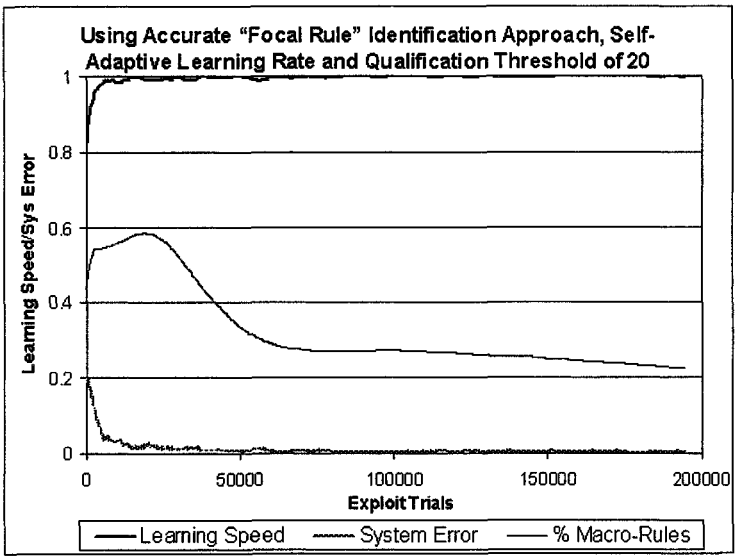


Fig. 24. Learning Speed and System Error for mXCSR without *Action Set Subsumption* using the *Accurate* "focal rule" identification approach with Self-Adaptive Memetic Learning

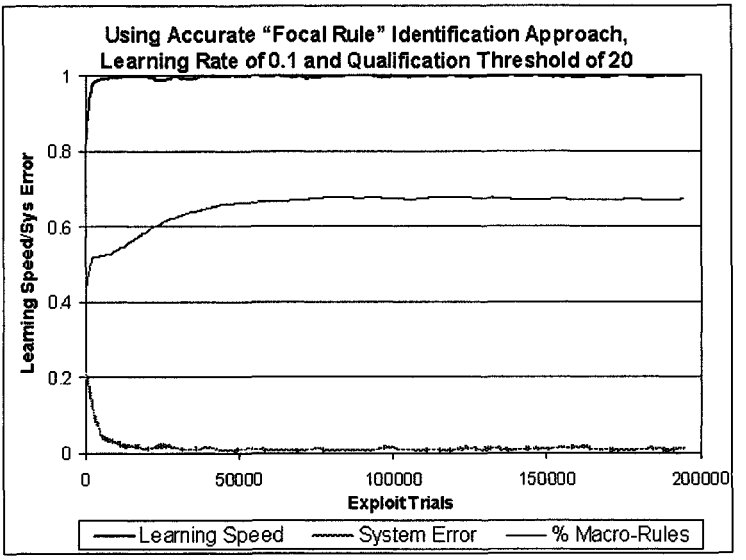


Fig. 25. Learning Speed and System Error for mXCSR without *Action Set Subsumption* using the $Accurate_{(\eta=0.1)}$ "focal rule" identification approach

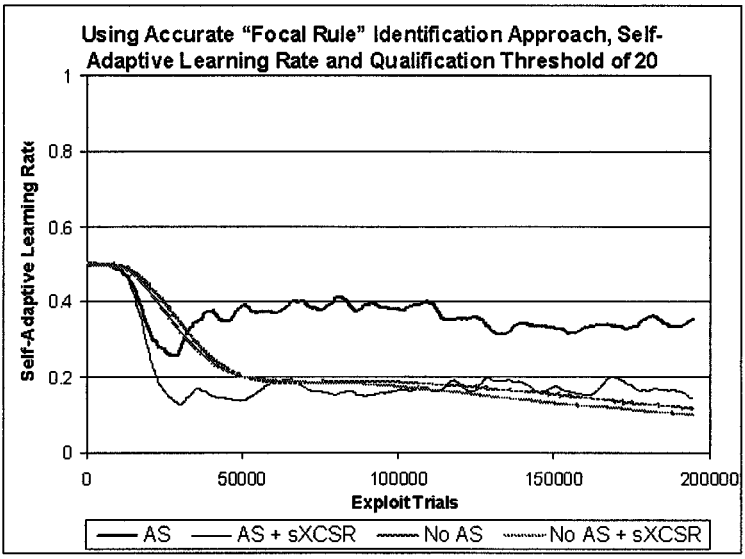


Fig. 26. Self-Adaptive Learning Rates for mXCSR with and without *Action Set Subsumption* and *Simplified Learning Scheme* using the *Accurate* "focal rule" identification approach

7.2 Six-dimensional Test Environment

Although results from using the $Accurate_{(\eta=0.1)}$ "focal rule" identification approach in the six-dimensional test environment were less clear than was the case in the two-dimensional environment, this identification approach is used throughout this subsection. It is hoped that using this approach together with the use of two different update qualification values, $\xi = 20$ and $\xi = 50$, will provide a consistent picture of the effectiveness of the self-adaptive approach. In this subsection, eight self-adaptive versions of the system (with and without *Action Set Subsumption* and *Simplified Learning Scheme* for two different update qualification values) are compared with the same non-adaptive versions.

Table 6 presents results for both update qualification values. It is clear from the results that improvements in performance have been made for all but two of the self-adaptive versions of the system. In fact, the improvements shown for three of the four systems using $\xi = 20$ are statistically significant at the level of greater than 95% and, in particular, at a level greater than 97.5% for the two systems using the *Simplified Learning Scheme*. Although the other four systems using $\xi = 50$ do not show significant improvements, they do provide evidence of the robustness of the self-adaptive technique, that is, there is no significant degradation in performance even when changes to other sensitive parameters (such as ξ) are made.

Table 6. Classification Accuracy on Test Data for Several Different Versions of XCSR when applied to a Six-dimensional Test Environment

	With *AS-Sub*		Without *AS-Sub*	
	mXCSR	smXCSR	mXCSR	smXCSR
$Accurate_{(\eta=0.1,\xi=20)}$	$80.6_{(1.2)}$	$79.8_{(1.6)}$	$80.8_{(0.9)}$	$78.8_{(0.7)}$
$Accurate_{(\eta=Adaptive,\xi=20)}$	$81.8_{(1.1)}$	$81.2_{(1.0)}$	$80.2_{(5.5)}$	$82.5_{(0.7)}$
$Accurate_{(\eta=0.1,\xi=50)}$	$76.8_{(18.0)}$	$81.7_{(1.2)}$	$81.4_{(1.8)}$	$82.7_{(0.7)}$
$Accurate_{(\eta=Adaptive,\xi=50)}$	$81.8_{(1.0)}$	$82.2_{(1.0)}$	$82.1_{(0.9)}$	$82.1_{(1.5)}$

Figures 27 and 28 show a comparison of performance between self-adaptive and non-adaptive versions of the system ($\xi = 20$) where *Action Set Subsumption* is turned on and the *Simplified Learning Scheme* is not used. The increase in generalisation as well as in the degree of convergence seen in the two-dimensional test environment is also evident here – although the magnitudes of both are reduced. It is also clear from Figs. 29 and 30 that there is some evidence of a speed-up in learning when using the self-adaptive version of the system, where the adaptive system reaches an accuracy of 0.8 at around 20000 exploit trials earlier than the non-adaptive version of the system. Results for the self-adaptive version of the system using an update qualification value of 50 (not shown) do not indicate any learning speed-up nor do they show an increase in generalisation, although there is evidence of a greater degree of convergence during the last 20000–30000 exploit trials of the experimental run.

