Adapting Evolutionary Approaches for Optimization in Dynamic Environments

by

Abdunnaser Younes

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Systems Design Engineering

Waterloo, Ontario, Canada, 2006

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Many important applications in the real world that can be modelled as combinatorial optimization problems are actually dynamic in nature. However, research on dynamic optimization focuses on continuous optimization problems, and rarely targets combinatorial problems. Moreover, dynamic combinatorial problems, when addressed, are typically tackled within an application context.

In this thesis, dynamic combinatorial problems are addressed collectively by adopting an evolutionary based algorithmic approach. On the plus side, their ability to manipulate several solutions at a time, their robustness and their potential for adaptability make evolutionary algorithms a good choice for solving dynamic problems. However, their tendency to converge prematurely, the difficulty in fine-tuning their search and their lack of diversity in tracking optima that shift in dynamic environments are drawbacks in this regard.

Developing general methodologies to tackle these conflicting issues constitutes the main theme of this thesis. First, definitions and measures of algorithm performance are reviewed. Second, methods of benchmark generation are developed under a generalized framework. Finally, methods to improve the ability of genetic algorithms to efficiently track optima shifting due to environmental changes are investigated. These methods include adapting genetic parameters to population diversity and environmental changes, the use of multi-populations as an additional means to control diversity, and the incorporation of local search heuristics to fine-tune the search process efficiently.

The methodologies developed for algorithm enhancement and benchmark generation are used to build and test evolutionary models for dynamic versions of the travelling salesman problem and the flexible manufacturing system. Results of experimentation demonstrate that the methods are effective on both problems and hence have a great potential for other dynamic combinatorial problems as well.

Acknowledgements

I would like to extend my sincere gratitude to all people who have assisted me in completing this thesis. In particular, I would like to thank the following individuals.

Thanks to Professor Otman Basir and Professor Paul Calamai for supervising this work and for their willingness to allow it to develop in the direction I set. I am sure they are as proud of this work as I am since they have contributed tremendously to its accomplishment. Moreover, their encouragement and financial support for attending several international conferences have enabled me to present and publish my work.

Thanks to the members of my thesis committee who have taken the trouble to read the thesis and provide valuable comments on my work.

Thanks to my colleagues and the staff at our department for the friendly atmosphere, with special thanks to the smiling Vicky Lawrence for being there to help all the time.

I would like also to thank Sue McCarter, Brab Martin and Janet Dunlea of The Canadian Hearing Society in Kitchener, and Rose Padacz and Ildiko Denes of The Office for Persons with Disabilities in The University of Waterloo, for their unlimited support.

I am thankful for the financial support from The NSERC Postgraduate Scholarship, The Ontario Graduate Scholarship, The UOW President's Graduate Scholarship, and The Systems Design Department Graduate Scholarships.

Thanks to the researchers who pioneered the field of evolutionary dynamic optimization for inspiring my work, with special thanks to those anonymous reviewers who had provided valuable comments on my published work.

Thanks to Professor Shawki Areibi who roused up my interest in dynamic optimization, and with whom I have shared many fruitful discussions. Above all, I am most grateful for his friendship.

Finally, I am indebted most of all to my wife and daughters for their patience and affection, and to my parents and extended family in Libya who think of any accomplishment of mine as their own.

Abdunnaser Younes
Waterloo, 18th October 2006

To MY BIGGEST FANS

Mohamed and Fatma,
Nadia,
Eslam, Asma, and Ayat

of TAGHERBOOST

that stubborn place,
home of the people whom
I hold the highest esteem

Contents

List of Tables

List of Figures

List of Acronyms

ADM	adaptive diversity model	137
AHDM	adaptive hybridized diversity model	154
AHIM	adaptive hybridized islands model	154
AIM	adaptive islands model	144
ANOVA	analysis of variance	144
BG	benchmark generator	47
COP	combinatorial optimization problem	1
CBP	current best performance	61
DCAGA	diversity-controlling adaptive genetic algorithm	137
DGEA	diversity-guided evolutionary algorithm	137
DS	dynamic solver	46
ECM	edge change mode	110
EA	evolutionary algorithm	21
FM	fixed model	111
FMS	flexible manufacturing systems	5
$\mathcal{GD}$	genotypic diversity	125
GA	genetic algorithm	23
GLS	guided local search	19
GRASP	greedy randomized adaptive search procedure	20
IDM	insert/delete mode	164

Chapter 1

Introduction

The last fifteen years have seen a growing interest in the use of evolutionary algorithms to solve dynamic optimization problems. A substantial portion of existing research targets continuous optimization problems, while a much lesser portion is directed to combinatorial problems even though many real-world problems are both discrete and time-dependent. Moreover, dynamic combinatorial problems are often discussed in an application based context. However, many issues related to dynamism can be addressed in a general way despite variations in the structure of the underlying problem.

1.1 Motivation

Many optimization problems of practical as well as theoretical importance seek to find an optimal arrangement, grouping, ordering or selection of discrete objects often finite in number. These problems are grouped into a class known as combinatorial optimization problems (COPs). The importance of this class can be sensed from its broad range of applications in engineering, operation research, and social sciences. Under the subclass of the vehicle routing problem (VRP), we find the routing of passenger cars to transport

seniors and disabled, the routing of mail and delivery trucks to customers, the routing of automated vehicles to transport work in progress and hazardous items between work stations in a factory. Under the subclass of scheduling problems, we find job shop scheduling (JSS) in plants, maintenance planning in airline companies, and aircraft crew scheduling, to name a few. Examples of other applications include timetabling, flexible manufacturing systems, and facility location planning. Many of these applications are interrelated as well. For example, the relatively new discipline *supply chain management* encompasses facility location planning and network design, transportation and vehicle routing, product design and development, warehouse management, E-commerce, E-logistics, E-manufacturing, and more.

Efficient routes or schedules are beneficial not only to the individual organization, but to the whole country as well. For example, the U.S. National Council of Physical Distribution Study shows that transportation costs can be as high as 15% of the total gross national product [Goetschalckx et al. 1999]. Indeed, benefits from optimizing transportation and manufacturing processes are not limited to reducing the monetary costs. In the long run, efficient utilization of the country's resources and energy makes substantial contributions towards improving air quality and other environmental conditions.

In addition to their practical importance, COPs can present two challenges: NP-hardness and dynamism. The overwhelming majority of COPs is hard to solve [Rardin 1998], known as the NP-hard class. Exact algorithms that solve NP-hard problems to optimality need exponential time growth in the worst-case and often can only solve small problems in reasonable time. Therefore, for NP-hard problems with real world complexity, non-exact algorithms and metaheuristics, especially in the case of problems of substantial size, are often the method of choice. With the advances made in computer technology and science, it is becoming possible to tackle larger and larger COPs. However, other advances in computer and communication technology are behind the emergence of newer versions of COPs that are more complex and larger in size than their predecessors. In particular, as

real-time information and communication systems become increasingly available and the processing of real-time data becomes increasingly affordable, more and more new versions of highly dynamic real-world applications are created. In such applications, information on the problem is not completely known a priori, but instead is revealed to the decision maker progressively with time. Consequently, solutions to different instances of a typical dynamic problem have to be found as time proceeds, concurrently with the incoming information.

With the static versions of most COPs hard to optimize, one would expect the presence of time and the associated uncertainty in the dynamic versions to increase problem complexity [Weicker and Weicker 1999], making the dynamic version even harder to solve than its static counterpart. However, environmental changes in real life typically do not alter the problem completely but affect only some part of the problem at a time. For example, not all vehicles break down at once, not all pre-made assignments are cancelled, weather changes affect only parts of roads, any other events like sickness of employees and machine breakdown do not happen all at once. Thus, after an environmental change, there remains some information from the past that can be used for the future. Such problems call for a methodology to track their optimal solutions through time. The required algorithm should not only be capable of tackling combinatorial problems but should also be adaptive to changes in the environment. Evolutionary Algorithms (EAs) exhibit a number of potential advantages for such purposes.

We first note that EAs are one of the most commonly used metaheuristics for solving combinatorial problems as they proved to be effective solvers for a broad range of static problems [Pham and Karaboga 2000]. Second, there are several characteristics inherent and attributed to EAs that encourage their use for dynamic problems (see Figure 1.1 for the growing interest in the subject). The underlying principle of EAs is based on natural evolution, and hence they are expected to be capable of adaptation to environmental changes. In addition, EAs have proved to be suitable for "noisy" environments [Mitchell 1996] due to their ability to exploit previous or alternate solutions. One of the most

appealing features for dynamic environments is that, at any given instant, EAs deal with a population of solutions rather than a single solution. Hence, even if the environment changes, it is likely that some solutions in the population remain feasible and retain some of their good quality. Furthermore, EAs have often proved to be easy to hybridize with local search techniques. Thus, by using EAs, it is possible to formulate general techniques to address the dynamic issues, while leaving issues of the particular (static) problem to already developed relevant local search and repair heuristics.

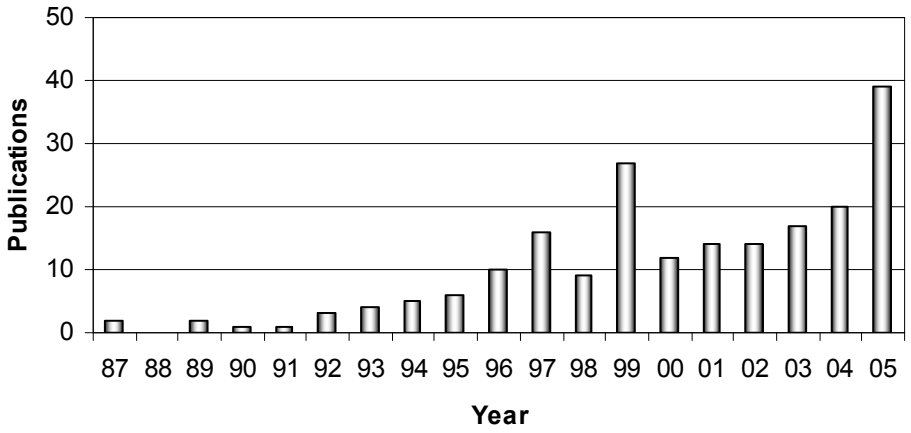


Fig. 1.1: Publications on evolutionary optimization in dynamic environments

One final point worth mentioning is that developing a potentially good algorithm is not enough; there should be some appropriate benchmarks to demonstrate its performance. This need is widely appreciated in evolutionary optimization [Whitley et al. 1996; Holland 2000]. However, in dynamic optimization, a problem is characterized, in part, by scenarios postulating a sequence of events. Thus, one is faced with the additional difficulty of incorporating the patterns of environmental changes in the dynamic benchmark. Fortunately, there is a wealth of benchmarks for almost all static COPs. Therefore, research should be devoted to developing dynamic benchmarks out of the available problems.

1.2 Scope and objectives

This thesis is concerned with the use of evolutionary algorithms to tackle dynamic combinatorial problems. Although many real world problems can be viewed as dynamic we are interested only in those problems where the decision maker does not have prior knowledge of the complete problem, and hence the problem can not be solved ahead of time (see Section 4.2 for a formal definition of the dynamic problem being considered in this thesis).

The thesis revolves around two fundamental aspects of dynamic optimization: developing adaptive algorithms and generating benchmarks to test them. By using robust heuristics such as EAs on a representative problem such as the travelling salesman problem (TSP), the proposed approaches are expected to be easily generalized to a broad range of applications.

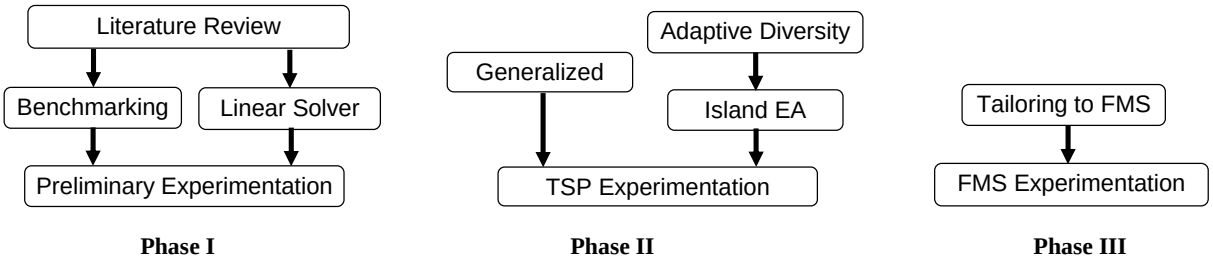


Fig. 1.2: Research progress

The research presented in this thesis has progressed in three phases, as illustrated in Figure 1.2. In the first phase, literature on the use of evolutionary algorithms to solve dynamic problems is surveyed. Several outstanding issues are identified and are outlined in Section 3.7. The second, the main phase, covers the development of adaptive algorithms and dynamic benchmarks on the dynamic TSP. Relevant issues are also illustrated on the knapsack problem and the flexible manufacturing systems (FMS) problem when deemed

necessary. The third phase aims to demonstrate the applicability of the developed ideas to more complicated test problems. A dynamic FMS problem is used for this purpose. In addition to its importance as a real-world problem, FMS can be used to represent dynamic COPs with multiple objectives.

This thesis has generated several contributions in the field of evolutionary optimization in dynamic environments: A framework for benchmark generation that treats the dynamic problem as a time sequence of static instances is proposed. Several issues that hinder benchmark generation in dynamic COPs are identified and addressed. A scheme to generate dynamic benchmarks with known optima, without the need of re-optimization is presented. Models of adaptive EAs are developed, including a model that employs time dependent genetic parameters, a diversity controlled model, a local search hybridized model, and an island model. These models employ novel methods to efficiently measure and control diversity throughout the search process, and schemes to adaptively regulate genetic parameters and efficiently embed local search heuristics. Experimental results on dynamic versions of FMS and TSP are presented to demonstrate the effectiveness of these models in improving solution quality with limited increase in computation costs. Detailed discussion of these contributions is given in Section 9.2.

1.3 Outline of the thesis

Chapter 2 gives a general overview on conventional techniques for solving a class of hard problems and concentrates on metaheuristics, with the main aim of identifying the relative benefits of using evolutionary algorithms to solve these hard problems. Readers familiar with the basics of metaheuristics can skip this chapter and go directly to Chapter 3 which contains a survey of how dynamic environments are tackled by EAs.

Chapter 4 is an overview of some basic concepts and measurements that are ambiguously defined in the literature. It also includes new definitions and their usage in the thesis. The

chapter ends by outlining the general scheme for experimentation and result reporting.

Chapter 5 addresses issues related to dynamically insignificant changes and misleading patterns of changes that can hinder the generation and the effectiveness of benchmarks. The chapter proposes a mapping based scheme for benchmark generation and presents a framework for dynamic benchmark generation for COPs.

Chapter 5 deals with benchmark generation. It addresses issues related to dynamically insignificant changes and misleading patterns, presents a mapping based scheme for benchmark generation, and proposes a framework for dynamic benchmark generation for COPs.

Chapter 6 details the preliminary experimentation carried out to investigate the merits of adapting solutions to changes, relative to restarting the algorithm after environmental changes.

Chapter 7 presents and investigates adaptive dynamic solvers that include a diversity controlling EA model, an island model, and two local search hybridized model.

The main goal of Chapter 8 is to demonstrate that the adaptive models and benchmarking schemes presented in this thesis can be applied to realistic problems. The dynamic solvers of Chapter 7 are compared on benchmarks developed for dynamic TSP and dynamic FMS.

Chapter 2

Background: Solving Hard Problems

the very essence of good GA design is retention of diversity, furthering exploration, while exploiting building blocks already discovered.

Holland [2000]

. . . schema theory tells us almost nothing about GA behaviour . . .

Vose [1999]

2.1 Introduction

This chapter consists mainly of elementary background material on metaheuristics and exact search methods. Readers familiar with the basics of metaheuristics and evolutionary algorithms can skip this chapter and go directly to the next one.

This chapter gives an introductory overview of methods commonly used for *hard* optimization problems. It starts with a quick introduction to the theory of computational complexity and commonly used techniques for accelerating enumeration, then it reviews metaheuristic methods that are used as alternatives to exact methods. At the end, the chapter focuses on traditional genetic algorithms, highlighting relevant theoretical and implementation aspects.

For a more thorough yet concise overview of metaheuristic methods the reader is referred to Blum and Roli [2003], and for implementation and code of simulated annealing, tabu search and genetic algorithms the reader is referred to Pham and Karaboga [2000]. More insight into evolutionary algorithms can be gained by consulting other references, such as Beasley et al. [1993a,b] for an introduction to GAs, Michalewicz [1992] and Mitchell [1996] for more details on genetic operators, Gen and Cheng [1999] for implementing GAs on Engineering problems, and Reeves and Rowe [2002] and Whitley [2001] for theoretical aspects.

2.2 Techniques of enumeration

Computational complexity is a measure used for quantifying the difficulty of a problem to an algorithm in an absolute sense. The work of Cook [1971] and that of Karp [1972] on complexity in discreet optimization have initiated the vast literature found today on complexity and profoundly affected all phases of research on discrete optimization [Parker and Rardin 1988].

It is common to describe computational complexity, relative to the worst case, as the greatest number of elementary operations executed by the algorithm to solve an instance of the problem. Accordingly, decision problems are classified in the P class if there exists a polynomial-time algorithm that can solve them. Problems which cannot be solved by a known polynomial-time algorithm but whose solutions can be verified by a polynomial time algorithm are classified among the *NP*. A problem p is considered in the *NP-complete* class (a subset of NP) if any problem in NP can be reduced to p in polynomial time. These classes are pictured in Figure 2.1, which reflects the common belief that P $\neq$ NP, although whether P=NP is still an open question.

This thesis concentrates on combinatorial problems that are known to be *NP-hard*. An NP-hard problem is an optimization problem that can be solved by a polynomial number of solutions of an NP-complete problem. Thus, one will generally expect an NP-hard problem to be harder than an NP-complete problem.