Figure 31 shows the average learning rate for four self-adaptive ($\xi = 20$) versions of the system, that is, with and without both *Action Set Subsumption* and *Simplified Learning Scheme*. This figure clearly shows a positive selective pressure for lower average learning rates after an initial period of around 50000 exploit trials. The initial period of static average learning rate is linked to the effects of the covering operator and the way it initialises the learning rates of newly generated rules uniformly in the interval $[0.0, 1.0]$ leading to an average learning rate of 0.5.

Figure 32 shows the average learning rate for four self-adaptive ($\xi = 50$) versions of the system. This figure shows that there is little positive selective pressure for lower average learning rates. The reason for this lack of selective pressure may be a result of the increasing the delay before the memetic learning is applied thereby providing evolution with less opportunity to test a rule's learning rate. A simple experiment was performed to see if there was selective pressure toward a higher average learning rate, that is, initial learning rates were restricted to the interval $[0.0, 0.2]$. Results (not shown) indicate that there is little selective pressure even when the interval is restricted providing some evidence to support the hypothesis given above.

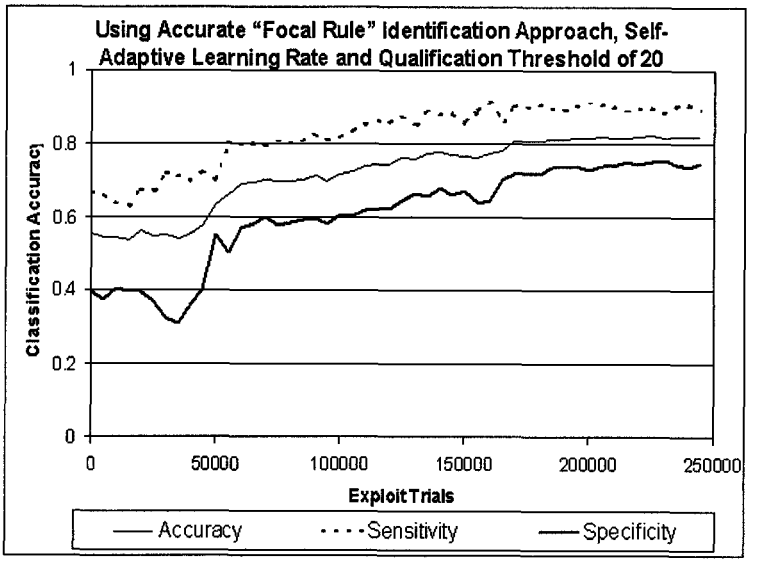


Fig. 27. Classification Accuracy for mXCSR with *Action Set Subsumption* and without *Simplified Learning Scheme* using the $Accurate_{(\xi=20)}$ "focal rule" identification approach with Self-Adaptive Memetic Learning

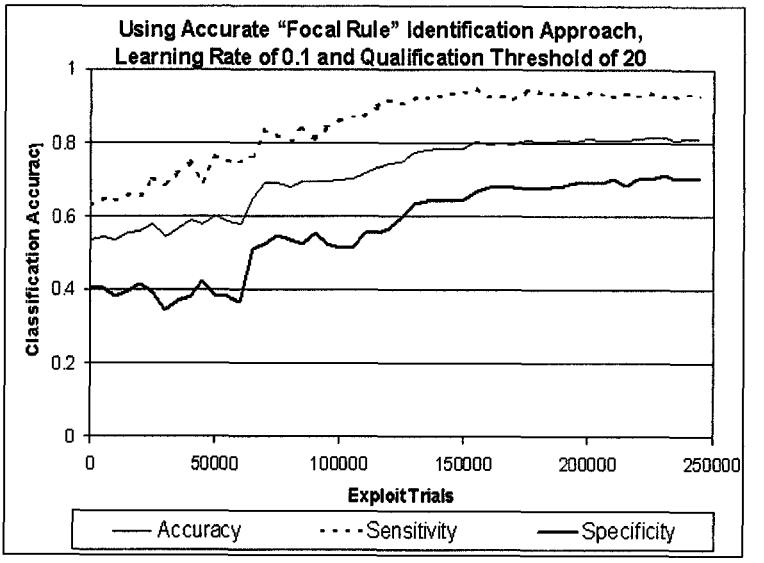


Fig. 28. Classification Accuracy for mXCSR with *Action Set Subsumption* and without *Simplified Learning Scheme* using the $Accurate_{(\eta=0.1,\xi=20)}$ "focal rule" identification approach

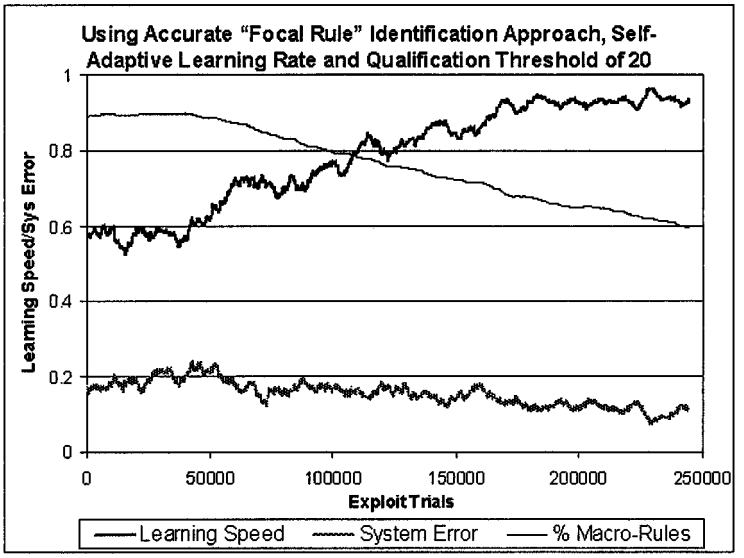


Fig. 29. Learning Speed and System Error for mXCSR with *Action Set Subsumption* and without *Simplified Learning Scheme* using the $Accurate_{(\xi=20)}$ "focal rule" identification approach with Self-Adaptive Memetic Learning

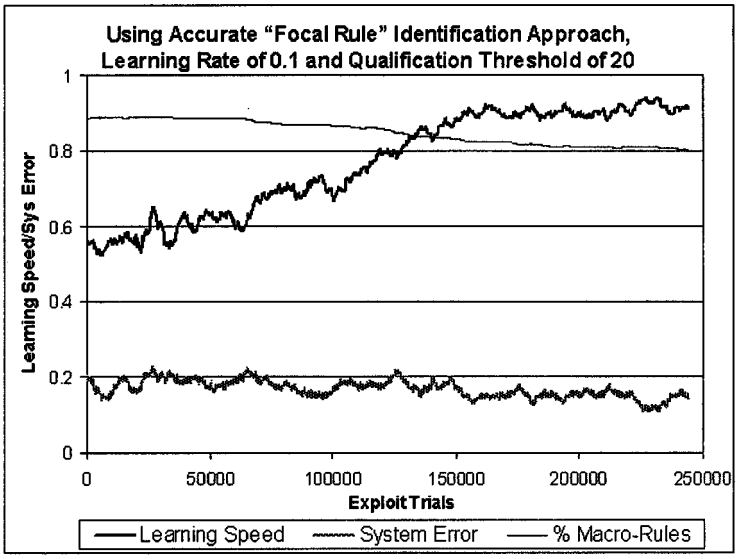


Fig. 30. Learning Speed and System Error for mXCSR with *Action Set Subsumption* and without *Simplified Learning Scheme* using the $Accurate_{(\eta=0.1,\xi=20)}$ "focal rule" identification approach

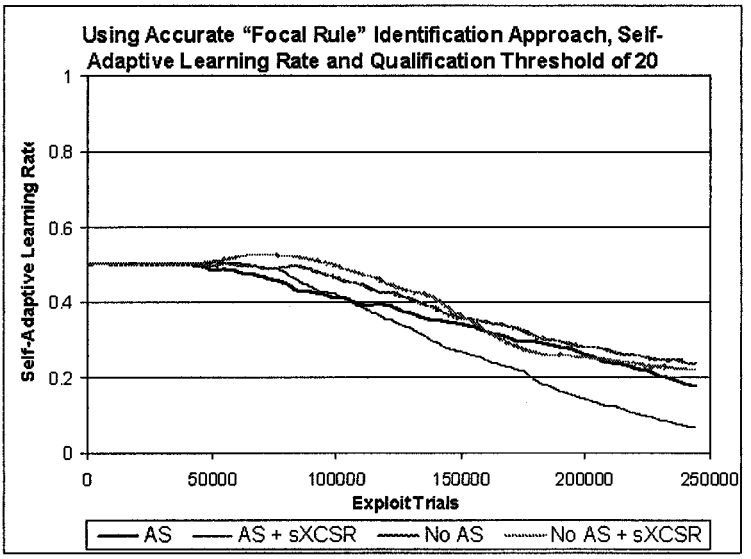


Fig. 31. Self-Adaptive Learning Rates for mXCSR with and without *Action Set Subsumption* and *Simplified Learning Scheme* using the $Accurate_{(\xi=20)}$ "focal rule" identification approach

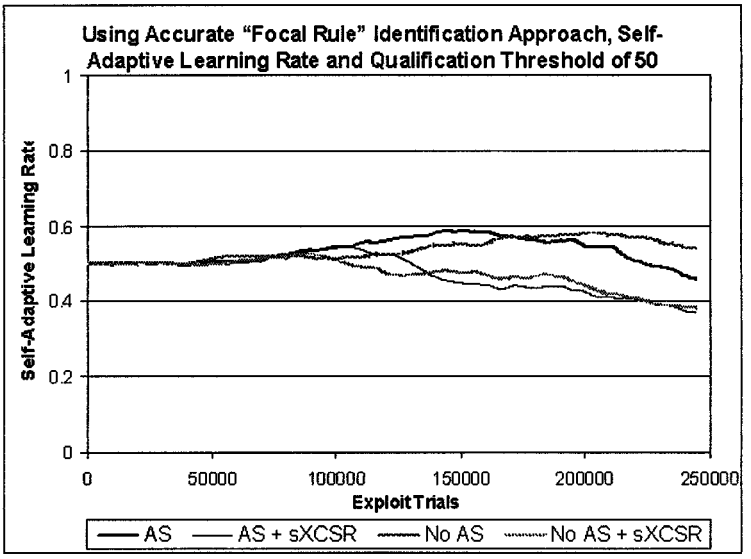


Fig. 32. Self-Adaptive Learning Rates for mXCSR with and without *Action Set Subsumption* and *Simplified Learning Scheme* using the $Accurate_{(\xi=50)}$ "focal rule" identification approach

8 Decomposing the Six-dimensional Test Environment

The six-dimensional test environment used in this study is defined by the additive effect of three different two-dimensional planes, shown in Figs. 11–13, allowing detailed analysis of experimental results that would otherwise be non-trivial for a non-decomposable six-dimensional function. By combining local regions of high performance, as shown in Table 2, an environment of eighteen unique regions of globally high performance is defined.

Table 7 shows the classification accuracy of twelve different versions of the system ($\xi = 20$) that include those systems with and without memetic or self-adaptive memetic learning, termed *Memetic*, *Non-Memetic* and *Self-Adaptive Memetic* (or *SA-Memetic*) accordingly. Comparisons between these systems show that for all parameter settings the *SA-Memetic* version is statistically equivalent to the strongest of the *Non-Memetic* and *Memetic* versions of the system and is significantly ($> 95\%$) better in 1 or 2 regions.

It is clear from the results that the performance of the system in each of the eighteen regions is at least 60% (except for 5 regions in Table 7) and for many regions performance is greater than 70%. Figure 33 represents an overview of Table 7 with respect to the number of regions falling into the performance intervals $[0\%, 70\%]$ and $[70\%, 100\%]$ for the *Memetic*, *Non-Memetic* and *SA-Memetic* versions of the system. It is clear from Fig. 33 that in terms of performance across the regions that the *SA-Memetic* version outperforms the other two versions and it is also clear that the number of regions above 70% performance rises from *Non-Memetic* through *Memetic* to *SA-Memetic*.

9 Conclusion

The motivation for this work was to investigate the effects of applying a memetic learning paradigm to the normal operations of the XCSR classifier system. Previous work showed how the classifier system is able to describe high performance regions in a design-oriented environment. It was hoped that the new memetic learning method would provide results that were at least comparable with the original system. In fact, the new method showed clear improvements for some parameter combinations in the two test environments studied here.

Results for the two-dimensional test environment show a clear improvement in performance for those parameter combinations that have *Action Set Subsumption* turned on and the system is still performing at a significantly higher level when *Action Set Subsumption* is turned off. Results for this environment highlighted a clear statistically significant improvement ($> 99\%$) in performance for $Accurate_{(\eta=0.1)}$ over the other two approaches when *Action Set Subsumption* is turned on and only slightly weaker performance when turned off. The $Accurate_{(\eta=0.1)}$ "focal rule" identification approach appears to be the best one for this test environment. However, the *Simplified Learning*

Table 7. Classification Accuracy on Test Data for Several Different Versions of XCSR when applied to a Six-dimensional Test Environment by High Performance Region (HPR_{region}), where (a)*Non-Memetic*, (b)*Memetic* or (c)*SA-Memetic*