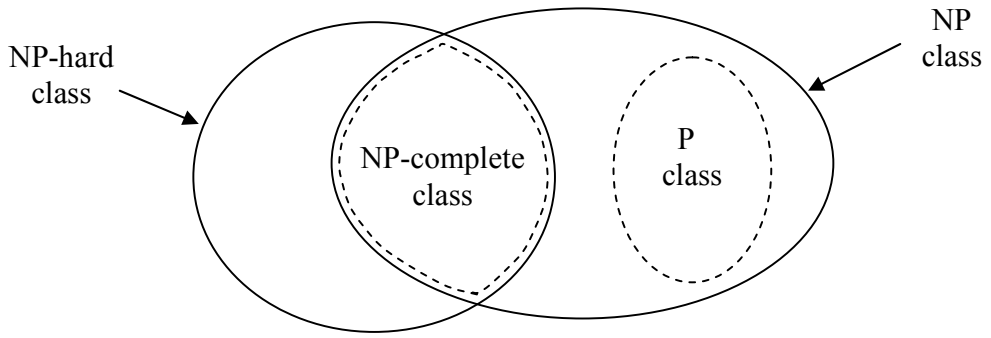


Fig. 2.1: Complexity sets.

In general, solving an *integer programming* (IP) problem can be much more difficult than solving a (continuous) *linear programming* (LP) problem. The difficulty is basically due to the integrality constraints of a COP where the feasible region is not convex—as in the case of LP—but is a lattice of points in the case of pure IP problems or a set of

disjoint line segments in the case of mixed integer programming problems. Thus, a COP is a multimodal problem in which global optimality can not be proved by the "conventional" derivative-based approaches.

Approaches used for COP can be divided into two general categories: exact and approximate. The proposed research targets metaheuristics which are approximate approaches. In principle, any combinatorial problem, whether NP-hard or not, can be solved by (exact) exhaustive search. Unfortunately, this naive approach is not a practical way to solve problems of realistic sizes[1], where the search space is forbiddingly large. There are at least two exact methods that are frequently used in addition to direct enumeration, namely branch and bound, and branch and cut.

2.2.1 Branch and bound

Branch and bound is a general method that aims to accelerate enumeration by ignoring some regions of the search space, because it has already been implicitly determined that those regions do not contain the optimal solution to the problem. The efficiency of this method depends on two key elements: an intelligent way to divide the search space into subregions and consequently the problem into subproblems (*branching*), and an efficient technique to find tight lower bounds (we consider a minimization problem) on the optimal solutions of the subproblems (*bounding*).

The most commonly used methods to compute lower bounds on the optimal value depend on *linear or Lagrangian relaxation.*

The *linear relaxation* of a linear integer (pure or mixed) subproblem is obtained by removing the integrality constraints in it. The resultant continuous linear programming problem will be easier to solve than the IP problem. However, bounds obtained from

[1]However, many organizations, due to the lack of sophisticated optimization tools, still use manual enumeration. Consequently, they are forced to be content with very low quality solutions. Often the objective of optimization is reduced to the trivial requirement of finding any feasible solution.

linear relaxations are often weak and can be arbitrary far from the optimal solution to the problem.

Lagrangian relaxation is a method that removes some of the inequality constraints rather than the integrality constraints. The constraints to remove are selected carefully so that after dropping them from the problem, the resulting problem though still an integer one is easier to solve than the original. At the same time, Lagrangian relaxation penalizes any violation of the removed constraints in an attempt to force feasibility. This relaxation method often produces bounds tighter than those of linear relaxation, but at the expense of increased computation time. In addition, the technique is problem dependent requiring sufficient understanding of the problem to determine which of the constraints are difficult and ought to be relaxed in order to make the problem easier to solve.

2.2.2 Branch and cut

Branch and cut is a hybrid of the branch and bound method described earlier and cutting plane methods. Cutting plane methods were first proposed by Gomory [1958], but they were not strong enough to tackle large problems and their algorithms were very slow to converge. Their success started in the early 1980's, after the development of the polyhedral theory [Hoffman and Padberg 1985, 1991, 1993; Padberg and Rinaldi 1991].

A cutting plane is an inequality that is added to the constraints of the LP relaxation in order to remove non-integer optimal solutions from the feasible region of the LP relaxation without removing any solution from the set of feasible solutions to the original problem. With the additional cutting plane, the solution to the resultant LP relaxation becomes closer to the optimal solution of the problem. Thus, the procedure is repeated until an optimal solution to the original IP problem is found.

Branch and cut has been successfully applied to several COPs; however, it is problem dependent and requires elaborate work to find the appropriate cuts for a given problem. Good introductions to branch and cut methods can be found in Grötschel and Holland

[1991] and Padberg and Rinaldi [1991].

Despite the great progress in exact methods, they can still take huge amounts of time to solve many COPs [Glover and Kochenberger 2003]. Furthermore, being highly problem dependent(relying heavily on identifying best ways to exploit the structure of the problem at hand), exact methods are harder to apply to dynamic problems that change unpredictably over time.

On the other hand, metaheuristics, though they may not guarantee optimal solutions, are less problem dependent and can give high quality solutions at low costs. Furthermore, metaheuristics are easy to implement, often requiring simple mathematical and algorithmic knowledge. An overview of some of these methods is given in the next section.

2.3 Metaheuristics, an overview

During the last two decades, metaheuristics moved to the forefront of optimization techniques gaining increasing popularity for solving complex optimization problems that arise in business, engineering, industry, and many other disciplines.

Nowadays, there are many metaheuristics in use either in their simple forms or hybridized with other optimization techniques or even with other metaheuristics. The following sections highlight their general characteristics and mechanisms, with some focus on important ones.

Voβ et al. [1999] define metaheuristics as:

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

This definition is by no means the only one. There are other definitions, such as those given by Osman and Laporte [1996], Stützle [1999], and Hromkovic [2001]. Although these definitions may seem different, they collectively outline certain common characteristics in metaheuristics that make them stand out among other optimization techniques:

- Metaheuristics are robust techniques. The same metaheuristic can be applied to a wide range of different problems, even if these problems have very different combinatorial structures. Their robustness combined with their ease of implementation is behind their popularity with many researchers.
- Metaheuristics are non-exhaustive and efficient; nevertheless, they do not guarantee the quality of their computed solutions.
- They guide the search in a systematic way, yet they take some random decisions from time to time in order to promote exploration of new regions of the search space.
- Metaheuristics often guide some subordinate heuristic that otherwise would very likely terminate with a low quality solution.
- They employ some essential mechanisms to escape local optima.
- They may use some combination of short-term, intermediate-term, and long-term memories to prevent revisiting old solutions, store good quality solutions, and guide the search towards new regions.

For a metaheuristic to be efficient, the algorithm must not search the solution space exhaustively. Yet, to produce high quality solutions it is also necessary to fine-tune the search. The conflict between both these requirements leads to a quality-efficiency tradeoff that is a central aspect in the design and application of all metaheuristics, commonly referred to as *diversification-versus-intensification balance*.

Metaheuristics can be classified in different ways. Blum and Roli [2003] show that the classification can be done according to the origin of the technique (nature inspired vs. non-nature), the number of solutions manipulated at each iteration (single solution vs. a population of solutions), the neighborhood or the objective function definition (constant, varying), and memory usage (with memory, without memory). For the purpose of this thesis, we adopt a classification from Hertz and Widmer [2003], which classifies metaheuristics according to their underlying principle into local search methods and population search methods.

2.4 Local search methods

Local search methods (also known as trajectory methods) build one (or more than one) initial solution then progressively enhance it by iteratively moving from the current solution to a neighbor solution. This group includes the well known *simulated annealing*, *tabu search*, *GRASP*, and *variable neighborhood* algorithms. Since these algorithms are basically modified versions of the classical *local search* technique, a quick review of local search is warranted.

The classical local search (LS) algorithm is based on the concept of exploring the vicinity of the current solution. It accepts solutions that produce reductions in objective functions (presuming a minimization problem), without making use of any information gathered during the execution of the algorithm. Thus, LS will always terminate at the nearest local optimal solution, which can be arbitrarily far from the global optimum. Consequently, the effectiveness of LS in practice depends heavily on factors that influence the trajectory of solution progression. These include:

The starting initial solution x_0. Only if the initial solution is in the basin of attraction of a local optimum, may the solution trajectory end at that local optimum.

The definition of neighborhood structure $\mathcal{N}(.)$. The very same initial solution can lead to different local optima if different neighborhood structures are used. Also, a local minimum according to some neighborhood structure may no longer represent a local optimum in another structure.

The Function to be minimized $f(.)$. Ruggedness and smoothness of the landscape of the search space is determined in the first degree by the objective function. Changing the definition of the objective function will change the relative attractiveness of different solutions to the LS heuristic.

2.4.1 Simulated annealing

Simulated annealing (SA) [van Laarhoven and Aarts 1985; Gelfand and Mitter 1985; Hajek 1988] can be viewed as a local search augmented with random decisions that lead the search out of local minima. While simple LS stops when improvements are not possible any more, SA accepts non-improving moves with a probability that favors the lesser deteriorating moves and hence it can climb up moderate hills to reach the basin of attraction of another local optimum, as shown in Figure 2.2. Nevertheless, the probability of accepting bad moves gradually decreases with the number of iterations until the algorithm eventually reduces to a simple local search.

Simulated annealing has the potential of finding feasible solutions of high quality that are not expected to be found through the use of a basic local search. This is because the quality of SA output does not depend entirely on the choice of initial solutions. However, substantial experimental work is necessary to tune the SA parameters for a particular problem. A more serious disadvantage with SA is that it can be inefficient because, in accepting bad moves, the algorithm may cycle back to old solutions.

2.4.2 Tabu search

Similar to SA, Tabu Search (TS) [Glover and Laguna 1997] will accept bad moves to escape local minima. However, TS makes use of a memory called the *tabu list* to store old solutions, or their attributes, in order to discourage moves towards previously visited regions of the search space, and hence avoids the greatest pitfall of SA. Currently, TS is among the most commonly used metaheuristics for COPs, especially routing problems. However, parameter tuning is a more difficult task in TS than in SA.

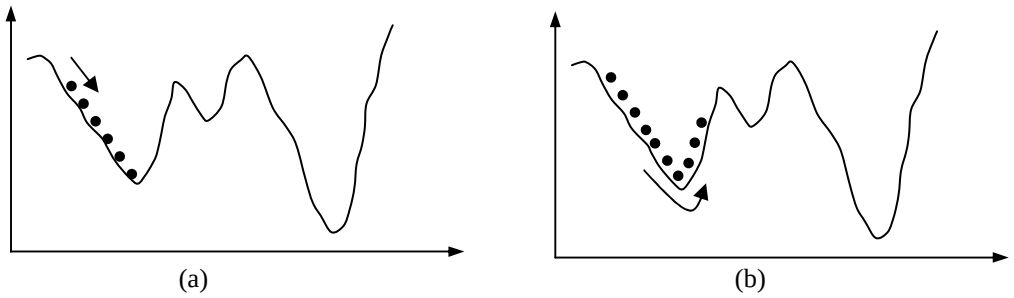


Fig. 2.2: Local search, simulated annealing and tabu search. (a) LS terminates at the first encountered local minimum. (b) SA and TS accept bad moves in order to escape a local minimum.

2.4.3 Iterated local search

Iterated local search (ILS) [Stützle 1999; Lourenço et al. 2001; Lourenco et al. 2002] employs a two-phase cycle: a local search and a solution *perturbation*. Once the local search phase ends at a local minimum, the perturbation phase disturbs the solution to produce a new solution, ideally, located in the basin of attraction of another local minimum (see Figure 2.3(a)). Repeating the ILS cycle causes the algorithm to move from one local optimum to another until it hopefully reaches the global minimum.

An important consideration with ILS is to balance the strength of perturbation to be

strong enough to escape the current local minimum but not so strong as to reduce the algorithm to a random restart LS.

2.4.4 Variable neighborhood search

Variable neighborhood search (VNS) [Hansen and Mladenović 1999; Hansen and Mladenovic 2001] and its variants adaptively change the definition of the neighborhood in order to escape local optima. In doing that, previous solutions deemed bad because they lead to inferior local optimum according to a certain definition of the neighborhood structure may lead to a better local optimum under another neighborhood definition (see Figure 2.3(b)).

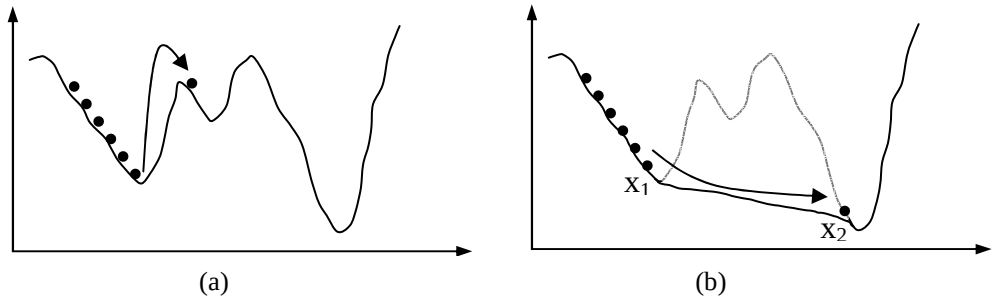


Fig. 2.3: Iterated local search and variable neighborhood search. (a) In ILS perturbation of local minimizer can move it to another basin of attraction. (b) After redefining the neighborhood structure, x_1 is no longer a local minimum, and $x_2 \in \mathcal{N}(x_1)$.

2.4.5 Guided local search

In Guided local search (GLS) [Voudouris and Tsang 1999], the objective function is changed adaptively. When a local minimum is reached, GLS gradually increases the objective function by penalizing some features to which the relative goodness of the current optimum is attributed. For example, in the traveling salesman problem, edges of short length in the local optimum are penalized in order to increase the total cost of the solution. Thus, the

local optimum gradually loses its attraction until the algorithm is able to leave the whole basin of attraction of the current minimum, as illustrated in Figure 2.4.

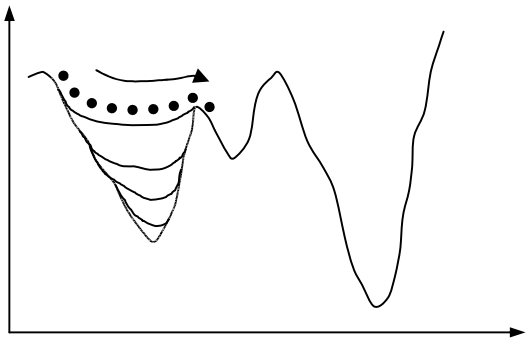


Fig. 2.4: Guided local search. In GLS, local minimum is made less attractive (direction of arrow) so that the algorithm can escape it to the nearest basin of attraction.

2.4.6 Greedy randomized adaptive search procedure

The greedy randomized adaptive search procedure (GRASP) [Feo and Resende 1995] is based on the idea that local search will more likely end up with a high quality solution if it starts from a good initial solution.

GRASP first forms a candidate list of high-quality (according to some greedy function) solution elements. Then, elements are selected randomly from the list to construct a good initial solution. Once an initial solution is constructed, it is used to start a local search until a local optimum is reached. The whole process repeats by constructing a different initial solution, and storing the best found local optima.

In addition to the ease of implementation and the efficiency in the number of parameters to set and tune, GRASP has the advantage of producing quality solutions in low computation time. The fundamental shortcoming in GRASP is the lack of memory, which often leads to final solutions of comparatively low quality.

The discussion on this group of metaheuristics can be summarized as follows: Local Search heuristics stops at a local optimum, which is likely of inferior quality, and in order for the heuristic to produce solutions of acceptable quality without exhaustively searching the solution space, it is necessary to accept non-improving moves, change the starting solution, or manipulate the definition of neighborhood structure or the objective function.

2.5 Population search methods

Unlike the trajectory methods, population search methods manipulate a set of solutions at any instance during the search rather than a single solution, as illustrated in Figure 2.5. These methods include *evolutionary algorithms*, *scatter search* and *ant colony optimization*.

In evolutionary algorithms (EAs) the generation of *offspring* (new) solutions from *parent* (old) solutions resembles the mechanics of natural evolution. The best offspring of the parent solutions are retained for the next *generation* (iteration), thereby proceeding in an evolutionary fashion that encourages the survival of the fittest. Nowadays, EAs include *evolutionary programming* by Fogel et al. [1966], *evolution strategies* by Rechenberg [1973], and *genetic algorithms* by Holland [1975] and Goldberg [1989]. However, their differences have so diminished during the last decade that it is hard to draw clear-cut lines between them.

The main differences between scatter search and evolutionary algorithms are that scatter search specifies stricter rules for recombining solutions, and borrows some memory related ideas from tabu search (see Laguna and Marti [2003] for more discussion on general principles and implementation of scatter search).

Ant colony optimization [Dorigo et al. 1996] has a solution construction phase and a solution improvement phase. The probability of adding an element to the partially constructed solution depends on the level of pheromone, which reflects the goodness of old solutions constructed using that element. A key element in the ant colony algorithm is the

14

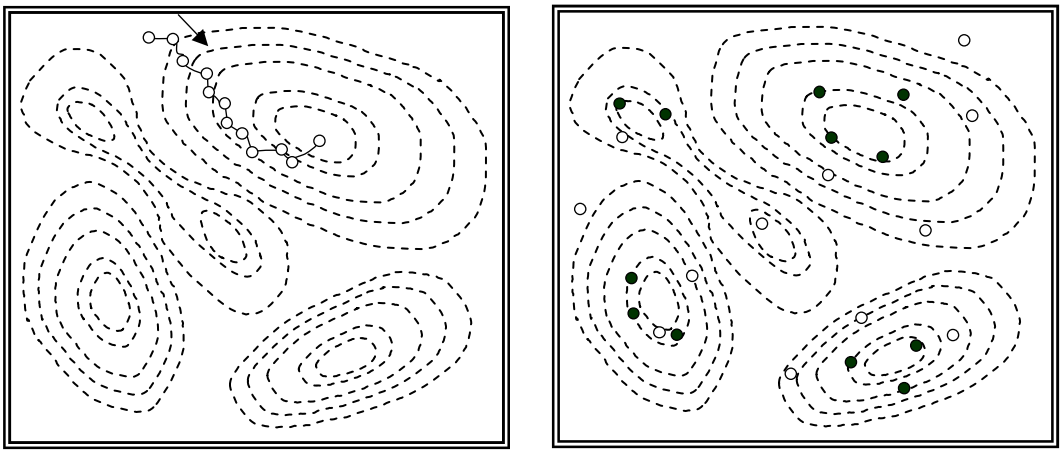


Fig. 2.5: A trajectory metaheuristic versus a population heuristic. (a) trajectory heuristic would likely terminate at a local minimum. (b) GA parent solutions (white) combine to produce offspring solutions (dark).

rules that update pheromone values since these rules change the probability distribution used to sample the search. Reducing pheromone values reduces the attractiveness of visited solutions and leads the search towards new regions and thus helps to avoid early entrapment in local optima that may happen with extremely large pheromone values. More details on the principle of ant colony optimization and how it is engineered to optimization problems can be found in Dorigo and Di Caro [1999].