	With *AS-Sub*						Without *AS-Sub*					
	XCSR			sXCSR			XCSR			sXCSR		
	(a)	(b)	(c)	(a)	(b)	(c)	(a)	(b)	(c)	(a)	(b)	(c)
HPR_1	63.7	63.5	66.0	65.6	65.4	66.5	67.3	65.4	66.0	68.3	60.0	68.3
HPR_2	73.1	83.1	83.6	80.8	79.4	80.8	85.3	84.4	83.1	88.1	81.4	87.5
HPR_3	67.1	70.7	73.4	68.0	69.3	69.2	74.6	70.1	73.1	76.3	69.7	75.6
HPR_4	72.0	71.6	78.7	82.1	80.3	81.5	79.7	78.9	79.0	80.3	76.6	80.3
HPR_5	69.0	73.3	79.6	80.8	79.6	79.2	78.7	78.3	78.3	80.0	81.0	79.0
HPR_6	68.4	69.2	77.8	75.3	77.2	77.0	76.9	76.0	76.1	78.3	75.4	77.1
HPR_7	72.0	76.0	76.0	74.5	76.5	79.0	78.5	75.5	79.5	76.1	76.5	82.5
HPR_8	71.0	77.9	80.3	69.0	78.6	74.5	77.9	78.6	76.9	79.5	78.6	83.1
HPR_9	63.4	65.4	69.4	65.6	66.4	65.9	71.1	67.6	69.3	76.9	64.1	73.0
HPR_{10}	56.3	57.0	60.0	62.2	63.0	64.1	69.3	65.9	65.2	65.6	63.7	61.9
HPR_{11}	58.8	71.5	68.5	58.1	75.4	64.6	76.9	74.6	74.6	71.2	77.7	73.1
HPR_{12}	54.8	57.5	61.6	62.7	64.9	64.4	63.7	63.0	62.4	63.5	63.3	63.5
HPR_{13}	68.2	69.8	73.2	75.0	70.3	76.1	74.2	69.7	71.6	75.0	70.3	72.6
HPR_{14}	70.2	73.8	79.7	77.5	73.2	77.2	78.6	75.9	77.9	79.9	73.2	80.4
HPR_{15}	73.1	75.8	81.7	78.8	76.8	79.1	80.8	76.0	77.9	80.0	75.3	80.8
HPR_{16}	74.1	76.8	83.5	83.2	80.5	82.2	82.6	77.3	80.5	82.7	78.0	83.1
HPR_{17}	66.7	69.3	77.4	76.1	76.5	75.4	77.5	72.4	75.1	76.9	74.6	76.9
HPR_{18}	67.3	71.8	76.7	76.7	77.3	78.4	76.9	73.9	75.1	75.9	74.9	77.0

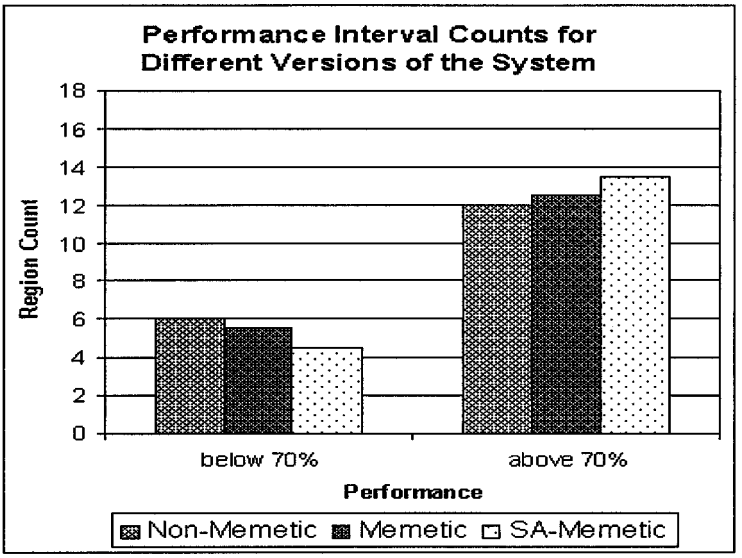


Fig. 33. Performance Interval Counts for *Memetic*, *Non-Memetic* and *SA-Memetic* Systems ($\xi = 20$)

Scheme appeared to degrade performance when *Action Set Subsumption* was used as well as failing to provide any increases in learning speed.

For the more complex six-dimensional test environment, the memetic learning appears to provide little or no advantage to performance over the non-memetic system except for a slight degradation when *Action Set Subsumption* is turned off. However, the memetic learning approach does provide a clear learning speed-up of some 25000–30000 exploit trials over non-memetic learning when the *Simplified Learning Scheme* is used. It is clear that any potential increase in learning speed must be balanced with a potential decrease in performance for a given problem.

Another important result was seen when the update qualification value, ξ, was allowed to vary between 20 and 200 in experiments on the six-dimensional test environment. Results showed that a small, but statistically significant ($> 95\%$) improvement can be made in performance when $\xi = 50$. In fact, differences in performance for those systems with *Action Set Subsumption* turned off, discussed above, are mitigated by the alteration of the threshold ξ to 50.

The study also considered the use of a self-adaptive learning mechanism for adjusting the Widrow-Hoff learning rate used in the memetic learning. The mechanism was applied to both test environments. Results for the two-dimensional test environment showed that although performance varies (depending on *Action Set Subsumption*) there is clear evidence of increased generalisation as well as solution convergence when the self-adaptive technique is

used. Results for the six-dimensional test environment also showed evidence of increased generalisation and solution convergence in addition to a learning speed-up of some 20000 exploit trials. One disadvantage of using the memetic learning approach is an increase in rule-set size. This is clearly at odds with the requirement for a compact rule-set. However, using the self-adaptive mechanism seems to reverse the bloating effects of the standard memetic learning approach on rule-set size for both test environments.

Finally, experimental results for the decomposable six-dimensional test function were analysed region by region. Three different versions of the classifier system, *Non-Memetic*, *Memetic* and *Self-Adaptive Memetic*, were compared clearly showing that for all parameter settings the *Self-Adaptive Memetic* version is statistically equivalent to the strongest of the *Non-Memetic* and *Memetic* versions and is significantly ($> 95\%$) better in 1 or 2 regions. It was also shown that the *Self-Adaptive Memetic* version outperformed the other two by simply counting the number of regions for which each version achieved 70% or greater performance.

10 Mathematical Description of Test Environments

This section defines several related equations that can be used to define the two and six dimensional test environments used in this study. In order to estimate the fitness of a given sample point, a vector of real values is needed, where each value is within some pre-defined interval. For the two dimensional test environment, $x_1, x_2 \in [-6.0, 6.0]$, and for the six dimensional test environment, $a \ldots f \in [0.0, 1.0]$.

10.1 Two-dimensional Test Environment

The two-dimensional test environment is the multi-modal modified Himmelblau function defined by the equation:

$$f(x_1, x_2) = 200 - \left(x_1^2 + x_2 - 11\right)^2 - \left(x_2^2 + x_1 - 7\right)^2$$

10.2 Six-dimensional Test Environment

The six-dimensional test environment is defined by the additive effect of three different two-dimensional planes. Each plane has an associated "local" fitness value and the "global" fitness value of the six-dimensional function is defined by adding each of these "local" fitness values together:

$$fitness_{global} = fitness_{plane1} + fitness_{plane2} + fitness_{plane3}$$

Plane 1

$$z_1 = \frac{0.41}{\frac{(0.8-a)^2}{0.04} + \frac{(0.7-b)^2}{0.04} + 1}$$

$$z_2 = \frac{0.42}{\frac{(0.2-a)^2}{0.0225} + \frac{(0.75-b)^2}{0.09} + 1}$$

$$z_3 = \frac{0.44}{\frac{(0.4-a)^2}{0.2025} + \frac{(0.05-b)^2}{0.09} + 1}$$

$$fitness_{plane1} = \min \begin{cases} 0.5 \\ z_1 + z_2 + z_3 \end{cases}$$

Plane 2

$$z_1 = 0.5 - \frac{0.5}{\frac{(0.25-c)^2}{0.09} + \frac{(0.75-d)^2}{0.09} + 1}$$

$$z_2 = 0.5 - e^{(\cos(48\pi c)+\cos(48\pi d))/14.8}$$

$$z_3 = 0.5 - \frac{0.5}{\frac{(0.75-c)^2}{0.09} + \frac{(0.25-d)^2}{0.09} + 1}$$

$$z_4 = 0.5 - e^{(\cos(24\pi c)+\cos(24\pi d))/14.8}$$

$$z_5 = \max \begin{cases} 0.0 \\ z_2 - z_1 \end{cases}$$

$$z_6 = \max \begin{cases} 0.0 \\ z_4 - z_3 \end{cases}$$

$$z_7 = \begin{cases} z_5 & \text{if } (c < 0.5) \text{ AND } (d > 0.5) \\ 0.0 & \text{otherwise} \end{cases}$$

$$z_8 = \begin{cases} z_6 & \text{if } (c > 0.5) \text{ AND } (d < 0.5) \\ 0.0 & \text{otherwise} \end{cases}$$

$$z_9 = \begin{cases} 0.0 & \text{if } (c < 0.5) \text{ AND } (d < 0.5) \\ \frac{0.35}{\frac{(0.75-c)^2}{0.09} + \frac{(0.75-d)^2}{0.09} + 1} & \text{otherwise} \end{cases}$$

$$z_{10} = \begin{cases} 0.0 & \text{if } (c > 0.5) \text{ AND } (d > 0.5) \\ \dfrac{0.35}{\frac{(0.25-c)^2}{0.09} + \frac{(0.25-d)^2}{0.09} + 1} & \text{otherwise} \end{cases}$$

$$fitness_{plane2} = z_7 + z_8 + z_9 + z_{10}$$

Plane 3

$$z_1 = \min \begin{cases} 0.5 \\ 1.8e^2 + 3f^2 \end{cases}$$

$$z_2 = \begin{cases} 0.0 & \text{if } (e > 0.6) \text{ OR } (f > 0.5) \\ z_1 & \text{otherwise} \end{cases}$$

$$z_3 = \begin{cases} 0.625f & \text{if } (f < 0.8) \\ 0.5 & \text{if } (f \geq 0.8) \text{ AND } (f < 0.85) \\ \frac{10(1-f)}{3} & \text{if } (f \geq 0.85) \end{cases}$$

$$z_4 = \begin{cases} z_3 & \text{if } (e > 0.6) \\ 0.0 & \text{otherwise} \end{cases}$$

$$z_5 = \min \begin{cases} 0.5 \\ z_4 \end{cases}$$

$$z_6 = \min \begin{cases} 0.5 \\ 1.5(0.6 - e) \times 2.5(f - 0.5) \end{cases}$$

$$z_7 = \begin{cases} z_6 & \text{if } (e \leq 0.6) \text{ AND } (f > 0.5) \\ 0.0 & \text{otherwise} \end{cases}$$

$$fitness_{plane3} = z_2 + z_5 + z_7$$

Acknowledgements

The first author is funded by the Engineering and Physical Sciences Research Council. The authors would also like to thank Professor Ian Parmee for his continued support.

References

1. Bäck T (1992) Self-Adaptation in Genetic Algorithms. In: Proceedings of the First European Conference on Artificial Life. MIT Press, Cambridge. pp. 263–271
2. Baldwin J (1896) A New Factor in Evolution. American Naturalist 30: 441–451
3. Beasley D, Bull D, Martin R (1993) A Sequential Niche Technique for Multimodal Function Optimisation. Evolutionary Computation, 1(2): 101–125
4. Bernadó E, Llorà X, Garrell J (2001) XCS and GALE: a Comparative Study of Two Learning Classifier Systems with Six Other Learning Algorithms on Classification Tasks. In: Lanzi PL, Stolzmann W, and Wilson SW (Eds.), Advances in Learning Classifier Systems. Fourth International Workshop (IWLCS-2001), Lecture Notes in Artificial Intelligence (LNAI-2321). Springer-Verlag:Berlin, pp. 115–133
5. Blake C, Merz C (1998) UCI Repository of Machine Learning Databases. University of California, Irvine
   Available at `http://www.ics.uci.edu/~mlearn/MLRepository.html`
6. Bonham C (2000) Evolutionary Decomposition of Complex Design Spaces. PhD Thesis, University of Plymouth
7. Bonham C, Parmee I (1999) An Investigation of Exploration and Exploitation Within Cluster-Oriented Genetic Algorithms (COGAs). In: Banzhaf W, Daida J, Eiben A, Garzon M, Honavar V, Jakiela M, Smith R (Eds), Proceedings of the Genetic and Evolutionary Computation Conference 1999, Morgan Kaufmann, pp. 1491–1497
   Available at `http://www.ad-comtech.co.uk/Parmee-Publications.htm`
8. Bull L, Wyatt D, Parmee I (2002) Initial Modifications to XCS for use in Interactive Evolutionary Design. In: Merelo J, Adamidis P, Beyer HG, Fernandez-Villacanas JL, Schwefel HP (eds) Parallel Problem Solving From Nature - PPSN VII, Springer Verlag, pp. 568–577
9. Butz M, Wilson S (2001) An Algorithmic Description of XCS. In: Lanzi PL, Stolzmann W, Wilson SW (Eds.), Advances in Learning Classifier Systems. Third International Workshop (IWLCS-2000), Lecture Notes in Artificial Intelligence (LNAI-1996). Springer-Verlag: Berlin, pp. 253–272
10. Holland J (1975) Adaptation in Natural and Artificial Systems, The University of Michigan Press, Ann Arbor
11. Holland J (1986) Escaping Brittleness: the Possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-based Systems. In: Michalski RS, Carbonell JG, Mitchell TM (Eds.), Machine Learning, An Artificial Intelligence Approach. Morgan Kaufmann: Los Altos, California
12. Holmes J (1996) A Genetics-Based Machine Learning Approach to Knowledge Discovery in Clinical Data. Journal of the American Medical Informatics Association Supplement 883
13. Japkowicz N, Stephen S (2002) The Class Imbalance Problem: A Systematic Study. Intelligent Data Analysis, Volume 6(5): 429–450
14. Kaelbling L, Littman M, Moore A (1996) Reinforcement Learning: A Survey. Journal of Artificial Intelligence Research 4: 237–285
15. Kocis L, Whiten WJ (1997) Computational Investigations in Low Discrepancy Sequences, ACM Transactions on Mathematical Software, 23(2): 266–294
16. Kohavi R, Provost F (1998) Glossary of Terms. Machine Learning 30: 271–274