2.6 Genetic algorithms

Genetic Algorithms (GAs) are stochastic search techniques that seek improved performance by sampling promising regions of the solution space, regions with a high probability for leading to good solutions, Venkatasubramanian and Androulakis [1991]. The algorithms were introduced by Holland in 1975 and were called genetic because in manipulating their solutions they mimic mechanisms of natural selection.

The basic idea of these algorithms is to start from an initial, usually randomly created, set of solutions. In every iteration, new solutions are generated from the existing set using mechanisms modelled after those currently believed to apply in natural evolution of organisms. Some solutions are retained for the subsequent iteration, in which a new cycle of genetic operations are performed; thereby, the algorithm proceeds in an evolutionary manner where the fittest individuals survive. Figure 2.6 shows a traditional genetic algorithm, where the set of solutions under consideration during the i^{th} iteration of the algorithm is called the *population* of the i^{th} *generation* . In fact, this algorithm reflects the characteristics of most evolutionary algorithms. Here it should be noted that, the terms EA and GA are often used interchangeably; however, in referring to the algorithms developed in this study, the term GA is used.

2.6.1 Algorithm components and terminology

In order to implement a genetic algorithm successfully, several issues have to be resolved. These issues are briefly mentioned while discussing the major components of the genetic algorithm.

Encoding

At the outset, there must be a scheme to encode candidate solutions into bit strings. The encoded version of each parameter of a candidate solution is termed a *gene*, and

```
Procedure GeneticAlgorithm
    g = 0;   // initiate generations counter
    Generate(pop);   // generate initial population;
    Evaluate( pop );
    repeat
        g = g+1;
        pop(1) = Select( pop );
        pop(2) = Cross( pop(1) );
        pop = Mutate( pop(2) );
        Evaluate( pop );
    until terminating condition
    Return best individual ;
```

Fig. 2.6: General structure of a genetic algorithm

the encoding of the entire solution is termed *genotype*. Thus, a gene is the basic unit of the genotype. In natural biology, a genotype consists of several *chromosomes*, each chromosome consisting of a number of genes. GAs, however, often use a single chromosome to represent a candidate solution. Hence, the terms *individual*, chromosome, and genotype are used interchangeably. Still, we need to distinguish between the terms *phenotype* and genotype. Whereas genotype refers to the representation of the solution in the chromosome, phenotype refers to the problem version of the genotype, i.e., the actual solution. This is important because genetic operators (selection , crossover, and mutation) work within a genotypic space and manipulate chromosomes rather than the solutions themselves.

Traditionally GAs used binary encoding; nowadays, they also use real values, integers, and permutations. The latter are often used in combinatorial problems.

Fitness evaluation

In applying a genetic algorithm to any optimization problem one has to devise a particular type of objective function called a *fitness function* or an *evaluation function*. This function evaluates and ranks individuals in the population according to their *fitness* (i.e., how good their solutions are) so that individuals producing the best solutions in the population are retained. It is worth noting that nowhere except in the fitness function is there any information (in the typical genetic algorithm) about the problem to be solved.

The term *fitness landscape* is commonly used to visualize the relation between fitness values of all candidate solutions and distances between them as a structure of fitness hills and valleys.

Selection

The term *selection* comes from the well known metaphor, natural selection, introduced by Charles Darwin to refer to the process he believed to induce organisms to adapt to their environments. In GAs selection operators perform the equivalent role to natural selection; that is, they determine which individuals in the current population survive and reproduce offspring. Individuals are selected for mating according to their relative fitness: those with greater fitness are awarded more offspring than those with lesser fitness. There exist many selection schemes in the literature, such as roulette wheel selection, stochastic universal sampling, ranking selection, and tournament selection. Details on these schemes and several others can be found in Michalewicz [1992] and Mitchell [1996].

Crossover

Crossover, also called *recombination* operator, is used to recombine genetic material in parent chromosomes (usually two) to produce one or two child chromosomes sharing characteristics of both parents. A *single point crossover* is a kind of crossover in which the

parent strings are both bisected at same randomly chosen point. Then, one child string is produced from the leftmost portion of one parent and the rightmost portion of the other parent. If a second child is to be produced, it is made from the unused portion of both parents. In a *two point crossover*, each parent chromosomes is bisected at two cut points. Then, the outer portions of one chromosome are recombined with the inner portion of the other chromosome to produce one child chromosome. The second child chromosome can be constructed from the unused portion of the parents. Figure 2.7 illustrates one point crossover and two point crossover.

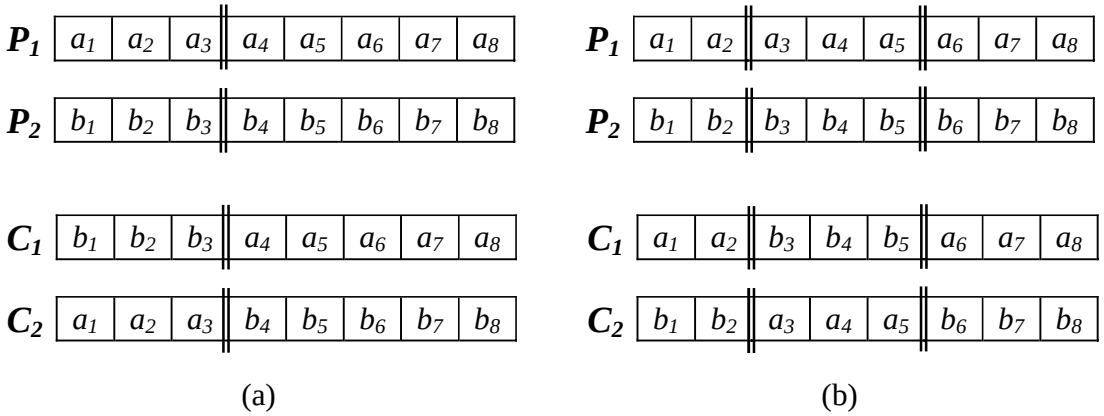


Fig. 2.7: Single point and two point crossover. (a) In single point crossover, Each of the parents P_1 and P_2 is split at one point into two portions. Children C_1 and C_2 are produced by recombining portions from different parents, and combined to produce children. (b) In two point crossover, P_1 and P_2 are split at two points, the resulting portions recombined to produce C_1 and C_2.

Besides these two means of crossover there exist several other techniques. If permutation representation is used, then conventional crossover operators will produce infeasible solutions. *Order crossover* [Davis 1985], *partially mapped crossover* [Goldberg and Lingle, Jr. 1985], and *cycle crossover* [Oliver et al. 1987] are examples of operators that are specifically developed to recombine solutions into feasible children. A detailed description of

these operators with many more can be found in Michalewicz [1992].

Mutation

The main function of *mutation* operators is to ensure that the population is supplied with new genetic material throughout the search process. Mutation enables the GA to maintain diversity in the population and introduces some random search behavior necessary to explore new unvisited regions of the search space that may not be reached by crossover only. Thus, mutation plays a complementary role to that of crossover, which works on material already present in the population and thus cannot introduce new genetic material. In other words, without a mutation operator, genetic material nonexistent in the initial population or lost with discarded individuals can never be developed again.

Mutation works as a unary operator that transforms one parent chromosome into a different child chromosome. In binary representations, the mutation operator replaces each bit in the string by a randomly selected bit, if a probability test is passed. In real valued representations, an individual can be mutated by slightly increasing or decreasing the value of some of its genes. In permutation representations, mutation can be accomplished by swapping two genes in the chromosome to produce a different permutation.

In the following chapters, the term *Genetic Operators* will be used to refer to the main operators of the GA (selection, crossover, and mutation) collectively, and the term *genetic parameters* to refer to their rates (or probabilities) of application.

Termination criteria

Unlike local search, which stops when a local optimum is encountered, GAs can in principle evolve without stopping, thus it is often the case that a total number of generations is pre-specified by the user at which the run terminates. Other less frequently used criteria terminate the GA run when a *best-so-far* solution cannot be improved, pre-determined solution quality is obtained, or a lower limit of population diversity is reached.

2.6.2 Theoretical aspects of genetic algorithms

The schema theorem of Holland [1975] was the first rigorous attempt to explain how GAs work [Beasley et al. 1993a]. Holland introduced the term *schema* to explain his theorem in binary representation. A schema is a template made from the alphabet {0 1 #}, with the "#" symbol matching 0 or 1. For example, the schema {0 1 1 # 0 0} matches the two strings {0 1 1 0 0 0} and {0 1 1 1 0 0}, and the schema {# 1 1 0 # 0} matches the strings {0 1 1 0 0 0}, {0 1 1 0 1 0}, {1 1 1 0 0 0}, and {1 1 1 0 1 0}. He defined the *order* of the schema as the number of *0*'s and *1*'s it contains, and the *defining length* of the schema as the distance between the two outer most non-# symbols in the schema. With these terms, the schema theorem, can be stated as follows.

Theorem 2.1 (Schema Theorem) *At a certain generation of the genetic algorithm, existing schemata that are of short defining length, low order, and above the average fitness receive exponentially increasing trials in subsequent generations.*

Goldberg gives a similar explanation as to how GAs work in his *Building Block Hypothesis* [Goldberg 1989]: "Just as a child creates magnificent fortresses through the arrangement of simple blocks of wood, so does a genetic algorithm seek near optimal performance through the juxtaposition of short, low order, high-performance schemata, or building blocks".

These two widely known, but not so widely accepted, explanations are criticized by many researchers. Both explanations presume that high quality solutions consist of good "building blocks" where a good building block is a few genes in the chromosome that contribute to the high fitness value of the individual. With these explanations, the success of a GA in finding good solutions is primarily determined by its ability to discover, emphasize, and recombine good building blocks from different chromosomes to construct final solutions. At one time, these explanations were seen to be satisfactory, and many algorithms were designed to exploit the building blocks and increase the good schemata.

However, a contemporary view, that is shared by many researchers, is that building blocks should be viewed as a dynamic entity that changes over the search process. Under this view, the quest to detect and combine good building blocks makes little sense, since what can be a good block at one generation might become less good in the next generation.

More refreshing analysis of the schema theorem and the building block hypothesis can be found in Reeves and Rowe [2002]. At the present time there is no generally accepted theorem that explains the GA behavior (nor the behavior of other metaheuristic methods).

2.6.3 Why genetic algorithms?

Over the last two decades, evolutionary algorithms have been successfully applied in many fields, such as engineering, robotics, economics, chemistry, genetics, operations research, art, and social sciences.

There are some characteristics of GAs to which their success as search procedures is attributed. First, GAs manipulate a population of solutions at each generation; thus, the probability of getting trapped in local optima is reduced compared with methods that proceed from point to point in the solution space. This characteristic also makes them suitable for parallelization. Second, genetic operators make use of a coding of the parameter space rather than the parameters themselves (only objective function information is used), which simplifies implementation. Third, although they employ some stochastic components, they make use of all the information that has been obtained during the search [Goldberg 1989]. However, to be effective on COPs, GAs are often hybridized with other local search based techniques [Maniezzo V. 2002].

When considering dynamic problems, multiple objective problems or both , GAs have additional advantages over other search methods. Their manipulation of a population of solutions proves to be advantageous. Even when a change in the environment renders the current best solution unacceptable, it is likely that the quality of some of the remaining solutions in the population does not deteriorate. Furthermore, the population will be

helpful for multiple objective problems where the goal is to provide several non-dominated solutions to the decision maker for a final decision.

However, running a GA entails setting a number of parameter values, such as the population size, mutation rate, and crossover rate. Finding the best setting for the problem at hand is not a trivial task. A poor setting can lead to final solutions of unacceptable quality. The main problem here is that these parameters affect each other in a non-linear way that is also not clear; hence, they are often hard to optimize one at a time. This thesis addresses this difficulty to some degree in a later chapter as it faces dynamic problems as well.

2.7 Summary

An overview of the basic metaheuristics and exact search methods has been given in this chapter. Metaheuristics in general involve sampling the search space, which helps them produce solutions quickly; however, their main disadvantage is their inability to guarantee solution quality.

Population-based metaheuristics tend to be more effective than local search based ones; however, in terms of fine-tuning specific search areas, the latter group is known to be more effective. In general, the degree of success of these methods on a given problem depends largely on their ability to strike a balance between exploration and exploitation.

Theoretical foundations of evolutionary algorithms, and those of most metaheuristics, are debateable. So far there is no robust theory that conclusively explains the behavior of these algorithms. Practitioners depend mainly on empirical experiments to demonstrate the potential of their implementations.

The ability of evolutionary algorithms to sample the search space and the fact that they simultaneously manipulate a group of solutions increase their potential for dynamic problems. Techniques which exploit theses qualities are reviewed in the he next chapter.

Chapter 3

Literature Review: Evolutionary Algorithms in Dynamic Environments

It is difficult to determine a useful amount of diversity: Too much will resemble restart, while too little doesnt solve the problem of convergence.

Jin and Branke [2005b]

3.1 Introduction

In this chapter, the use of evolutionary approaches to track shifting optima in dynamic problems is surveyed. The literature is classified according to the underlying elementary approaches in order to show the trends in recent research and more importantly to form a basis for combining these approaches into more effective ones.

The chapter concludes with a highlight of the gaps in the literature that can be the subject of future research, with emphasis on those issues that are addressed in this thesis.

3.2 Dynamic problems, an overview

Dynamism in real-world problems can be attributed to several factors: Some are natural like wear and weather conditions; some can be related to human behavior like variation in aptitude of different individuals, inefficiency, absence and sickness; and others are business-related like the addition of new orders and the cancellation of old ones.

However, the mere existence of a time dimension in a problem does not mean that the problem is dynamic. Problems that can be solved ahead of time are not dynamic and not considered in this thesis even though they might be time dependent. Psaraftis [1995] discusses three examples of such problems: In the "time-dependent TSP", travel times between cities change during the day (early morning, rush hour, late at night) in a well-known manner. This problem can be solved in advance although it is changing with time. In the "probabilistic TSP", travel times are constant but demand at each city occurs with a known probability. Again this problem is not dynamic because it calls for a solution that is computed ahead of time. Although the actual city status might be given in real time (i.e whether there is some demand at that city or not) the optimization process has already been concluded. The pre-computed solution will be implemented regardless of the actual demand; i.e., the vehicle will be dispatched on the computed route and will

simply skip any cities that give zero demand while the vehicle is on the move. The third example considers a stochastic VRP, where a set of solutions is pre-determined so that some overall objective cost is minimized. Again, there is no need for re-optimization in the future and thus the problem remains virtually static. Similar examples include the scheduling of nurses and aircraft crews and the timetabling of faculty. In these problems if the variations are limited to the seasonal considerations or days of the week, then these problems, though time-dependent, are not considered dynamic. A different example is the time-phased allocation and budgeting problem. If future demands are either known in advance or predictable with sufficient accuracy, then the whole problem can be solved ahead of time.

According to Psaraftis [1995],Bianchi [1990], and Branke [2001], the following features can be found in most real-world dynamic problems:

- Time dependency: the problem can change with time in such a way that future instances are not completely known, yet the problem is completely known up to the current moment without any ambiguity about past information.
- A solution that is optimal or near optimal at a certain instance may lose its quality in the next instance, or may even become infeasible.
- The goal of the optimization algorithm is to track the shifting optima through time as closely as possible.
- Solutions cannot be determined ahead of time but should be found in response to the incoming information.
- Solving the problem entails setting up a strategy that specifies how the algorithm should react to environmental changes, e.g. to resolve the problem from scratch at every change or to adapt some parameters of the algorithm to the changes.

- The problem is often associated with advances in information systems and communication technologies which enable the processing of information as soon as received. In fact, many dynamic problems have come to exist as a direct result of advances in communication and real-time systems.

Techniques that work for static problems may therefore not be effective for dynamic problems which require algorithms that make use of old information to find new optima quickly. The next section is a survey of adaptive algorithms specifically developed to track optima in changing environments.

3.3 Tracking optima

The main disadvantage of applying a conventional "static" algorithm in a dynamic environment is that once the algorithm starts to converge around some optimal or near-optimal solution, it will very likely lose its ability to continue the search for new optima if the environment shifts. Hence, the key point in optima-tracking approaches is to increase the diversity of the search so that the algorithm retains its ability to explore the new search space when the problem changes, even after it has converged or nearly converged to some optimum.

The following sections review elementary tracking approaches reported in the literature. The general idea is to study the approaches in their elementary form in order to develop and analyze complex approaches for real-world problems.

3.3.1 Restart

The most straightforward approach to increase diversity of a GA search is to restart the algorithm completely by reinitializing the population after each environmental change. If the entire population is reinitialized, we have a *complete restart*, which indicates that the

dynamic problem is treated as a series of static problems to be solved independently of each other. As a consequence, any information gained in the past search will be discarded with the old population after every environmental change. Thus, if changes in the problem are frequent, this time consuming method will likely produce results of low quality. Furthermore, successive instances in the typical dynamic problem do not differ completely from each other. Hence, some researchers use restart partially. With *partial restart*, rather than reinitializing the entire population randomly, a fraction of the new population is seeded with solutions that proved to be of good quality in the past. Louis and Xu [1996] reported good results with partial restart (10% of the new population is replaced with old solutions), nevertheless, their method failed to produce good results when more than 50% of the individuals were seeded from the old run.

An interesting observation was made by Louis and Johnson [1997]. They noted that when the change severity is large, it is better to select individuals with low fitness—rather than individuals with high fitness—from the old run to seed the current population. This observation could be attributed to the increase in diversity resulting from low fitness individuals, which are expected to be located far from the optimum (while high quality solutions tend to be located in the same basin).

Ramsey and Grefenstette [1993] proposed expressing the environment in terms of a few variables so that a knowledge can be built by matching the values of these variables with the optimal solutions found. This knowledge can be used later to determine the best seed solutions for a new environment by comparing the environment variables with those stored in the knowledge base. Although the idea seems interesting, a real world environment is so complex that it is hard to represent even with all its variables. Nevertheless, for this method to be effective it should use only a few expressive variables. The difficulty is then to determine which variables to choose to represent the environment, and to determine whether they will really be able to represent it fairly. Furthermore, the extra task of comparing knowledge and environments is likely to be time consuming.