17. Kononenko I, Bratko I (1991) Information-Based Evaluation Criterion for Classifier's Performance. Machine Learning 6: 67–80
18. Krasnogor N, Smith J (2000) Memetic Algorithms: Syntactic Model and Taxonomy. Technical Report, Intelligent Computer Systems Centre, University of the West of England, Bristol, U.K. in 2000 and Automated Scheduling, Optimisation and Planning Group, University of Nottingham, U.K. in 2002
19. Kubat M, Holte R, Matwin S (1997) Learning when Negative Examples Abound. In: Proceedings of the Ninth European Conference on Machine Learning, LNCS 1224, Springer-Verlag, pp. 146–153
20. Lewis D, Gale W (1994) A Sequential Algorithm for Training Text Classifiers. In: Proceedings of SIGIR-94, 17th ACM International Conference on Research and Development in Information Retrieval, ACM/Springer, pp. 3–12
21. Ling C, Li C (1998) Data Mining for Direct Marketing: Problems and Solutions. In: Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-98), AAAI, pp. 73–79
22. Morgan C (1896) On Modification and Variation, Science 4: 733–740
23. Moscato P (1989) On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts - Towards Memetic Algorithms. Technical Report 826, California Institute of Technology, Pasadena, California.
24. Osborn H (1896) Ontogenic and Phylogenic Variation, Science 4: 786–789
25. Parmee I (1996) The Maintenance of Search Diversity for Effective Design Space Decomposition using Cluster-Oriented Genetic Algorithms (COGAs) and Multi-Agent Strategies (GAANT). In: Proceedings of 2nd International Conference on Adaptive Computing in Engineering Design and Control, PEDC, University of Plymouth, pp. 128–138
Available at `http://www.ad-comtech.co.uk/Parmee-Publications.htm`
26. Stone C, Bull L (2003) For Real! XCS with Continuous-Valued Inputs. Evolutionary Computation 11(3): 299–336
Also available at `http://www.cems.uwe.ac.uk/lcsg`
27. Swets JA (1988) Measuring the Accuracy of Diagnostic Systems. Science 240: 1285–1293
28. Weiss G, Provost F (2001) The Effect of Class Distribution on Classifier Learning: An Empirical Study. Technical Report ML-TR-44, Department of Computer Science, Rutgers University
29. Whitley D, Gordon S, Mathias K (1994) Larmarckian Evolution, the Baldwin Effect and Function Optimization. In: Davidor Y, Schwefel H, Manner R (Eds.) Parallel Problem Solving From Nature - PPSN III, Springer-Verlag, pp. 6–15
30. Wilson S (1995) Classifier Fitness Based on Accuracy. Evolutionary Computation 3(2): 149–175
31. Wilson S (2000) Get real! XCS with Continuous-valued inputs. In: Lanzi PL, Stolzmann W, Wilson SW, (Eds.) Learning Classifier Systems. From Foundations to Applications Lecture Notes in Artificial Intelligence (LNAI-1813) Springer-Verlag: Berlin, pp. 209–222
32. Wilson S (2001) Compact Rulesets for XCSI. In: Lanzi PL, Stolzmann W, Wilson SW, (Eds.) Advances in Learning Classifier Systems. Fourth International Workshop (IWLCS-2001), Lecture Notes in Artificial Intelligence (LNAI-2321). Springer-Verlag: Berlin, pp. 197–210
33. Wilson S (2001) Mining Oblique Data with XCS, In: Lanzi PL, Stolzmann W, Wilson SW, (Eds.) Advances in Learning Classifier Systems. Third Interna-

tional Workshop (IWLCS-2000), Lecture Notes in Artificial Intelligence (LNAI-1996). Springer-Verlag: Berlin, pp. 158–177
34. Wyatt D, Bull L (2003) Using XCS to Describe Continuous-Valued Problem Spaces. Technical Report UWELCSG03-004.
Available at `http://www.cems.uwe.ac.uk/lcsg`

Angels & Mortals: A New Combinatorial Optimization Algorithm*

Francesc Comellas and Ruben Gallegos

Departament de Matemàtica Aplicada IV, Universitat Politècnica de Catalunya, Avda. Canal Olímpic s/n, 08860 Castelldefels, Barcelona, Catalonia, Spain.
comellas@mat.upc.es

Summary. In this paper we present a new optimization algorithm based on the collective behavior of a set of agents which encode the problem to be solved. We compare its performance with a standard genetic algorithm on a classical difficult problem, the k-coloring of a graph. The results show that this new algorithm is faster and outperforms a standard genetic algorithm for a range of random graphs with different sizes and densities. Moreover, it might be easily adapted to solve other NP-complete problems providing a new efficient tool to deal with difficult combinatorial problems.

1 Introduction

An adaptive complex system consists of a large number of interacting units where different processes of learning, change and selection take place; these processes are often driven by information obtained from the environment. In some way, and by mechanisms which we are just starting to understand, these systems discriminate relevant details from random noise and use this information in a collective way. Thus, in an adaptive complex system the global behavior does not depend only on the individual features of its parts, but also on its structure and on the different sort of relations which can be established between them and the environment

A swarm of bees, an ant colony, companies which supply a big city (water, electricity, telephone, etc.), are all complex systems where global patterns emerge from the interaction of a large number of similar elements. Furthermore, this global behavior allows the different systems to attain certain achievements without the presence of an administrative hierarchy or a central control mechanism.

* Research supported by the Ministry of Science and Technology, Spain, and the European Regional Development Fund (ERDF) under projects TIC2001-2171 and TIC2002-00155.

Very often the macroscopic behavior of these adaptive complex systems is studied by introducing equations to describe collectively the microscopic components according to continuous density distributions. However, it has turned out that the opposite is true, the consideration of the individual parts of the system in a discrete way leads to the right explanation of the macroscopic features. This is precisely the approach that Shnerb, Louzon, Bettelheim and Solomon used in a simple model to prove that while the continuum equations would predict extinction of a certain population, the microscopic approach explains the existence of localized subpopulations with collective adaptive properties that allow their survival and further development. Moreover, they show this happens when the relations among the parts occur essentially in two dimensions.

Although we believe that some of these concepts are implicit in certain optimization algorithms, we decided to translate it explicitly into a combinatorial optimization algorithm and test if the "prevalence of life" found by the authors of [11] helps to drive the algorithm towards a quasi-optimal solution of the problem to solve. We call the new method *angels & mortals*, like the game or computer simulation that they use to test the results in their paper. We give a detailed description in Section 3.

The problem which we have considered to test our implementation is the of a graph. This is known to be an NP-complete optimization problem [9] and finding an optimal solution is a computationally hard task, but there exist efficient algorithms to find quasi-optimal solutions as, for example, simulated annealing, genetic algorithms or ant colony based systems. All these general combinatorial optimization methods are used to obtain an acceptable answer in a reasonable time, see [1, 2, 6, 8].

In Section 2 we give a short introduction to the terminology used and a description of the genetic algorithm approach to the problem considered. In Section 3 we describe the motivation and general aspects of our implementation of the *angels & mortals* algorithm. In Section 4 , we present the details of the implementations and the results obtained and in Section 5 we discuss briefly the interest of this new combinatorial optimization method and a possible extension of the algorithm to other problems.

2 Graph coloring and combinatorial optimization methods

A *proper coloring* of a graph $G = (V, E)$ is a function from the vertices of the graph to a set C of *colors* such that any two adjacent vertices have different colors. If $|C| = k$, we say that G is *k-colored.* The minimum possible number of colors for which a proper coloring of G exists is called the *chromatic number* $\chi(G)$ of G. The problem of finding the chromatic number and a proper coloring of a graph is of great interest for its widespread applications in areas such as scheduling and timetabling and particularly in frequency assignment in radio

networks [2, 3, 12]. As many other problems in graph theory, it is an NP-complete problem [9]. Efficient algorithms for this problem are known only for particular graphs, e.g. triangle-free graphs with maximum degree three [4].

When exact methods are not possible, sometimes it is sufficient to obtain an approximate solution with a fast and easy to implement method. This is the case of simulated annealing, genetic algorithms, neural networks, ant colony based systems, etc.

To implement any of these optimization methods we need a way to encode the problem which has to be solved, and a system to quantify the "goodness" of a solution. In the case of k-coloring, a possible solution may be encoded using a list such that each position is associated to a vertex of the graph and its value to a color. The cost function simply counts the number of times that an edge joins vertices with the same color.

We will compare our technique, *angels & mortals*, with a genetic algorithm because the latter is the method that, although different in concept, has more aspects in common. Both algorithms, for example, consider a set of individuals or population which evolves over time. However in our method the *physical* distance between individuals, and their relation with the environment plays a role which does not exist in a standard genetic algorithm. We would like to emphasize that, for graph coloring problems, other methods, like simulated annealing or ant colony systems, are more efficient than this genetic algorithm or the *angels & mortals* algorithm introduced here. The aim of our paper is to fully describe this new optimization technique and compare it with equivalent methods. Further tests with standard benchmarks will be needed to assess if *angels & mortals* performs better than other algorithms for some specific combinatorial optimization problems.

In a standard genetic algorithm, see [7, 10], the starting point is a collection of possible solutions generated at random, known as *population.* A suitable encoding of each solution in the population is used to compute its *fitness* through a *cost function.* At each iteration a new population, or *generation*, is obtained by *mating* the best of the old solutions with one another. To create the next generation, new solutions are formed through *selection*, *crossover* and *mutation.* In our implementation we rank the individuals according to their fitness and select the best to form a *parent pool* used to obtain a new generation. The solutions that will be considered for crossover are probabilistically selected according to the fitness values from this parent pool. Crossover creates two new *child* solutions from two solutions sampled from this pool. In this way, fitter parents have a better chance of producing children. The process is repeated until a new population with the same size as the original is generated. Children solutions are obtained by interchanging random parts of their parents (i.e. fragments of the corresponding lists). Some randomness is also introduced through the mechanism called *mutation* to ensure that the algorithms avoid getting stuck at local minima. Mutation changes selected parts of a solution (for example a value in the list is replaced

by a new one). The crossover and mutation operations are done with fixed probabilities, thus ensuring that some solutions from the current generation will be kept in the new generation.

Once a new generation is created, the fitness of all solutions is evaluated and the best solution is recorded. The process is repeated until either the results stabilize or the optimal solution, when it can be identified, is reached.

Therefore the main aspects to decide in the genetic algorithm are the representation of the solutions, the cost function and the crossover and mutation operators. Important parameters are the population size, the size of the parent pool, and the probabilities of crossover and mutation.

3 The *angels & mortals* algorithm

The aim of the algorithm is to associate possible solutions for a given problem to the individuals of an artificial world that evolves according to certain rules. Like in other algorithms a fitness is associated to the quality of the solution but in this algorithm the fitness is also tied to the lifespan of the evolving individual.

We have chosen the *angels & mortals* model of Shnerb, Louzon, Bettelheim and Solomon [11] as their research shows that it is a very simple that explains the behavior of complex ecological systems. They distribute randomly a certain number of *mortals* over a torus grid. These individual have a given lifespan and at each clock tick it is reduced by one unit. On the other hand, there are also a few eternal agents, or *angels*, scattered over the same board. The mortals and angels move randomly from cell to cell of the grid. There is one simple rule: when a mortal meets an angel the mortal is cloned to a near place. They wondered what will be the evolution of this world, and the interesting result is that this depends on the way of looking at it. Given average population densities of angels and mortals, it is easy to write an equation that predicts the average death and birth rates. Under some conditions this continuum approach predicts the extinction of the mortals. However a computer exact simulation at the individual level leads to a totally different outcome. Although there is an initial reduction of the mortals population, later this recovers. This contradiction between the continuum and discrete approaches is explained by the adaptive behavior of the mortals. When some of them meet an angel new births take place in its neighborhood and the overall mortal population increases at these sites. The result is clouds of mortals moving around following their angels. Clouds are unstable as they grow, split up and join again, but because of this apparently adaptive behavior, the population of mortals survive. However the individual mortals have no explicit rules other than they duplicate in front of an angel. In that aspect they differ from standard adaptive agents with complex rules embedded in them. Here a nonadaptive individual produces an adaptive global world.

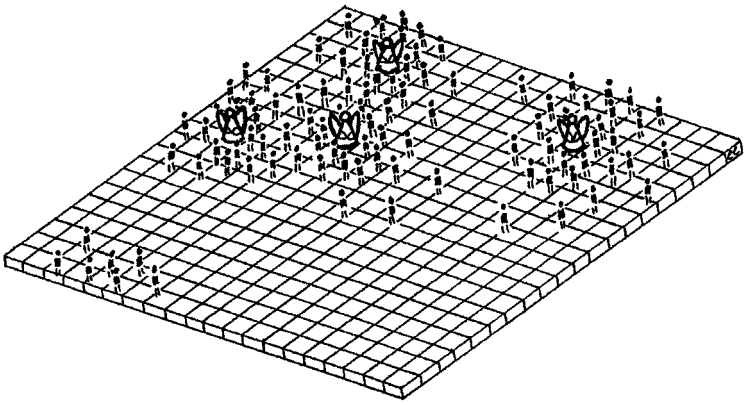


Fig. 1. A representation of the world with angels and mortals. After some generations, mortals form clouds that follow the angels.

In our implementation we construct first a toroidal world of $n \times m$ cells where each cell may be empty or contain an angel or a mortal. A certain number of angels and a larger number of mortals are assigned at random to different cells of this world. We then read the adjacencies of the graph to be colored and the number of colors that the algorithm will consider. The next step consists of generating as many random solutions (lists of colors, such that each list position is associated to a vertex of the graph) as mortals are in the world. The fitness of each solution is calculated (number of edges which join vertices with different colors) and, according to this fitness a is assigned to the corresponding mortal. A better fitness translates into a longer lifespan. A generation consists of moving, if possible, to one of eight near cells each of the angels and mortals, clone a mortal if it happens to be near an angel, decrease one unit the life counter of all mortals and eliminate those mortals that reach zero life and finally mutate all mortals. The reaper also eliminates those mortals that have 40 percent or more of the associated edges joining vertices with the same color. The process is repeated until a solution to the problem is found or a predefined maximum number of generations has elapsed. We discuss now briefly the main operators of this algorithm.

Mutation. It is exactly the same operator as in a GA, but while the probability of mutation for a GA is small, in the angels & mortals algorithm there is always mutation. Mutation considers a list position at random and replaces the current color by the best possible color, it this exists. If that is not possible, then no change is performed. It would seem that this sort of mutation, the only source of change in the system, would drive the fitness to a local minimum, but the presence of angels and the cloning process produces enough diversity to avoid minima as different clones of the same mortal can evolve

```
Angels &Mortals Algorithm():
Begin
  Set MaxGenerations;
  Initialize an n × m world with A angels and M mortals.;
  Associate a random solution to each mortal;
  Assign the lifespan of each mortal accordingly to their solution fitness;
  Repeat Until ( currentGeneration < MaxGenerations ) Do
    Look for the best mortal;
    If (this mortal solution is the global optimum) Then
      Report solution and exit algorithm;
    endIf
    Decrease life counters;
    Reap mortals;
    Move randomly all angesl and mortals to a neighbor cell;
    Clone mortals near angels ;
    Mutate mortals;
    Recalculate fitnesses and assign new lifespans;
    Increment currentGeneration;
  endDo
End.
```

Fig. 2. A basic version of the Angles & Mortals Algorithm.

very differently and explore other paths of the state space. In the last section we will discuss other variations of the mutation operator.

Cloning. When a mortal encounters an angel, the cloning process produces a new mortal and puts it in the first available cell starting by the cell just up and checking clockwise the eight cells around it. This simple rule helps to the survival of the best individuals. Because good individuals have a longer life they have also more chances to meet an angel. We have also tested a variation in which the encounter of a mortal with an angel does produces a clone but increases its lifespan.