It should be noted that in seeding new populations with old solutions it is assumed that solutions remain feasible after the environment changes. However, environmental changes can involve the problem constraints. In such cases, old solutions may no longer be feasible and cannot be directly used as seeds. Bierwirth and Kopfer [1994a] and Bierwirth et al. [1995a] consider the case of new job arrivals in a job shop scheduling problem. They modify all solutions when the environment changes in a way that makes the solutions reusable as seeds for the new population.

In summary, a complete restart provides the diversity necessary to continue the search for new solutions, but fails to exploit historical information on the problem. Its computational cost makes it inappropriate for dynamic problems in which the changes are frequent and with small severity. In such cases, alternative approaches that are capable of injecting diversity and at the same time able to exploit old information are preferred.

3.3.2 Adapting genetic parameters

It is well known that changing the parameters of a GA can change the behavior of the algorithm and consequently can affect the quality of the final solution. The general view now is that there is no fixed set of parameters that remain optimal throughout the search process—even in a static problem. Many researchers, such as Davis [1989] and Bäck [1992], have explored the use of adaptive genetic operators in stationary environments. Other researchers even investigated self-adapting parameters [Bäck 1997]. Self-adapting mutation rates are also widely used in evolution strategies [Bäck and Schwefel 1993]. An extensive survey and discussion of the method of parameter control in evolutionary algorithms in static problems can be found in Eiben et al. [1999].

With variable parameters (self adapting or otherwise) finding some success on static problems, it would be natural to investigate their use for applications where the problem environment is shifting. Now, many researchers do not use fixed parameter settings; instead, they employ techniques to adapt the algorithm to changes, such as adapting se-

lection pressure, mutation rate, and population size. This kind of parameter adaptation basically aims to avoid the pitfalls of the restart methods while retaining a level of diversity sufficient to maintain an effective search all the time.

Mutation, as the main diversity operator in GAs, is the most widely investigated parameter for adaptation. It works by introducing some change to the genes of an individual to create another individual. The number of genes altered by mutation is directly proportional to the rate of the mutation (probability that a given gene is altered). Thus, by controlling the rate of mutation, it is possible to change distances between individuals in the population and consequently manipulate the population diversity.

Cobb [1990] proposed a method called *hypermutation* to track optima in continuously changing environments. Cobb's approach measures the quality of the best performers over time. When this measure worsens (indicating a shift in the environment), the mutation rate is increased drastically. Morrison and Jong [2000] used what they call "Triggered Hypermutation". In effect, this approach re-initializes the population, but in a less drastic manner than that of the restart approach.

Grefenstette [1992] proposed a method called *random immigrants* (randomly generated individuals) to maintain diversity in the population without disturbing the current search. Random immigrants work by replacing only a fixed percentage of the population at every generation. Later, Cobb and Grefenstette [1993] investigated both methods. They reported that hypermutation produced better results than those of random immigrants when the severity of change is small, but random immigrants did better when environmental changes are severe.

Grefenstette [1999] investigated an interesting mutation model in which the rate of mutation is controlled genetically. In this approach, the chromosome contains a "mutation control gene". The additional gene undergoes all normal GA operations but does not contribute to fitness computation. Instead, the value decoded from the mutation control gene is used to calculate the rate of mutation. He compared it with two other models

(fixed mutation-rate and hypermutation) on abrupt and gradual changing environments. He reported that results of traditional fixed mutation were the worst in all runs. The genetically controlled mutation was slightly worse than the hypermutation model which gave the best overall results.

Nevertheless, the manner in which the rate of mutation self changes in response to environmental changes is of great interest as it can be used as a basis for designing dynamic solvers with an adaptive mutation rate as we will see in the next chapter.

3.4 Enhancing algorithm performance

A crucial requirement of an adaptive algorithm working in real-time is its ability to respond rapidly to environmental changes. In other words, the time spent to adapt a solution to changes should be small. Hence, an increasing amount of research is aimed at improving GA performance: reducing computation time as well as improving solution quality. Common approaches use memory, parallel GA, dedicated hardware, problem preparation, and solution robustness.

3.4.1 Memory

In static applications, a typical role of memory is to maintain diversity by keeping track of visited areas in the search space; consequently it guides the algorithm to unvisited areas of the search space.

In dynamic problems, additional memory can be used to store old good solutions with the assumption that the optimum may return to its former value. When certain aspects of the problem exhibit some kind of periodic behavior, old solutions might be used to bias the search in their vicinity and reduce computational time. Ng and Wong [1995] and Lewis et al. [1998] are among the first who used memory-based approaches in dynamic problems.

It may seem that these uses of memories have conflicting roles, but actually they do not, since information is not retrieved from these two kinds of memory at the same time. The memory used to maintain diversity prevents the algorithm from unnecessary wandering as long as the environment is stationary. As soon as an environmental change is detected, this memory is reset (enabling the algorithm to search the entire search space including visited areas); information is then retrieved from the memory used to store old solutions of high quality (to bias the new search towards regions that proved to be good in the past). Then, the memory used to maintain diversity regains control again.

The memory to store old solutions of high quality should be used with care as it may have the negative effect of misguiding the GA and preventing it from searching new regions. This is confirmed by Branke [1999b] who concludes that memory alone is not enough and it should be combined with a diversification mechanism to keep exploring the search space for new promising areas. This should be expected in dynamic environments where information stored in memory becomes more and more obsolete as time proceeds.

3.4.2 Multiple population genetic algorithms

The inherent parallel structure of GAs makes them ideal candidates for parallelization. Since the GA modules work on the individuals of the population independently, it is straightforward to parallelize several aspects of a GA including the creation of initial populations, individual evaluation, crossover, and mutation. Communication between the processors will be needed only in the selection module since individuals are selected according to global information distributed among all the processors.

The most common parallel model of GA is the Island Genetic Algorithm (IGA) by Tanese [1989], and Whitley and Starkweather [1990]. IGA divides the population into several subpopulations or *islands* allocated to one or more processors. These islands evolve independently from each other for a period of time, called the *isolation time*, during which no communication between processors is needed. At the end of each isolation time inter-

val, the islands exchange individuals in a process which does not require a high level of communication.

The IGA model not only alleviates the communication load but also has the advantage of developing niches which lead to better solution quality at the expense of slight slower convergence. In computational tests this model showed a speedup in computation time, and in addition, it required fewer objective function evaluations, when compared to a single population algorithm. Even if the IGA model is implemented in a serial manner (i.e. with single processor), it was quicker than the standard GA in reaching the global optimum.

Recently, three multi-population implementations were specifically developed for dynamic environments: the *shifting balance genetic algorithm* (SBGA) by Oppacher and Wineberg [1999], and Wineberg and Oppacher [2000]; the *multinational genetic algorithm* (MGA) by Ursem [1999, 2000]; and the *self-organizing scouts*(SOS) by Branke et al. [2000]. These are briefly described now.

Shifting balance genetic algorithm

In SBGA there is one large *core* population that contains the best found individual, and several small colony populations that keep searching for new optima. The main function of the core population is to track the shifting optimal solution. The colonies update the core population by sending immigrants from time to time.

Self-organizing scouts

The SOS approach adopts an opposite approach to SBGA by allocating the task of searching for new optima to the *base* (main) population and the tracking to the *scout* (satellite) populations. The idea in SOS is that once a peak is discovered there is no need to have many individuals around it; a fraction of the base population is sufficient to do the task of tracking that particular peak over time. By keeping one large base population, SOS behaves more like a standard GA—rather than an IGA—since the main search is allocated

to one population. This suggests that the method will be more effective when the environment is dynamic (many different optima arise through time) and hence the use of scouts will be warranted. SOS is more adaptive than SBGA which basically maintains only one good solution in its base.

Multinational genetic algorithm

MGA uses several populations of comparable sizes, each containing one good individual (the peak of the neighborhood). MGA is also self-organizing since it structures the population into subpopulations using an interesting procedure called *hill-valley detection*. Here, several random points on the line between the two given end points are evaluated, and a valley is detected if a sample point has lower fitness than that of both end points. The procedure is used to determine if an individual is not located on the same peak with the rest of its population and hence it should immigrate to a different population. The procedure can also lead to the merging of two populations if it finds that they represent the same peak. This method combines the advantage of SBGA (maintaining several search populations that can be very effective for multimodal functions) with that of SOS (maintaining several optima found through time). The main disadvantage of MGA is the frequent evaluations done for valley detection.

Regardless of the specific application in which these multi-population approaches were used, their use and testing was limited to continuous optimization problems. Hence, they require further modification in order to make them useful to COPs.

3.4.3 Adapting search to population diversity

This section reviews some of the schemes used to enhance GA performance in dynamic environments by using population diversity as a guide to control genetic parameters.

The idea of using diversity to guide evolutionary algorithms is adopted by several

researchers in many recent publications. Zhu [2003] presents an adaptive GA for vehicle routing problems. The population diversity is maintained at pre-defined levels by adapting rates of GA operators to the problem dynamics. Zhu and Liu [2004] present an empirical study of population diversity measures and adaptive control of population diversity for a permutation-based genetic algorithm. Burke et al. [2004] examine several measures of diversity in genetic programming. Ursem [2002] measures population diversity as the sum of distances to an average point and uses it to guide the search process. Riget and Vesterstroem [2002] use an approach similar to that of Ursem [2002] but on particle swarm optimization. Sörensen and Sevaux [2004] present a memetic algorithm with population management to control population diversity. They use *edit distance* between individuals in the population to measure its diversity. The edit distance, known also as Levenshtein distance, between two strings is the smallest number of operations required to transform one string into another, where an operation can be an insertion, a deletion, or a substitution [Algorithms and Theory of Computation Handbook 1999].

These publications have targeted either static problems or dynamic continuous optimization problems, thus none can be used without modification for dynamic COPs. Moreover, measuring diversity as the sum of distances between each individual and the rest of the individuals in the population is computationally expensive. Computational cost can be reduced by limiting the population size to a few individuals as proposed by Sörensen and Sevaux [2004], but computational expense is not a sufficient reason for limiting population size.

Another way of measuring diversity is to use a single aggregation point as a representative of the whole population, and thus reduce computation requirements of the diversity by a factor of n, where n is the population size [Riget and Vesterstroem 2002; Ursem 2002]. However, this approach will not work for COPs, where there is no obvious choice for an "average" point for the population.

3.4.4 Dedicated hardware

Hardware implementations like field-programmable gate array chips of GAs are extremely fast [Graham and Nelson 1996; Koza et al. 1998; Aporntewan and Chongstitvatana 2001]. Recently, Koonar et al. [2002] designed a new architecture for implementing GAa in hardware. They reported more than 100 times improvement in processing speed over the software implementations. Although their architecture was designed specifically to solve the circuit partitioning problem for VLSI CAD design, some modules of their design can be used for other problems. This is another advantage of using GAs where information specific to the problem at hand is processed in the evaluation function module only, allowing other modules to be used with different problems.

The knowledge that specific hardware is beneficial to GAs encourages the investigation of adaptive GAs on dynamic problems, with the assumption that the behavior of the final algorithm can be further enhanced on such hardware.

3.5 Robust solutions

Practical considerations may make perturbing previous schedules undesirable, because of commitments made to old solutions or due to the high costs of switching between different solutions. As a result, the decision maker would have to adhere to the same solution as long as possible even if the environment changes. The situation is more pronounced in those applications where human beings are directly affected by the solution at hand. For example, workers in a plant will be reluctant to move from one machine to another, similarly aircraft crews would not be keen on having new schedules once they have arranged their private lives to a certain schedule.

In these cases, the goal is to look for robust solutions, that is solutions that are expected to retain some, if not all, of their good quality over a wide range of time and environmental change. In addition, robust solutions might also be desired in those cases when the

environmental changes are too fast or when they cannot be detected quickly.

The objective function of the given problem will be modified to include minimization of sensitivity to changes as an additional objective. One common approach measures robustness of an individual by evaluating several random solutions in its neighborhood and taking their mean fitness as the fitness value for the individual. This approach is similar to optimizing noisy functions; the main difference is that random disturbances are added to the fitness value of a noisy function, whereas in measuring robustness, the individual rather than its fitness is disturbed. Many researchers report successful results in various applications, such as flight control subject to changing weather conditions [Blythe 1998], wingbox design optimization with manufacturing tolerances [McIlhagga et al. 1996], job shop scheduling [Reeves 1992; Tjørnfelt-Jensen and Hansen 1999], and robot control [Jakobi 1997; Nordin and Banzhaf 1996]. However, this method suffers from the obvious increased computation time due to the large number of evaluations.

Parmee [1996a,b] describes a method for measuring robustness in terms of sensitivity to parameter changes. He suggests restricting the search for robust solutions to regions of high performance only. This method aims to reduce computation time. However, it does not deal with the objective function directly and hence may produce inferior results. Furthermore, as it restricts the search to regions of high performance , it may miss some robust solutions that are located in suboptimal regions. The latter observation is confirmed by Wiesmann et al. [1998] who points out that robust solutions can be found even in regions of low performance.

The work by Hart and Ross [1999a,b] and Hart et al. [1998] aims to accelerate the construction of a robust schedule. They propose an artificial immune system that introduces disturbances by combining building blocks of a solution in different ways.

The interested reader is referred to Jin and Branke [2005a] for a comprehensive survey of the use of EAs in uncertain environments.

3.6 Benchmark generation

One of the most crucial outstanding issues in dynamic optimization is the generation of benchmark problems necessary to compare the ability of different algorithms to adapt to a wide variety of environmental changes.

Definition 3.1 (Benchmark problem) *By benchmark problems or simply benchmarks, we mean standardized test problems designed to serve as a basis for algorithm evaluation and comparison.*

The usual reason for running an algorithm on such problems is to obtain results that are comparable to studies on other algorithms and can, hence, attest to the superiority (or inferiority) of the tested algorithm. Benchmarks can also help highlight strengths and expose weaknesses of the optimizing algorithm and help in understanding the operation of the algorithmic ideas. These different usages give rise to distinct benchmarks. However, the focus in this thesis is on benchmarks intended for comparing the performance of competing strategies or algorithms.

There are two basic types of data for benchmark construction: real life data and synthetic data. Real life data produces cases that closely reflect real world problems, and can thus attest to the algorithm ability to fulfill its ultimate goal (solving a particular real world problem). They tend, however, to be very problem specific. The second type of data is often randomly generated and produces artificial landscapes that are less problem specific and typically unrelated to real world problems. This type is easy to obtain and analyze, and their use enables the drawing of general conclusions about the algorithm performance.

Benchmark problems, though often constructed from synthetic data, should show some justification to the real world in a way that enables meaningful conjecture to real problems. In dynamic optimization, however, a real world problem is also characterized by scenarios postulating the sequence of events (or environmental changes) in the problem. Thus, one

would be faced by the additional difficulty of identifying the typical scenario(s), upon which the benchmark should be based.

There exist a number of dynamic benchmark problems in the literature. Some are intended for general use and are usually referred to as general-purpose test cases or benchmarks. Others are specifically designed to test a particular algorithm. In most of the surveyed literature, the authors use only one kind of benchmark. Users of benchmarks of the first category focus on changing problem difficulty (to the solving algorithm) in order to expose any weaknesses in the algorithm. The results from using this approach however are generally too theoretical and have little or no bearing to the real world. On the other hand, users of benchmarks from the second category typically target some specific problem from the real world—attributing little significance to the first category.

3.6.1 General-purpose benchmark generators: Are they really needed?

Having diverse test problems to evaluate and demonstrate the effectiveness of non-exact algorithms is widely appreciated in static optimization. In dynamic optimization, test problems have the additional requirement to cover wide ranges of environmental changes so that they can pose as credible testers for *dynamic solvers* (DSs).

Researchers who introduced dynamic benchmarks have identified some fundamental features which characterize an effective set of benchmarks. Combining the remarks of Psaraftis [1995], Branke [2001], and Yang [2004], one can identify the following features:

- The benchmarks should not be too problem specific.
- They should be able to introduce dynamism to all elements of the optimization problem, with controllable degrees of frequency of change, severity of change, and other dynamics.

- Optimal solutions to the test problems should be known.

These requirements motivated several researchers [Branke 1999b; Grefenstette 1999; Morrison and De Jong 1999; Trojanowski and Michalewicz 1999] to introduce general purpose benchmark generators (BGs) that can generate artificial dynamic landscapes with controllable features.

The dynamic landscape presented by Grefenstette [1999] is specified as a set of components where each component consists of a single n-dimensional Gaussian peak characterized by three features: center, amplitude, and width. Of all peaks in the landscape, only one is optimal with an amplitude of 100, while the remaining peaks have amplitudes between 10 and 50. All peaks move over time in randomly selected directions, while their amplitudes are kept constant. Morrison and De Jong [1999] describe their problem generator as a two step process. In the first step, a baseline static landscape with the desired complexity is specified; In the second step, the desired dynamics are added. The function they use for the static landscape is similar to the Gaussian used in Grefenstette [1999], but in their implementation changes can be applied to any of the height, slope and position of selected peaks. In a similar work, Branke [1999b] suggests a *moving peaks function*, which is basically a multimodal function with controllable height, width and position for each peak. The moving peaks function, however, offers an additional parameter λ, ranging between 0.0 and 1.0 to quantify "how much a peak's change in location depends on its previous move". Setting $\lambda = 0.0$ makes the peak's change completely random, while the other extreme, $\lambda = 1.0$, means the peak continues to shift in the same direction. The search space of the test case generator presented by Trojanowski and Michalewicz [1999] is divided into a number of disjoint subspaces, where for each subspace there is one unimodal function defined within that subspace only. Thus, the number of these functions corresponds to the total number of local optima in the overall function, and dynamism can be introduced by changing the heights of some of the local optima.

Weicker and Weicker [1999] present a different BG in which the change in the dynamic

problem takes place in the form of rotation of the coordinates of the underlying function, rather than the usual translation of coordinates used by most other BGs. Unfortunately, such changes reflect only a small portion of the environmental changes that are encountered in reality.

Recently, Yang [2003] used a different approach to construct dynamic environments. Instead of explicitly defining time varying functions, he constructs the dynamic problem by continually introducing changes to a base stationary problem. He proposes using an exclusive-or (XOR) operator to introduce changes to a binary-encoded stationary problem. Yang and Yao [2003] use the XOR operator to generate a series of dynamic problems from a randomly generated stationary knapsack problem.

Yang [2004] embarks on a different aspect of benchmark generation that aims to enable the testing of genetic algorithms in dynamic environments with changing but controllable difficulty levels. He builds on the work of Goldberg [2002], where problem difficulty for GAs is attributed to three core elements: deception, scaling, and noise. By controlling the three elements, it is possible to generate more complex dynamic environments with various levels of problem difficulty.

Jin and Sendhoff [2004] have introduced a computationally efficient method based on concepts from multi-objective optimization. They construct dynamic single objective and multi-objective test problems by aggregating different objectives of a multiple objective optimization problem and changing the weights dynamically.

Unlike the preceding examples, many researchers depend solely on problem specific benchmarks [Branke 2001]. Others even downplay the usefulness of general-purpose BGs. For example Ursem et al. [2002] note that the previous BGs create artificial dynamic problems where the shape and dynamics of the fitness landscape are introduced without any justifying relationship to real problems. Perhaps this explains why many researchers use problem specific generators only. There are clearly many applications where a dynamic solver cannot be tested on the previous BGs unless it is modified so drastically that the

new dynamic solver bears little resemblance to the original. Nevertheless, where applicable these generators are handy tools for comparing algorithms, especially when environmental changes of interest are not predictable.

The importance of general benchmark problems, which depend on synthetic and randomly generated data, is evident for many reasons. First, synthetic data can be more effective than real life data in comparing algorithms. With synthetic data, it is possible to introduce variations of different degrees to the elements of the optimization problem individually and in combination, whereas real life data is often too complex to evaluate easily. Second, synthetic data might be the only way to detect deficiencies (in an algorithm) that are not visible through the real world data available at the time of evaluation. Third, the "No Free Lunch Theorems for Optimization" of Wolpert and Macready [1997] imply that results of comparisons of competing algorithms are valid only within the class of problems under consideration. Thus, the use of general benchmarks permits conjecture to a wider variety of problems, and hence promotes portability of ideas to other problems. Furthermore, it is well known that the use of GAs is often justified by their robustness, and it is unreasonable to confine test cases of a general algorithm to problem-specific data.

In summary, the use of problem-specific test cases alone, at best confines the results of testing to the particular optimization problem under consideration, and at worst hinders the generalization of the results to problem instances other than those specifically used. In any case, problem specific tests do not encourage using algorithmic ideas with other problems. Yet, most general benchmark generators available in the literature target continuous optimization. As they stand, the available general generators have little use in discrete optimization, except maybe for the limited cases discussed in the next section.

3.6.2 Representative dynamic combinatorial problems

Combinatorial problems typically assume distinct structures (for example vehicle routing versus job shop scheduling). Consequently, benchmark problems for COPs tend to be very

specific to the application at hand. The test problems used for dynamic scheduling and sequencing with evolutionary algorithms are typical examples [Bierwirth and Kopfer 1994b; Bierwirth et al. 1995b; Bierwirth and Mattfeld 1999; Lin et al. 1997; Reeves and Karatza 1993]. However, the *time-varying knapsack problem* and the *dynamic travelling salesman problem* have static counterparts that are often considered representative of various combinatorial problems. This reason partially explains the popularity of these benchmarks [Alba and Saucedo 2005; Eyckelhof and Snoek 2002; Goldberg and Smith 1987; Guntsch et al. 2001; Lewis et al. 1998; Mori et al. 1996, 1998; Younes and Basir 2002]. In what follows both problems are used as examples to illustrate some issues relevant to all COPs.

Here, it should be mentioned that there exist other combinatorial test problems primarily designed to help in understanding the operation of the optimizing algorithm. Examples of such test problems are the dynamic bit matching function [Stanhope and Daida 1999] used to predict GA performance, and the shaky ladder hyperplane-defined functions [Rand and Riolo 2005] used to investigate the algorithm behavior. However, these benchmarks are not covered in this thesis.

Knapsack problem

The *knapsack problem* (KP) models the problem of placing objects in a knapsack with a limited capacity (weight or volume). As each object has its own value and weight, the objective is to load the knapsack with the most valuable objects without violating the maximum capacity constraint.

There exist several time-dependent variants of this problem used as benchmarks for dynamic optimization. For instance, Goldberg and Smith [1987] present a dynamic benchmark using a seventeen-object knapsack with a weight capacity oscillating between two values. Other researchers [Lewis et al. 1998; Mori et al. 1996, 1998] increase the number of objects and make the weight vary over several values. The main idea of dynamism in

these benchmarks is to vary the allowable weight limit with time. In this way, an optimal solution becomes infeasible if the knapsack capacity is sufficiently reduced. These benchmarks offer an additional feature over the conventional continuous benchmark generators of Section 3.6.1 by including changes in the problem constraints. However, dynamism in a KP should not be limited to changing the knapsack capacity since a comprehensive dynamic KP should take into account the possibility of changing the number, value, and weight of the objects.

Travelling salesman problem

Although the *Travelling Salesman Problem* (TSP) finds applications in science and engineering, its real importance stems from the fact that it is typical of many COPs. Furthermore, it has often been the case that progress on the TSP has led to progress on other COPs. The TSP is modelled to answer the following question: if a travelling salesman wishes to visit exactly once each of a list of cities and then return to the home city, what is the shortest route the travelling salesman can take?

As an easy to describe but a hard to solve problem, the TSP has fascinated many researchers, and some have developed time-varying variants as dynamic benchmarks. Guntsch et al. [2001] introduce a dynamic TSP where environmental change takes place by exchanging a number of cities from the actual problem with the same number from a spare pool of cities. They use this problem to test an adaptive ant colony algorithm. Eyckelhof and Snoek [2002] test a new ants system approach on another version of the dynamic problem. In their benchmark, they vary edge length by a constant increment or decrement to imitate the appearance and the removal of traffic jams on roads. Younes and Basir [2002] introduce a scheme to generate a dynamic TSP in a more comprehensive way. In their benchmarks, environmental changes take place in the form of variations in the edge length, number of cities, and what they term city-swap changes.

3.6.3 Shortcomings of discrete benchmarks

Existing dynamic discrete benchmarks are limited both in the size of the basic static problem and in the applied dynamics. The cause of these limitations can be better addressed by drawing a distinction between the generation of benchmarks for continuous optimization and that for discrete optimization.

Benchmark generators for continuous optimization use functions with adjustable parameters to generate dynamic test problems with known optimal solutions. The generator typically starts with a clearly defined static landscape then adds changes to some of the peaks (location, height, and width) to create various landscapes in which optima shift through time. The generator might even change the difficulty of the underlying static function, as in Yang [2004], but the landscape at any instance of the dynamic problem is still well defined in that values and locations of the local and global optima can be (pre-)determined.

In discrete optimization, however, the metaphor of landscape (together with its related terms such as hills, valleys, basins, etc.) remains indistinct until a neighborhood structure is defined. (See Reeves and Rowe [2002] for an in-depth discussion of fitness landscapes). Thus, a dynamic landscape in which local optima shift cannot be defined at the level of benchmark construction. The dynamic benchmark is constructed as a time sequence of static problems created according to some scenarios describing how changes can occur over time, but it is often hard to—effectively and efficiently—implement these scenarios. These deficiencies are addressed in Chapter 5

3.7 Summary

It appears that any self contained research work on dynamic optimization has to deal with three issues: benchmarking, performance measures, and adaptation to environmental changes.

In general, the computation time allowed for a dynamic instance is limited, calling for algorithms that adapt quickly to environmental changes. GAs have been successfully applied to most COPs and have the potential to be effective dynamic solvers. They combine the advantages of manipulating several solutions simultaneously, robustness, and adaptability. However, once a GA converges or nearly converges around some solution, it may lose the ability to continue the search after an environment change. A simple remedy is to restart the algorithm after each change but this may prove to be both expensive and ineffective, since valuable information about the search history is discarded with every restart. A key element in the successful dynamic solver is its ability to maintain diversity throughout the search process while retaining useful past information. This requirement adds another dimension to the traditional issue of balancing diversification and intensification.

Several techniques have been used to enhance the performance of the standard GA in dynamic environments. Examples can be found in adaptive genetic parameters, memory, and parallel GAs.

For the purposes of this thesis, a promising scheme is to adopt techniques that have proved successful for dynamic continuous optimization problems. In particular techniques based on parameter adaptation and multiple populations seem to be the most promising. However, several issues have to be addressed in order to apply these techniques to COPs, namely, inter-solution distance measures, diversity measures, a way to incorporate neighborhood search techniques, benchmarking, and performance measures. Benchmarking related issues are addressed in Chapter 5 and those related to adaptation and diversity measures are addressed in Chapter 7. However, a number of basic definitions and measurements of dynamism are not uniquely defined in the literature and have to be clarified before other issues can be addressed.

Chapter 4

Basics of Dynamic Optimization, Revisited

The measurement of performance is a critical issue when reporting computational results obtained through the use of a heuristic method.

Barr et al. [1995]

4.1 Introduction

Researchers still do not agree on what constitutes a good measure of performance for dynamic solvers. Actually there is not even a unified clear cut definition to the dynamic problem itself.

In this chapter, we give some basic definitions that are necessary to understand and appreciate the work in this thesis. Most of these definitions are available in the literature but with some ambiguity. Hence, definitions of dynamic COPs and performance measures are reviewed and modified here. As well, issues related to performance measures and algorithm comparison are addressed. The chapter ends by outlining the general scheme for experimentation and result reporting.

4.2 Dynamic problems of interest

In discussing status and prospectives of the existing research on the dynamic vehicle routing problem (VRP), Psaraftis gives a general rule for distinguishing a dynamic VRP from a mere time-varying VRP [Psaraftis 1995]. This rule can be rephrased to include other COPs as follows:

> If the output of a certain formulation is a set of pre-determined solutions that are not re-optimized and are computed from inputs that do not evolve in real-time, the problem is static. If, on the other hand, the output is not a set of solutions, but rater a policy that describes how the solutions should be obtained as a function of those inputs that evolve in real-time, then the problem is dynamic.

Based on this characterization, a terminology such as on-line problems or real-time problems is more appropriate than dynamic problems. However, in this thesis, the term dynamic is used to describe such problems since it is the most common.

To give a formal definition for the dynamic COP, a definition of a static COP is first presented as given by Blum and Roli [2003].

Definition 4.1 *Combinatorial Optimization Problem*
A combinatorial problem $P = \{\mathbf{X}, f\}$ *can be defined by the following ingredients:*

- ⋆ *a set of decision variables* $V = v_1, \ldots, v_n$ *with their respective domains* $\Omega_1, \ldots, \Omega_n$ *;*
- ⋆ *an objective (cost) function* $f : \Omega_1 \times \cdots \times \Omega_n \rightarrow I\!R^+$ *;*
- ⋆ *constraints that restrict the feasible assignments to the set:*
  $\mathbf{X} = \{x = \{(v_1, \omega_1), \ldots, (v_n, \omega_n)\} |\ \ \omega_i \in \Omega_i$ *, x satisfies all the constraints* $\}$
  where $\mathbf{X}$ *is often called the* search space*;*
- ⋆ *an aim of the optimization problem: to find an optimal solution* x^* *such that* $f(x^*) \leq f(x)\ \forall\, x \in \mathbf{X}$

Since any maximization problem can be rewritten as a minimization, the minimization model above is, without loss of generality, used throughout this thesis. Also, the terms minimum and optimum are used interchangeably.

Definition 4.2 *Dynamic combinatorial problem*
A dynamic combinatorial problem can be expressed as $P(t) = \{\mathbf{X}(t), f(t)\}$*, which is a time augmented formulation of the previous definition of the static COP. That is, in the broadest definition, any of the ingredients of the COP can change with time.*

The definition captures the notion that any of the objective function, decision variables and the constraints may change with time. This definition clearly indicates that the optimal solution might change at any time due to changes in the environment. In other words, the decision maker does not have a priori knowledge of the complete problem and hence the problem cannot be solved in advance. Furthermore, the goal of the optimization process is

no longer finding a single optimal solution but, rather, tracking the shifting optima through time, since the optimal solution for one instance could be a poor solution or possibly even infeasible for the next instance.

However, the view of a *continuously* changing problem in not practical. The environment in practice remains (or has to be considered) dormant between solution requests and during the re-optimization process, since no dynamic solver adapts instantly to environmental changes. Actually, a solver receives and handles not one dynamic problem that changes with time but a series of static problems occurring over non-zero intervals of time.

Thus, a more practical treatment of the dynamic problem views the problem as a series $\mathbf{P}$ of instances; each instance is a static problem that comes into effect at time t and must be solved within a specific deadline $\bar{t}$. The ingenuity of solving this problem is in finding methods to exploit previously gathered information (often solutions of previous instances) to speed up and enhance solution of the current instance.

Definition 4.3 *Discrete-time dynamic combinatorial problem*
A discrete-time dynamic combinatorial problem can be expressed as

$$\mathbf{P} = \{\mathbf{P}_m = (P_m, t_m, \bar{t}_m),\ m = 0, 1, \ldots, m_{max}\}$$

With this formulation the duration of instance m is $\tau_m = t_{m+1} - t_m$, and the action of the environmental change Δ_m on the instance m is expressed by $P_{m+1} = P_m \oplus \Delta_m$. The maximum number of instances m_{max} can be infinite if the problem is open ended.

We use this formulation in our handling of the benchmarks in Chapter 5 and in setting measures of performance in the reminder of this chapter.

4.3 Performance measures in dynamic environments

4.3.1 What makes a superior heuristic?

Barr et al. [1995] identify a superior heuristic method as "one that quickly identifies solutions, identifies high-quality solutions, and is robust, performing well across a wide range of problems and tuning-parameter settings (and is simple to implement?)". What follows is a discussion of these characteristics in the reverse order, and placed in the context of evolutionary algorithms.

Simplicity of implementation

Simplicity of implementation of evolutionary algorithms is evident in the vast majority of evolutionary research in diverse disciplines. One of the fundamental considerations in this thesis is to maintain this quality in all the proposed adaptation strategies. We believe that simplicity of implementation is one of the most valuable assets of evolutionary algorithms that should be maintained as much as possible.

Robustness

Robustness can be defined as the ability of the algorithm to perform well over a wide range of problems without readjusting the setting of the algorithm parameters. That is, the input parameters are either kept constant for all the problems considered or allowed to vary according to features of the problem.

The importance of robustness is clear, since an algorithm that produces high quality solutions to only a few instances is of little use in practice. Yet, robustness is generally difficult to assess, since it is closely related to two problematic issues: test problems and parameter tuning. Both issues are addressed in this thesis. The generation of test problems is covered in Chapter 5, and the issue of parameter tuning is covered in Section 4.4.

Solution quality and speed of convergence

Evolutionary algorithms sample the search space alternating between focusing on high quality regions and exploring new regions. Thus, the longer an EA is run the better the quality of the final solution, but this practice necessitates a trade-off between run time and solution quality. Consequently, the appropriate measure of performance should reflect both quality and time. De Jong [1975] suggested two measures, *online performance* (ONP) and *offline performance*(OFP) , that are commonly used in comparing genetic algorithms in static environments.

These two measures dynamically abstract the evolving search into a single value that at any time assesses the algorithm performance up to that point, but whereas ONP provides information on the convergence by addressing the population as a whole, OFP focuses on the ultimate goal of the optimization process by focusing on the the best-so-far value only [Grefenstette 1999].

At time step T both measures can be defined as follows:

$$ONP(T) = \frac{1}{T}\sum_{t=1}^{T} e_t \tag{4.1}$$

$$OFP(T) = \frac{1}{T}\sum_{t=1}^{T} \varepsilon_t, \quad \varepsilon_t = max\left\{e_1, e_2, \ldots, e_t\right\} \tag{4.2}$$

where e_t is the fitness of the individual evaluated at time step t.

4.3.2 Measures in use

In dynamic optimization, the typical goal is to track optima shifting through time. Solution quality, therefore, is determined by how close the obtained solutions are to the shifting optimal solutions. Consequently, an effective performance measure is one that abstracts

the quality of solutions over time. Offline performance, as given in Equation 4.2, is of little benefit in dynamic environments, since the value of previously found solutions is irrelevant after an environmental change.

Several alternative measures have been devised. Mori et al. [1997] used an *adaptation performance* that averages the ratios of the best found solution to the optimal solution over the length of the entire search process [Trojanowski and Michalewicz 2000]. Hadad and Eick [1997] proposed using square error to the best value instead of simply the error. Trojanowski and Michalewicz [2000] proposed an *accuracy measure* based on the difference between the best found and the optimal value.

The majority of researchers use the *best-of-generation* value or the *current best performance* (CBP) value [Goldberg and Smith 1987; Cobb 1990; Dasgupta and McGregor 1992; Grefenstette 1992; Mori et al. 1996; Vavak et al. 1996; Angeline 1997; Lewis et al. 1998; Grefenstette 1999; Yang 2003].

If R runs are used, then CBP at generation g is given as:

$$CBP(g,R) = \frac{1}{R}\sum_{r=1}^{R}\varepsilon_g^r, \qquad \varepsilon_g^r = max\left\{e^r \;\middle|\; e^r \in pop_g^r\right\} \tag{4.3}$$

where pop_g^r is the set of fitness values of the individuals in the population at generation g during run r. This measure is better suited to dynamic problems than the traditional offline performance measure.

A common method for comparing algorithms is to plot CBP versus generations for each algorithm and to compare the plots visually [Branke 2001]. However, such comparison is neither easy nor conclusive, nor is it suitable for comparing more than two algorithms on complex dynamic problems (in which the optimal value is not necessarily monotonically changing). In such cases the resulting plots could be intermingled, such as those shown in Figure 4.1, prohibiting the determination of which of the competing algorithms is performing best overall. Remember that because the problem is dynamic, we need to consider as many solutions as the number of environmental changes.