Reaper. After moving a mortal, cloning it -if he is near an angel- and mutating it, its life is decreased by one unit. When it reaches 0, the individual is removed from the world. We see that the association of fitness to lifespan is crucial for the right convergence of the algorithm and is also directly related with the world size. If life is too long the world could become overcrowded. Set too short a life and the mortal population will disappear. We have introduced also a reaper mechanism that kills an individual which have more than 40 percent of its edges joining vertices with the same color. The reaper helps to improve the performance of the algorithm as there are less individuals to compute and it creates empty cells for clones.

In the next section we will give the values considered for our implementation

4 Results

For our study, a large number of instances of the problem to be solved have been generated to test and compare the performances of a standard genetic algorithm and the new *angels & mortals* algorithm. We use random graphs of orders ranging from 30 to 200 vertices and densities of 5%, 15% and 20% (the density of a graph is the ratio between the number of edges that actually has the graph and the maximum number that may contain). For each case 20 simulation runs were performed.

All simulations were programmed in C (less than 500 lines) and executed on a PC (AMD K7 Athlon at 1411 MHz) under Windows Me.

Possible solutions have been coded as lists where each position represents a vertex of the graph and has values 0 to $k-1$ according to the color assigned to it. The cost function calculates the number of edges that do not allow in the associated graph a proper coloring and substracts this number from the size (total number of edges) of the graph.

The main parameters of the genetic algorithms are: *Population*= 200, *parent pool size*= 150, *crossing probability*= 0.9 and *mutation probability*= 0.001.

We have tested two different versions of the *angels & mortals* algorithm.

$A\&M_1$ is a implementation of the Shnerb, Louzon, Bettelheim and Solomon concept, see [11]. It starts with a toroidal 20×20 world with 25 angels and 5 mortals. The maximum number of mortals allowed is 200. When this value is reached, the cloning process does not act. The lifespan of a mortal is $100 \left\lfloor \frac{\text{fitness}}{\text{total number of edges}} \right\rfloor$. After a mutation the life is modified according to the change in the fitness.

$A\&M_2$ is a simple variation of the algorithm such that initially the population of mortals is set to 200 and there is no cloning. When a mortal encounters an angel, its lifespan is extended by a fixed amount (6 units).

Table 1 shows the results corresponding to graphs of orders 30,50,70 100 and 200 with edge densities of 5%, 15% and 25.

We have performed a wide range of experiments testing different world sizes, mutation operators, parameter values etc. In most cases the algorithm converges similarly or better than the genetic algorithm. In that sense, Table 1 does not represent runs corresponding to the best possible performance of the algorithms but just a complete set of experiments. Furthermore, and as it has been reported in Section 2, there are other methods more suitable for graph coloring problems. We have implemented, for example, a simple version of a algorithm (around 100 lines of code in C language) which finds solutions of a similar quality four to five times faster. In this paper, however, we decided to compare *angels & mortals* with a technique computationally equivalent. More tests considering other combinatorial optimization problems will help to determine the exact role that our new algorithm can play.

Table 1. k-coloring problem for graphs of orders 30,50,70 100 and 200 with edge densities of 5%, 15% and 25 %. Comparative values between *angels & mortals* algorithms and a standard genetic algorithm. ($A\&M_1$: *angels & mortals* with cloning; $A\&M_2$: *angels & mortals* with life extension; GA: genetic algorithm, see text for details).

		Colors			*Time*			*# Suc.*		
Vert	Edges	$A\&M_1$	$A\&M_2$	GA	$A\&M_1$	$A\&M_2$	GA	$A\&M_1$	$A\&M_2$	GA
30	22	2	2	3	0.01	0.06	0.01	18	19	20
50	61	3	3	3	0.09	0.03	0.08	20	14	11
70	120	3	3	4	0.19	0.13	0.14	3	10	16
100	247	4	4	4	1.35	0.25	1.14	2	13	1
200	995	9	7	8	2.95	0.80	2.15	3	5	1

		Colors			*Time*			*# Suc.*		
Vert	Edges	$A\&M_1$	$A\&M_2$	GA	A&M1	$A\&M_2$	GA	$A\&M_1$	$A\&M_2$	GA
30	65	3	3	4	0.10	0.04	0.02	4	13	19
50	184	5	5	5	0.44	0.11	1.14	15	20	8
70	362	6	6	7	0.47	0.25	0.30	9	17	7
100	742	8	8	9	1.89	0.38	0.48	2	12	1
200	2985	17	15	16	10.32	1.05	3.86	3	11	8

		Colors			*Time*			*# Suc.*		
Vert	Edges	$A\&M_1$	$A\&M_2$	GA	$A\&M_1$	$A\&M_2$	GA	$A\&M_1$	$A\&M_2$	GA
30	108	4	4	5	0.10	0.07	0.06	6	14	11
50	306	6	6	7	0.41	0.20	0.24	5	12	2
70	603	8	8	9	1.10	0.50	0.44	2	4	6
100	1237	12	11	12	1.71	0.45	0.94	7	10	9
200	4975	24	22	23	11.58	1.38	6.46	4	14	6

5 Discussion.

The model presented here opens a new range of optimization algorithms based on artificial life and other adaptive systems. The main idea is to associate possible solutions of a problem to individuals of a complex system and let it evolve.

The algorithm designed is very robust. We have tested worlds with sizes from 15 × 15 to 40 × 40, number of angels from 5 to 30, maximum number generations from 10.000 to 40.000, and a mutation mechanism accepting a decrease in the fitness (similarly to simulated annealing). In all cases the algorithm finds a solution of quality similar to a standard genetic algorithm.

We have restricted ourselves to a very simple model for these first tests and the results presented in the former section are encouraging.

Finally, the *angels and mortals* might be very easily adapted to solve other problems by considering the corresponding cost function and mutation mechanism. As a general conclusion, the results show that the algorithm performs better than a standard genetic algorithm even without fine tuning it. It is simple and easy to implement and can suggest other algorithms based on artificial life systems for which a problem is coded into an individual and the fitness of the corresponding solution is associated to a relevant characteristic of this individual, e.g. the lifespan. Moreover, the *angels and mortals* algorithm presented here, besides finding better solutions, runs much faster than a standard genetic algorithm and it might be also implemented on a parallel computer, thus improving further its performance.

References

1. Aarts E, Lenstra JK (1997) Local Search in Combinatorial Optimization. Wiley, Chichester New York
2. Abril J, Comellas F, Cortés A, Ozón J, Vaquer M (2000) IEEE Trans Veh Tech 49:1558–1565
3. Allen SM, Smith DH, Hurley S, (1999) Discrete Math 197/198:41-52
4. Bondy JA, Locke SC (1986) J Graph Theory 10:477–504
5. Costa D, Hertz A (1997) J Oper Res Soc 48:295–305
6. Duque-Antón M, Kunz D, Rüber B (1993) IEEE Trans Veh Technol 42:14–21
7. Goldberg DE (1989) Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading
8. Galinier P, Hao JK (1999) J Comb Optim 3:379–397
9. Garey MR, Johnson DS (1979) Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman, New York
10. Holland JH (1992) Scientific American 267:44–50
11. Shnerb NM, Louzoun Y, Bettelheim E, Solomon S (2000) Proc Natl Acad Sci USA 97:10322-10324
12. Smith DH, Hurley S (1997) Discrete Math 167/168:571–582

Index

Printing: Strauss GmbH, Mörlenbach
Binding: Schäffer, Grünstadt