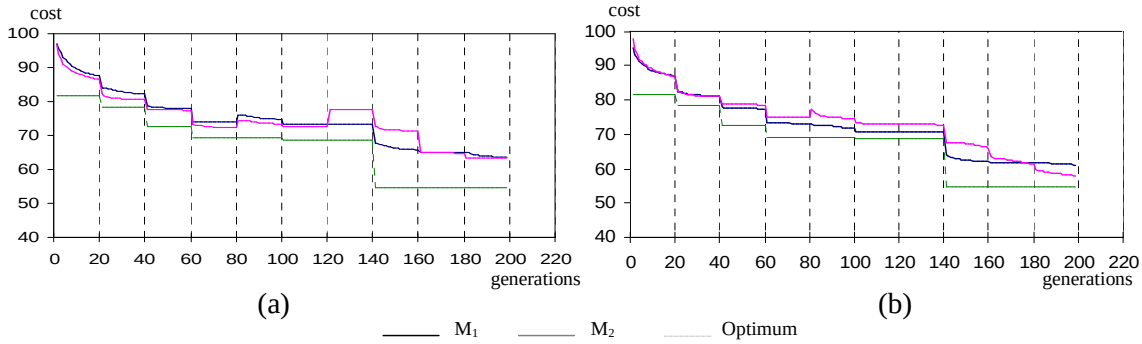


Fig. 4.1: Traditional comparison of best of generation curves. Two models M_1 and M_2 are compared against the actual optimal solution. It is not easy to decide on which model is performing better.

Branke [1999a] proposed a modified offline performance measure to overcome this shortcoming by resetting the computed best-so-far value at every environmental change. Still the modified *OFP* does not mask out variations in the value of the solutions found, and hence instances with large optimal values will dominate others. This shortcoming can be avoided when testing benchmarks by normalizing the evaluations by dividing them by their corresponding optimal solution values, which are usually known in advance in the case of test problems. Younes et al. [2005] used a normalized modified offline performance measure that takes into account changes in the optimal solutions and also averages the results over the considered runs.

Morrison [2003] suggested a *Collective Mean Fitness* (F_C) measure of the values of the best-of-generation averaged over the number of generations and over the number runs. Some researchers [Goldberg and Smith 1987; Dasgupta and McGregor 1992; Grefenstette 1992; Mori et al. 1996; Yang 2003] used measures based on the average of the population usually in addition to those based on the population best. With such measures, high values would indicate that the algorithm has identified a promising region and is keeping the population largely centered around some high quality solution [Grefenstette 1999].

Other measures are based on solution position rather than solution value. They require knowledge of the position of the current optimal solution, which is often unavailable, and thus their use is limited to test problems in which the optimal solutions are known a priori. However, they have the advantage of being independent of fitness scaling [Weicker 2002], hence some researches have opted to use them. For example, Salomon and Eggenberger [1997] based their performance measure on distance from the population average point to the optimal solution. This measure is analogous to the online performance measure, since it considers the whole population instead of the best solution only. Weicker and Weicker [1999] used a similar measure in which the distance measured was the minimal distance of the individuals in the population to the current optimum. This measure is more appropriate than the previous one, since the nearest individual to the optimum is often the best solution in the population. More details on these measures can be found in the survey of Weicker [2002], which also presents a systematic approach for examining performance measures for dynamic optimization problems. This survey classifies performance measures by the knowledge they need:

- Knowledge on the position of the optimum is available. These are the least used measures since the required knowledge is only available in some benchmarks.
- Knowledge on the best fitness value is available. This knowledge is usually available in benchmarks only.
- No global knowledge is available. This is the usual case when tackling a new problem.

4.3.3 Classification of performance measures

There are several publications that review performance measures such as Branke [2001], Weicker [2002], and Morrison [2003]. However, no attempts at classifying the measures are known to this author except the previously mentioned categorization of Weicker [2002]. This section offers a more direct and simple classification, as shown in Table 4.1.

Table 4.1: Classification of performance measures

	population based	population-best based
genotypic	I	II
phenotypic	III	IV

Performance measures are divided into four groups: First the measure is classified according to whether it targets the whole population or the best solution only. Then, the measure is classified according to whether it is genotypical or phenotypical.

1. Entire population versus population-best

   (a) Measures based on the entire population:
   These measures reflect the the influence of the whole population on the search process. Alone, they cannot be used as criteria for comparing competing algorithms since they do not assess the algorithm ability in achieving the optimization goal. Still, they can be used to assess the affectivity of operators that are applied at high rates, such as crossover, but would not be useful in assessing, for example, the effectiveness of mutation because mutation is often applied at a very low rate.

   (b) Measures based on the population-best:
   These measures are mainly concerned with how good the best found solution is and how close it is to the optimal solution. These measures are the most convenient and the most commonly used, since they try to assess the success of the algorithm in reaching its ultimate goal.

2. Genotypic versus phenotypic

   (a) Genotypic based measures:
   These measures are usually given in terms of distance to the optimal solution.

They have the advantages of being immune to fitness scaling, thus their effectiveness is not diminished when the optimal value changes from one instance to another. However, they require that the position of the optimal solution be known for each instance of the dynamic problem. Another disadvantage of these measures is that although genotypic distance may reflect the closeness to the optimal solution it does not reflect the optimization goal, since a solution that is close to the optimal position may be inferior to another solution that is positioned farther from the optimal solution. In this case, the genotypic measure will favor the inferior solution, which can adversely influence the results.

(b) Phenotypic based measures:
Phenotypic based measures are more goal oriented since they usually measure the fitness of the solution. Examples are online performance, offline performance, and best-of-generation. The main disadvantage when using these measures in dynamic problems is that they are biased to instances with large optimal solutions.

In short, the most appropriate measures for this thesis are in class IV of Table 4.1; Class III measures can be used too, but only to complement class IV measures. Furthermore, measures in both classes can be made to assess the speed of convergence of the algorithm as well by accumulating the measures over time.

4.4 General considerations

This section highlights some issues that have to be addressed during experiment setup and reporting.

4.4.1 Quality-speed tradeoffs

As mentioned earlier, a trade-off between run time and solution quality is often necessary. The correct balance depends on the application at hand, and thus it is generally left to the user to decide on the the best balance according to his/her understanding of the problem circumstances. It will be rather pointless to argue which is the determining factor in comparing algorithms. Instead, we propose basing the comparisons between various strategies on solution quality only, while limiting the run time allowed for the dynamic solver. For each instance of the dynamic problem, the dynamic solver will be allowed a fraction of the time (number of generations or evaluations) taken by a static algorithm to reach an acceptable solution of the same instance, as illustrated in Figure 4.2. There are two additional advantages to limiting run time between instances. First, comparison between competing algorithms will be more effective because differences in performance between various algorithms tend to diminish with time, which render comparisons after extended periods rather pointless. Second, using unequal intervals between environmental changes in the problem will bias the final aggregate measure towards longer intervals.

4.4.2 Effects of stochastic sampling

Evolutionary algorithms, like most metaheuristics, incorporate some stochastic components to sample the search space. Thus, an evolutionary algorithm may produce different results with different seeds in the random function used. That is, running an algorithm several times on the same instance can give different results. For this reason, we report not only the best obtained results but also the average and the worst results obtained over the runs conducted, together with their standard deviations.

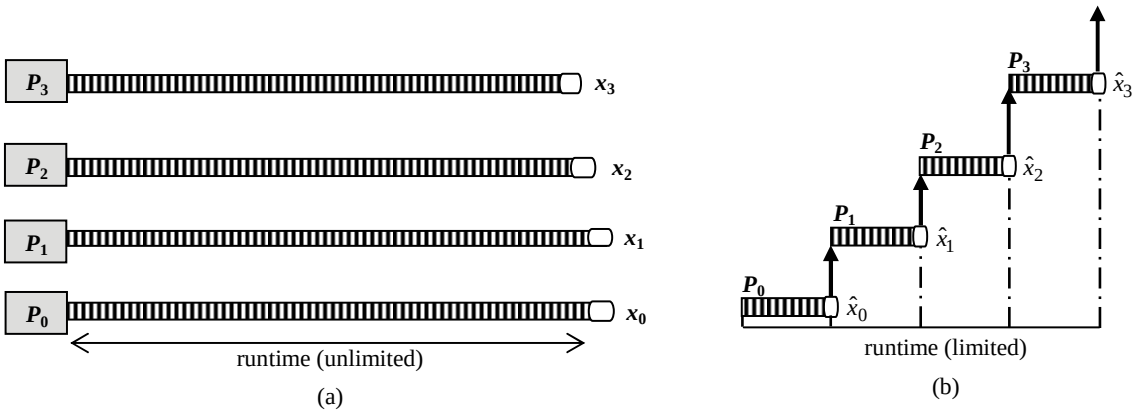


Fig. 4.2: Limited run time for dynamic Problems. (a) Static problems are first solved independently and with relatively unlimited run times. (b) Instances in the dynamic problem are solved consequently with limited time per instance. The closeness of DS solutions $(\hat{x}_0, \hat{x}_1, \hat{x}_2, \hat{x}_3, \ldots)$ to static solutions $(x_0, x_1, x_2, x_3, \ldots)$ reflects both solution quality and run time.

4.4.3 Changes in values of optimal solutions

One important factor that is not specifically addressed in the literature is the sensitivity of performance measures to extreme solution values. Measures that sum up the algorithm performance in a single value such as the modified offline performance measure and the collective mean fitness measure do not take into account the difference between solution values throughout the run. Such measures will be biased toward instances with large optimal values. This shortcoming can be overcome in test problems, where optimal or near optimal values are known in advance, using a simple normalization of the performance measure before averaging [Younes et al. 2003, 2004, 2005]. However, in cases where a basis for normalization cannot be pre-determined, it might be necessary to augment aggregate performance measures by outputting the evolution of the measure from one instance to another.

4.4.4 Units of time

The time required to evaluate a single individual is traditionally taken as *unit time* in result reporting. For example offline and online performance measures are usually expressed in terms of the number of evaluations, which has the advantage of making results less machine dependent. Still, this practice produces large amounts of data as there can be many evaluations at every generation. Furthermore, it does not reflect the mechanism of EAs, in which a generation is the basic iteration. In other words, the smallest time interval in an evolutionary run that encompass all genetic operations is one generation. Thus, it is more appropriate to report performance measures over a number of generations rather than over a number of evaluations.

4.4.5 Parameter tuning

One of the major shortcomings of heuristic methods has to do with the tuning of parameters. It is time-consuming, it can promote unfair comparisons between algorithms, and it can counter claims of algorithm robustness.

In a traditional EA, the fundamental tunable parameters are population size, mutation rate, and crossover rate, which would appear to be too few to constitute any difficulty. However, they are interrelated in a way that is still not completely clear (another consequence of lacking a proper theoretical background). This difficulty is more aggravated in contemporary EAs which incorporate additional techniques such as multiple populations, diversity controlling schemes, or even other hybridizing heuristics, with the consequence of more parameters to tune, requiring more time and greater efforts for tuning.

In general, it is possible to tune the algorithm parameters on a specific problem so that it produces very good results on the instance at hand. Such a practice raises several questions that seem to be unanswered. What is the best way to do tuning? Should it be done individually or collectively, randomly or deterministically? How much time and effort

can be allocated to tuning? How far can one go on tuning without biasing fair comparison between algorithms and without prejudicing claims to algorithm robustness?

Questions of this sort have been raised by many researchers [Barr et al. 1995; Hooker 1996; Johnson 2002]. Still, there do not seem to be any satisfactory answers. The amount of parameter tuning that can be done is also a subject of trade-off between solution quality from one side and experimentation time, algorithm robustness, and fairness of comparison on the other side. Ultimately, the decision to the amount of time that can be allocated to tuning is more or less left up to the the one carrying out the experiments.

On the other hand, there is the comforting findings of Grefenstette [1986], who in his search for the optimal set of parameters, concluded that a GA is basically so robust that the setting of parameters is not critical within quite wide margins.

Parameter tuning is addressed on two fronts in this thesis. First, experimentation is generally carried out so that parameters are either pre-determined from preliminary experiments or changed in a controlled manner without fine tuning (see the next section for details). Second, diversity is used to control genetic parameters instead of having them specified by the user. The details of this approach are presented in Chapter 7.

4.5 Experimentation scheme

Experiments in this thesis are carried out in three phases: preliminary, basic, and comparative.

4.5.1 Preliminary experimentation phase

The preliminary phase of experimentation contains experiments to test wide ranges of algorithm parameters, mostly on static instances. Most of these experiments are rather trial and error involving a large number of parameter combinations and hence are not reported, even though they had taken up a sizeable proportion of the overall experimentation time.

The main purpose of these experiments, however, is to roughly establish *working* ranges of the algorithm parameters by excluding values that tend to produce unreasonable results (very low solution quality or excessive number of generations). In this way, efforts of parameter tuning in experiments carried out on dynamic problems in the next phase are reduced. Furthermore, biasing in dynamic tests is reduced and any claims to algorithm robustness are more justified.

4.5.2 Basic experimentation phase

This phase constitutes the major experimental work. It consists of the experiments necessary to investigate the contribution and the effect of the individual components within the developed algorithm by monitoring the evolution of several variables over time (number of generations). The main output variables are population best solution, population average and standard deviation, and the best-so-far solution. The evolution of genetic control parameters and population diversity measures are also reported when needed. Algorithm parameters are varied in a controlled way, within ranges established in the literature or found during preliminary experiments. In other words, parameters are not fine-tuned but are varied in a predetermined manner. In this way, the effect of parameter setting on algorithm performance is also investigated. Where an individual parameter is investigated, several values within the feasible range of the parameter are used. However, when several parameters are investigated simultaneously, only three values (high, medium, low) of each parameter are tested in order to reduce the number of combinations to test.

4.5.3 Comparative experimentation phase

This phase is mainly concerned with comparing algorithms on benchmark problems, thus the performance measures used here are aggregate measures that abstract algorithm evolution over the entire run. The principal criterion for comparison is the normalized current-

best performance over a limited number of generations—to reflect both solution quality and time and to avoid bias to instances with long periods, as described earlier. As well, to consider the effects of randomness, several runs are used. The results are reported in terms of the runs-best solution, runs-average solution, and runs-worst solution.

The experiments are conducted either on fixed parameters (pre-determined from experimentation on other problems during the previous phases) or on variable parameters (changing with time, diversity measures or environmental changes) to avoid problems associated with the use of fine-tuned parameters described in Section 4.4.

Since the test problems considered in this thesis are minimization of cost functions, the related performance measures are directly based on the solution cost rather than on the fitness. First, a mean best of generation (MBG) is defined after G generations of the r^{th} run as:

$$MBG(G,r) = \frac{1}{G}\sum_{g=1}^{G}\left(\frac{\varepsilon_g^r}{\hat{c}_g}\right), \qquad \varepsilon_g^r = min\left\{e_\theta^r \;\mid\; t_g \leq \theta < t_{g+1}\right\} \tag{4.4}$$

where e_θ^r is the fitness of the individual evaluated at time step θ and run r, t_g is the time step at which generation g started, and $\hat{c}_g$ is the the optimal cost (or the best known cost) to the problem instance at generation g.

Then, the algorithm's performance on the benchmark over R runs can be abstracted as follows:

$$RunsAverageMBG(G,R) = \frac{1}{R}\sum_{r=1}^{R} MBG(G,r) \tag{4.5}$$

$$RunsWorstMBG(G,R) = max\left\{MBG(G,r) \;\mid\; r \in R\right\} \tag{4.6}$$

$$RunsBestMBG(G,R) = min\left\{MBG(G,r) \;\mid\; r \in R\right\} \tag{4.7}$$

With these definitions, smaller values of the performance measure indicate improved performance. Moreover, since *MBG* is measured relative to the value of the best solutions found during benchmark construction, it will in general exceed unity. Less than unity values if encountered indicate excellent performance of the corresponding model in that the dynamic solver with limited (time per instance) budget outperforms a static solver with virtually unlimited budget.

Statistical analysis

Statistical t-tests that are used to compare the means of two samples can be used to compare the performance of two algorithms. The typical t-test is performed to build a confidence interval that is used to either accept or reject a null hypothesis that both sample means are equal. In applying this test to compare the performance of two algorithms, the measures of performance are treated as sample means, the required replicates of each sample mean are obtained by performing several independent runs of each algorithm, and the null hypothesis is that there is no significant difference in the performance of both algorithms.

However, when more than two samples are compared, the probability of multiple t-tests incorrectly finding a significant difference between a pair of samples increases with the number of comparisons Hochberg and Tamhane [1987]. Analysis of variance (ANOVA) overcomes this problem by testing the samples as a whole for significant differences.

Therefore, in this thesis, ANOVA is performed to test the hypothesis that measures of performance of all the evolutionary models under considerations are equal. Then, a multiple post ANOVA comparison test, known as Tukey's test, is carried out to produce 95% confidence intervals for the difference difference in the mean best of generation of each pair of models.

4.6 Summary

This chapter gave an overview of basic aspects of dynamic problems. As a still developing field, dynamic optimization has some attributes that are vague, application dependent, or simply flawed. Therefore, it is essential to review these attributes in a framework congruent with this thesis.

The definition of the dynamic problem of interest and the criteria of performance measure were revised according to the goals of this thesis. This chapter also described schemes used for experimentation and result reporting that take into consideration effects of stochastic sampling, tradeoff between solution quality and cost, and fair algorithm comparison.

With the basic issues covered in this chapter, the next chapters focus on the main aspects of the thesis: benchmark generation and algorithm development.

Chapter 5

Benchmark Generation

Every time an algorithm uses a particular operator, it can be seen as traversing an edge, or taking a "step," on the landscape defined by the operator in question. I call this the "One Operator, One Landscape" ...

Jones [1995]

5.1 Introduction

Generating benchmarks for dynamic environments is considered an important goal in this research for two reasons. One reason stems from the well-known implications of the "no-free-lunch" theorems of Wolpert and Macready [1997], which show that any comparison between competing algorithms must be supported by a set of test problems [Grefenstette 1999]. The other reason is that there are still no agreed upon standard test sets for dynamic optimization.

This chapter addresses two overlooked issues in the generation of dynamic COP benchmarks: First, generating a sequence of instances that effectively test adaptability in algorithms generally requires solving each instance explicitly or at least determining upper bounds for solutions. Thus, computational costs can restrict the size of the base problem and length of sequence of the static instances constituting the dynamic benchmark. Therefore, a scheme for benchmark generation that needs only solve the initial instance of the dynamic problem is proposed. Second, since combinatorial problems assume different structures, their test cases tend to be too problem specific, and their results are often considered application dependent. While this may hinder a general treatment for static COPs and prevents the use of a general-purpose benchmark generator for dynamic COPs, it does not prevent a general treatment of their dynamic issues. In fact, it makes a general framework both interesting and challenging. To the author's knowledge, there exists no such framework in the literature. The closest work perhaps is that of Weicker [2000], but that framework was specifically designed to encompass the general BGs introduced a year earlier [Branke 1999b; Grefenstette 1999; Morrison and De Jong 1999; Trojanowski and Michalewicz 1999; Weicker and Weicker 1999]. It uses their way of describing an overall function in terms of several elementary functions or peaks, each having its own rule of dynamics (a rule of dynamics defines how the function coordinates and values are varied). Since these BGs are not designed for dynamic COPs, their framework

is also unsuitable for COPs. This chapter, therefore, presents a framework for dynamic benchmark generation for COPs. The framework aims to establish a basis for benchmark generation. It also enhances the ability of the benchmark problem to compare adaptability, by restricting environmental changes to those inducing adaptation.

5.2 Analysis of environmental changes in COPs

Changes can take place in a dynamic problem in many ways; in fact there can be an infinite number of scenarios in which changes can affect elements of an optimization problem. However, some of these changes can be so insignificant for benchmarking purposes that the resultant dynamic problem can still be considered static. Moreover, many scenarios, though distinct from one another, have the same effect on the DS and, as such, can lead to ambiguous results.

In this section, we start by categorizing environmental changes in order to isolate and address relevant issues later. Environmental changes that constitute a dynamic optimization problem can be divided into two basic categories: *dimensionality changes* and *non-dimensionality changes*. Changes in the first category correspond to adding variables to the optimization problem or removing variables from it. Examples of such changes are the insertion or the cancellation of assignments in a vehicle routing problem, orders in a job shop scheduling problem, cities in a TSP, and objects in a knapsack problem. These changes require a corresponding alteration in solution representation.

Changes in the second category result from variations in the values of the parameters and coefficients of the problem constraints and objective function. As some of these values change from one instance to another, a one time optimal solution might lose quality relative to another solution that was inferior to it in the past. Examples include changes in the capacity of the knapsack problem or in the weights or the values of its objects. Other examples can affect the travel time on some roads in a vehicle routing problem, and the

processing time and ready dates of a scheduling problem.

Moreover, benchmark problems involving non-dimensionality changes are harder to construct. Whereas the construction of benchmarks incorporating dimensionality changes may involve simply adding or deleting variables, the construction of benchmarks incorporating non-dimensionality changes is not as straightforward and is potentially complicated by what we will refer to as *dynamically insignificant changes*.

5.2.1 Dynamically significant changes

In applying non-dimensionality changes, the differences between successive instances can inadvertently be made dynamically insignificant; that is, the introduced changes are so trivial that any optimizing algorithm exhibits the same behavior whether the change is made or not. A dynamically insignificant change can thus be defined as an environmental change that does not alter the structure of the problem instance. Such a change would keep the optimal solution unchanged and hence does not induce any adaptation from the dynamic solver. In a knapsack problem, for example, reducing the value of an object that is not in the optimal solution (or increasing the value of an object that is in the optimal solution) will not alter the optimal solution. Furthermore, reducing the value of an object in the optimal solution (or increasing the value of an object not in the optimal solution) will not alter the optimal solution unless the changes are sufficiently large. Similarly, increasing or decreasing travel time on a road in a TSP may not be significant.

Algorithm behavior due to insignificant changes can be clearly pictured with the help of an example from continuous optimization, as shown in Figure 5.1. In this example, starting from an initial solution x_0, a hill climber should be able to climb $f_1(x)$ and eventually end at the peak at x^*. A solver will consistently return the same solution with $f_2(x)$ and $f_3(x)$, and with any similar function differing in peak's height—regardless of the magnitude of the difference as long as there is one local optimum in the search region. Clearly a dynamic problem in which the environmental changes make the objective function oscillate among

the functions $f_1(x)$, $f_2(x)$ and $f_3(x)$ is useless as a dynamic benchmark since even the simple hill climber can play the role of a dynamic solver due to the fact that it need not exhibit any kind of adaptation to changes taking place in this problem.

For continuous problems a BG can explicitly shift the location of the optima and thereby make environmental changes dynamically significant. However, since one cannot define a unique (algorithm independent) landscape as a COP benchmark (as was discussed in Section 3.6.3) , one would not be able to identify which of the applied non-dimensionality changes are dynamically insignificant. This suggests that a minimum requirement to ensure dynamic significance is that the optimal solutions of all instances in the benchmark be known. This requirement adds to the efforts of selecting the changing parameters and their corresponding values, and might even necessitate solving newer instances before actually adopting them as part of the benchmark.

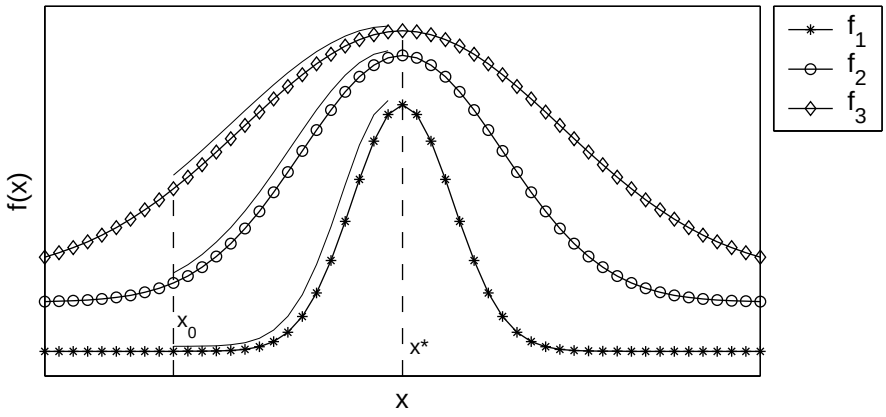


Fig. 5.1: Dynamically insignificant changes in a continuous function. Oscillating among the three functions does constitute significant changes.

5.2.2 Misleading patterns of change

In addition to being of little value from a testing perspective, dynamically insignificant changes give rise to misleading patterns in the *measures of dynamics*. A measure of dynamics is one that quantifies certain features of the change taking place in the dynamic problem. The most frequently used measures of dynamics are *severity* and *frequency* of change of the underlying parameters of the problem.

Measures of dynamics are varied in a benchmark problem to produce different patterns of environmental changes that can be used to test the ability of the algorithm to adapt in diverse environments. However, if these patterns are composed—even partially—from dynamically insignificant changes, they will be less effective in inducing adaptation than expected. In such cases, results of experimentation will be misleading if attributed to the input pattern of change as is often done, since there actually are much simpler patterns that produce the same effect on the the same DS. The following examples illustrate several cases of non-dimensionality changes to show how the *effective patterns* can be considerably different from the input patterns.

Example1: Changes affecting an optimal TSP edge

Consider a dynamic TSP where dynamism is introduced by changing the length of an edge in the optimal tour. To ensure dynamic significance, the edge length must be increased to a sufficiently large value that renders the current optimal solution suboptimal. Call this value the *level of significance*. Figure 5.2 shows some possible changes in the edge length, with p_0 denoting the original edge length and f_1, f_2, and f_3 denoting three possible patterns of change in the edge length.

It can be seen that throughout the three patterns, the edge length never reaches its level of significance. As environmental changes, these patterns do not differ from one another. In fact, their effect on the dynamic solver is similar to that of the original static pattern p_0, and we describe these cases as *dynamically equivalent* to p_0. There are an infinite number

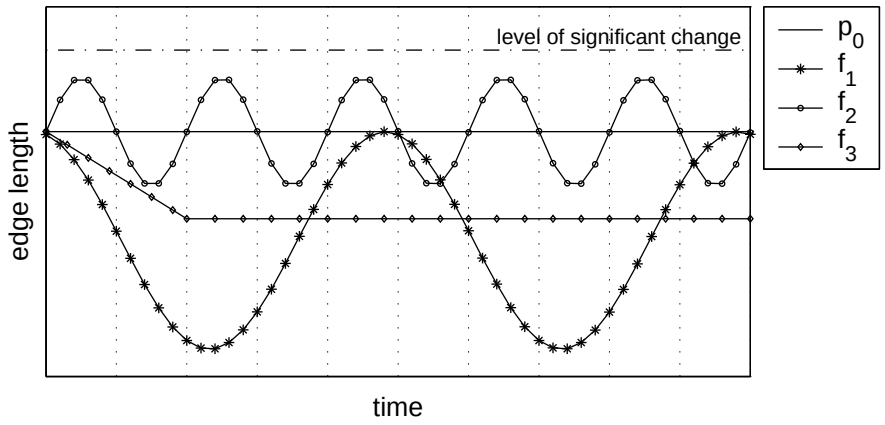


Fig. 5.2: Intricate but insignificant edge changes. Even the most complex input pattern may not not be challenging; f_1,f_2, and f_3 are in effect similar to the stationary pattern p_0

of patterns of this kind and though they might be very complex, they are worthless in testing adaptation in algorithms.

In other cases, where edge length increases beyond the level of significance, the overall pattern of change can still be misleading. In Figure 5.3 for example, the edge length in the pattern f_4 goes above the level of significant change. However, as an environmental change, this patten is equivalent to that of the simple pattern f_{4e}, since at any instance in time the explicit edge length is not particularly important; moreover, if an increase in length is dynamically significant at one time, subsequent increments in the same edge will not constitute dynamically significant changes.

Example2: Changes affecting a non-optimal TSP edge

In this example, we examine cases in which the changing edge in a TSP is not on the optimal tour. Change of this kind can only be dynamically significant if the new length is sufficiently smaller than the previous length (i.e., below the level of significance). Figure 5.4 shows a possible pattern of change (f_5) which is dynamically equivalent to the pattern of

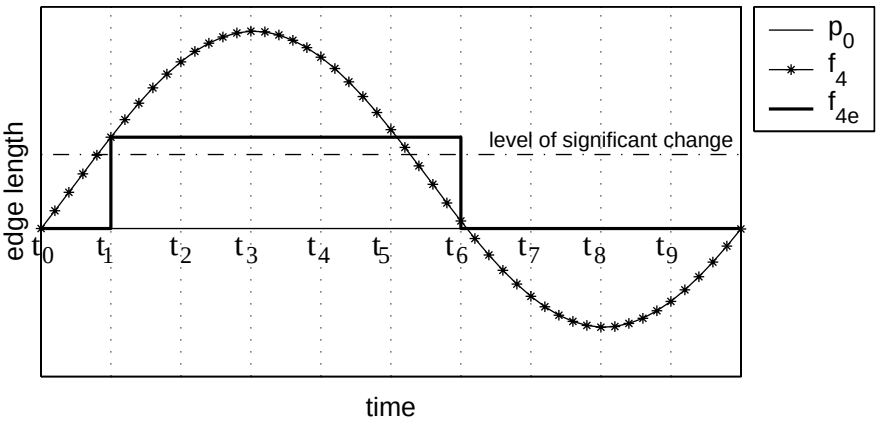


Fig. 5.3: Suppose that the dynamic solver is required to supply solutions to instances specified by the vertical dashed lines. The change from $f_4(t_0)$ to $f_4(t_1)$ is dynamically significant because it crosses the level of significant change. Subsequent values of f_4 remain above the level and hence changes of f_4 at t_2, t_3, t_4 and t_5 are not dynamically significant. But at t_6 there is a significant change while the changes of f_4 at t_7, t_8, and t_9 are not significant. The pattern f_{4e} shares the same dynamically significant changes (at t_1, t_6) with f_4 and hence both patterns are considered dynamically equivalent.

change labelled f_5e.

Example3: Changes affecting knapsack capacity

Misleading patterns occur with non-dimensionality changes of the knapsack problem which affect object value, object weight, and knapsack capacity. Changes in the first two parameters can be characterized in a manner similar to those discussed in the previous examples. The third type of change—which is also a popular way of constructing dynamic knapsacks—is more interesting since there are several levels of significance involved. In fact, there are as many levels as there are objects in the problem. Increasing the capacity beyond the highest level enables the knapsack to include all the objects whereas reducing the capacity below the lowest level precludes all objects. Figure 5.5 shows a possible dynamic knapsack constructed by varying the weight capacity f_6 over time where p_0, as before, denotes the

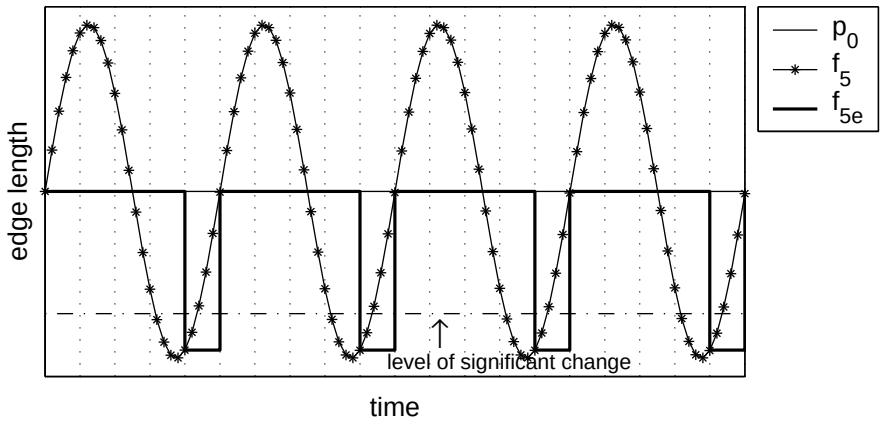


Fig. 5.4: Partially significant change on an optimal edge. Only small enough decrements in f_5 are effective and hence show in f_{5e}.

initial capacity. Four of the possible levels of significance are shown: level U_1 indicates the capacity at which the knapsack can admit an additional object, and level U_2 indicates the capacity that permits yet another object. The two other significant levels in the figure have the opposite meaning. At L_1 one object has to be removed and at L_2 a second object has to be removed. A change in capacity that yields a level between any two consecutive levels does not constitute a dynamically significant change and only changes that cross a level will be significant. Consequently, the input pattern f_6 is effectively reduced to the simpler one, f_{6e}.

These examples used simple elementary changes to illustrate how patterns of change can be misleading due to the presence of insignificant changes. This issue becomes more complicated in the general cases where an individual change is composed of several components involving different elements of the optimization problem. For example, Figure 5.6 shows the results of experiments done at the early stage of this thesis to compare the two strategies (M1 and M2) of dynamic adaptation on a simple ten-city TSP problem. The problem is made dynamic by randomly changing the cost of some edges, one at a time. At

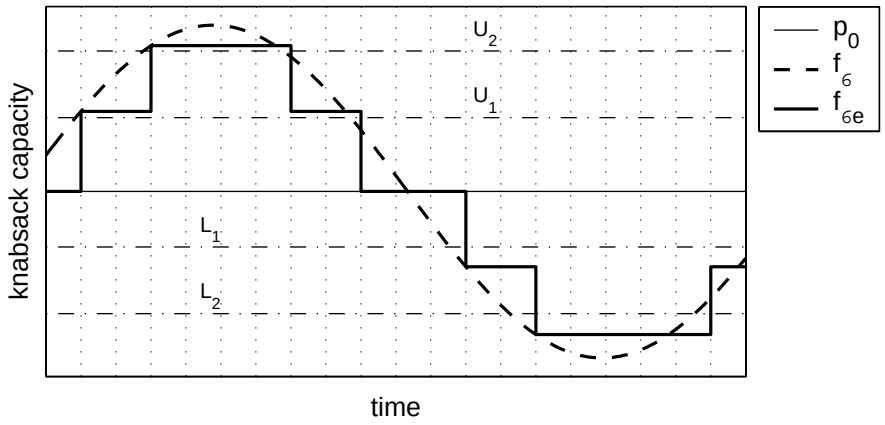


Fig. 5.5: Partially significant change in knapsack capacity. Only changes that cross L_1, L_2, U_1, or U_2 will be effective and show in f_{6e}

every environmental change a random edge is selected from the problem and its length is changed randomly (increased by a factor of 1.5 or decreased by a factor of 0.5). At first, it was expected that the optimal solution would also change randomly, either increasing or decreasing at every environmental change. But, as can be seen in the figure, the optimal solution did not increase. This is because increasing the length of the edge will affect the optimum tour only if this edge happens to be part of the optimum tour. This has a low probability of occurrence since the number of the edges on a single tour constitute only a fraction of the total edges of the problem. (The probability of this event occurring in this instance was 0.5*10/45 =0.11).

In cases with more complicated patterns, the effect of any existing significant components of change may unwittingly be attributed to all components of the change—the significant and the insignificant ones. Therefore, in order to study how dynamic solvers tackle different patterns of change, one needs to determine the *simplest effective pattern* by identifying insignificant changes in the input parameter. This is not a trivial task and may require solving many candidate instances before actually adopting any of them as a

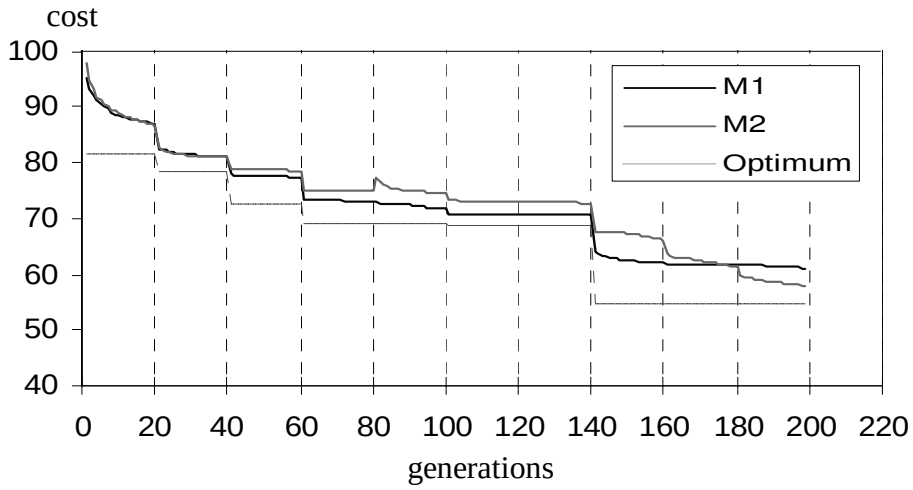


Fig. 5.6: Random edge change in a TSP

part of the benchmark problem.

5.3 Benchmarks general requirements, revisited

Based on the discussion in the previous section, we re-define the general features desired in an effective benchmark generator for dynamic combinatorial problems as follows:

- The generator should be able to vary all elements of the optimization problem.
- The generator should be able to provide a wide variety of dynamics; that is, variation of the changing element can take different patterns with controllable degrees of complexity.
- The generator should be able to test adaptive performance of a tested solver by ensuring that the environmental changes are dynamically significant.

- The generator should produce test problems in such a way that the optimal values are known, and ideally the optimal solutions are also known.
- The generator should not be too problem specific. Still, the static problems and scenarios of environmental changes on which the benchmark is constructed should permit conjecture to more realistic problems.

5.4 Mapping-based benchmark generation

It was noted previously that before a complete dynamic benchmark problem can be constructed it might be necessary to solve many problem instances. This requirement is very expensive if not impossible, especially when the problem in question belongs to the NP-hard class. Two options can be used to alleviate this difficulty. The first one limits the number and the size of the static instances constituting the benchmark. Of course, using small problem sizes may make the problem trivial, and using few instances may restrict the dynamics that can be modelled. The second option uses results of previously established algorithms to compare with those of the DS being testing. This option has two disadvantages: first, evaluations would be of a relative nature (to the quality of other algorithms). Second, as we do not have a wealth of results for other algorithms, the choice of the comparing algorithms and the way they are run can severely change the outcome of the evaluations.

These difficulties motivated the author to introduce a general scheme to generate benchmarks of arbitrary number and size of instances. The scheme is based on the work of Younes et al. [2003] who introduced a benchmark generator for a dynamic TSP with long sequences of static instances and controllable characteristics of dynamics. In what follows, we illustrate how the underlying concept can be generalized for COPs.

The main idea is centered on the fact that GAs do not handle solutions directly but work on their encoding. Thus, an environmental change can be simulated at any time by

manipulating the mapping function that encodes solutions into chromosomes: the actual environment remains stationary, but any stored chromosomes (in the population or in any other kind of memory) will map into new solutions. To the DS, the change in the stored potential solutions amounts to an environmental change, and the DS will be compelled to adapt to this change in order to track the shifting optima. To the benchmark user, the entire sequence of instances has the same optimal solution, and only the initial instance has to be solved. Consequently, by controlling the change in the mapping, the user can present the DS with a dynamic benchmark with controllable dynamics without the usual limitation on the problem size and the number of instances. The following examples illustrate the idea and its applicability to some problems.

5.4.1 Example 1: dynamic knapsack problem

This example considers the seven-object knapsack problem illustrated in Figure 5.7. In this setting, a candidate solution consisting of the objects O_1, O_4, O_3, O_2 and O_7 is encoded by a mapping function to the chromosome (B_1 B_4 B_3 B_2 B_7). Then if, for example, the objects associated with the labels B_3 and B_5 in the mapping function are swapped, the same chromosome (B_1 B_4 B_3 B_2 B_7) will represent a different individual, which consists of the objects O_1, O_4, O_5, O_2 and O_7. If all the chromosomes in the population are treated in this way, they will point to different individuals, and any re-evaluation of the population will reveal that it now consists of individuals which are actually different from their predecessors—some of the new individuals might even turn out to be infeasible. By repeatedly changing the encoding, a sequence of instances can be generated from a single problem. A dynamic solver will treat the sequence as a dynamic problem in that it will try to adapt to changes in the problem; whereas the benchmark designer knows the values of the optimal solutions to all the generated instances, since they are actually the same. Thus, in using this mapping-based scheme one needs only to optimize the initial instance of the dynamic problem.

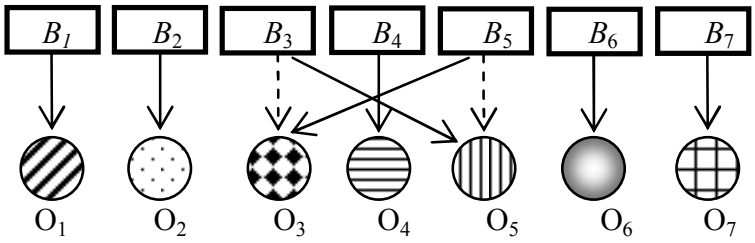


Fig. 5.7: Mapping swap in a KP

The severity of change in the mapping-based benchmark can be expressed as the number of interchanges imposed on the mapping function per environmental change. The change frequency can be expressed in terms of the number of iterations or evaluations between consecutive changes.

5.4.2 Example 2: dynamic traveling salesman problem

In a manner similar to that just described for the dynamic knapsack problem, benchmarks for the dynamic TSP can be generated by swapping city labels in the mapping function. With such a swap, every chromosome in the current population will decode into a new solution. Figure 5.8 shows a possible tour in an example of an eight-city TSP and the chromosome representing that tour. If the labels of cities 3 and 6 are interchanged, the same chromosome will point to a different tour as shown in the figure. As in the previous example, this mapping swap can be considered an environmental change, and the severity of this change will be proportional to the number of underlying interchanges.

Although mapping-based change may not reflect real-world situations, the technique serves the goal of generating dynamic test COPs with known optima without the usual limitations on the sequence length and instance size. Other COPs can be treated in a similar way to produce dynamic versions. For example, the mapping-based swap can be applied to job shop scheduling or the labels of machines, parts or operations in ways that make

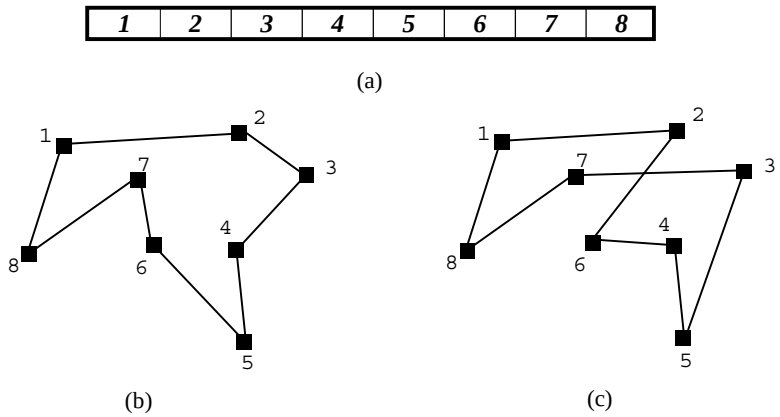


Fig. 5.8: Mapping swap in a TSP. The chromosome in (a) represents the tour in (b) and later the tour in (c).

a chromosome in a population represents a different solution. A facility location problem can be made dynamic by interchanging locations labels or facility labels. Furthermore, complicated test problems that are more real-world oriented can be constructed by combining mapping-based procedures with dimensionality and non-dimensionality changes. In any case, a mapping-based BG offers a simple, quick and easy way to generate problems that can be used to test and compare dynamic solvers. The effectiveness of this technique is compared with real-world environmental changes in Chapter 6, and its applicability to more complex problems in demonstrated is Chapter 8.

5.5 Generalized framework for benchmark generation

The idea of constructing general combinatorial BGs similar to those available for continuous optimization is appealing; however, it seems a distant goal, since COPs in their static forms tend to take very distinct structures—too distinct to permit a single BG for the diverse COPs we have. Still, this fact makes the idea of grouping dynamic combinatorial

benchmarks under a common framework both interesting and challenging.

This section presents a framework to encompass different types of environmental changes used currently to construct benchmarks for various COPs. The framework, thereby, establishes a basis for benchmark generation. As well, by expressing severity and period of change in terms of significant changes, the framework aims to make better distinction to what constitute *different* environmental changes.

The general idea of the proposed framework is to start with an initial static benchmark problem S_0 taken from the literature or from available real life data. Then, changes are repeatedly introduced to the problem in order to generate a sequence of static problems. At any time, the introduced change should ensure that some exploitable similarity between any two succeeding instances is retained without rendering the change dynamically insignificant; in other words, neither the problem changes completely nor is the change trivial. The operation of the generator is divided into two stages: a *sequence generation stage* that creates a pool of k_{max} static problems and a *dynamism control stage* that selects problems from the pool to construct one dynamic problem with m_{max} instances, as illustrated in Figure 5.9.

In the sequence generation stage, a controlled amount of change is applied to an element of the optimization problem P_k to create the next problem P_{k+1} in the sequence. We refer to the controlled change as an *elementary step* δ_k. As there are three types of changes (dimensionality, non-dimensionality, and mapping-based), the elementary step can take one of the following three forms:

- A dimensionality step, i.e. the addition or the deletion of a single variable.
- A non-dimensionality step, which corresponds to a change in the value of the parameters or the coefficients of the problem. In this case, it should be dynamically significant. If it affects problem constraints, it should also ensure feasibility of the new problem instance; i.e., it may render some of the feasible solutions of the previous

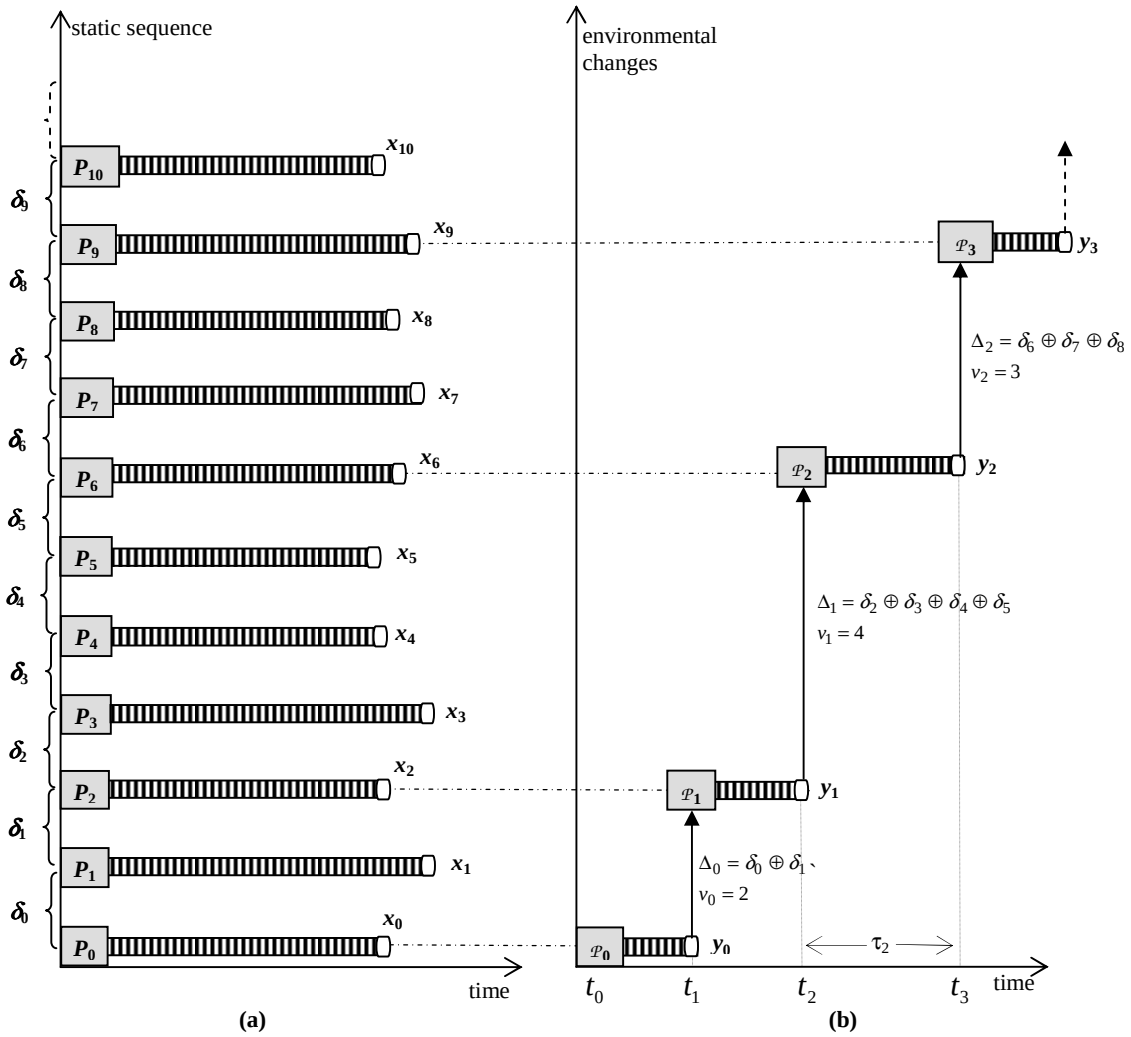


Fig. 5.9: Generalized benchmark generation. (a) Sequence generation stage (b) Dynamism control stage

instance infeasible but does make the new instance entirely infeasible.

- A mapping-based step, i.e. a single swap in the mapping function.

The above process of adding an elementary step can be written as follows:

$$P_{k+1} = P_k \oplus \delta_k, \quad k = 0, 1, \ldots, k_{max} - 1$$

Each newly created static problem is solved independently of the others. This stage finally ends with a sequence of static problems P_k and their corresponding optimal or near optimal solutions x_k. The sequence generation stage can be defined as follows:

$$\mathbf{S} = \{\mathbf{S}_k = (P_k, x_k),\ k = 0, 1, \ldots, k_{max}\}$$
where
$$P_k = P_0 \oplus \delta_0 \oplus \delta_1 \oplus \ldots \oplus \delta_{k-1}$$
$$x_k = argopt(P_k)$$

In the second stage, a complete dynamic problem $\mathbf{D}$ is created by selecting some of the static problems in the sequence $\mathbf{S}$ as instances of $\mathbf{D}$. The selection is done in such a way that the resultant dynamic problem has the required properties of dynamism. For example, the severity of the change can be increased by skipping some of the intermediate problems in the static sequence. Similarly the frequency of change can be specified by the number of evaluations (or generations) between successive instances.

To further elaborate on this stage, let us first define an *environmental shift* Δ_m as the change applied to the m^{th} instance of the dynamic problem to create the next instance. That is,

$$D_{m+1} = D_m \oplus \Delta_m$$

Then the severity of change ν_m can be expressed as the number of elementary steps comprising Δ_m, and the *period of change* τ_m can be defined as the duration (number of generations or evaluations between successive shifts) of the m^{th} instance.

Once the required severity ν_m and period τ_m are specified, the dynamic problem can be given as a sequence of problem instances D_m, points of time t_m and target solutions y_m as follows:

$$\mathbf{D} = \{\mathbf{D}_m = (D_m, t_m, y_m),\ m = 0, 1, \ldots, m_{max}\}$$

in which D_m and y_m are actually P_k and x_k in the sequence $\mathbf{S}$ respectively, where

$$k = \sum_{i=0}^{m-1} \nu_i$$

and instance m begins at time

$$t_m = \sum_{i=0}^{m-1} \tau_i$$

The performance of a dynamic solver on the constructed benchmark will be assessed by comparing solutions produced by the dynamic solver to the target solutions (y_m) produced by the benchmark generator.

Different test problems can be constructed under the generalized framework, by changing the static problem P_0, the elementary steps δ, the severity ν or period τ. As well, a second sequence of static problems can be added to the dynamic problem. The additional sequence is constructed by reversing the changes introduced to the first sequence. Then, by repeatedly adding and reversing changes, cycling environments can also be created.

We now give examples to illustrate how different environmental changes can be constructed in accordance with the generalized framework. These examples are based on the three modes of the dynamic TSP benchmark generator that Younes et al. [2003] introduced.

1. **Edge Change Mode**

   This mode reflects the introduction and removal of traffic jams. Distances between the cities are viewed as time periods or costs that may increase or decrease over time, hence a problem change can be introduced by changing the distance between cities. In this case, the elementary step of the change is the increase in the length of a single edge.

2. **Insert/Delete Mode**

In this mode, new assignments (cities) are added or deleted from the problem. The elementary step is the addition (or the deletion) of one city. As mentioned earlier, this mode might prove to be difficult to solve since it entails variable representation to match variable number of cities over time.

3. **Vertex Swap Mode**

   This mode, described earlier in Section 5.4, involves interchanging the locations of two randomly selected cities. The length of the optimal tour remains the same but the tour itself will be different. The elementary step is an interchange of the labels of two cities.

The knapsack problem can be easily fit in this framework. Dimensionality changes takes place as the addition or deletion of an object from the problem; the corresponding elementary step is simply the addition or deletion of a single object. For non-dimensionality changes, the elementary step can be any dynamically significant change in the value or weight of a single object, or in the capacity of the knapsack. If the mapping swap operator is used, the corresponding elementary step is an interchange of two objects (one pair).

5.6 Summary

General purpose benchmark generators, though important in continuous optimization, are of little use to COPs. The concept of *moving peaks*, which is the basis of these generators, is not applicable to COPs, where even the word *peaks* is ambiguous in a discrete problem.

Combinatorial benchmarks are best described as time sequences of static problems strung together, where each static problem is explicitly solved one way or another. However, existing benchmarks tend to be too problem specific to be of general interest. Furthermore, the construction of the dynamic variant is computationally expensive especially if the

base static problem is NP-hard. Fortunately, the mapping-based scheme presented in this chapter considerably reduces computation costs incurring during benchmark construction.

The chapter also presented a generalized framework that aims to establish a basis for benchmark generation for COPs. Recast in the generalized framework, application specific benchmarks can be of interest even to those researchers who find problem specific benchmarks too limiting.

Chapter 6

Merits of Adaptation

Since a genetic algorithm is an intrinsically dynamic and adaptive process, the use of constant parameters is thus in contrast to the general evolutionary sprit.

Gen and Cheng [1999]

6.1 Introduction

All metaheuristics have to deal with two interrelated issues that significantly influence the search process and its outcomes: the balance between exploration and exploitation, and the tuning of the algorithm input parameters. The way these issues relate to each other is not completely understood—at least it is not straightforward. In dynamic optimization, environmental changes add a new dimension to these issues, since the algorithm has to react in a proper way to each change. This chapter examines these issues and investigates a simple way to change genetic parameters in response to environmental changes.

6.2 Variable genetic parameters

The use of variable parameters is promoted by two beliefs: one belief is that genetic parameters influence quality of final solutions in static environments; the other is that genetic parameters can be made to delay undesirable population convergence in both static and dynamic environments.

6.2.1 Genetic parameters and solution quality

It is well known that changing parameters of an EA can change the behavior of the algorithm and, consequently, the quality of the final solution. Figure 6.1 and Figure 6.2 illustrate how rates of mutation and crossover affect the values of the the best solution in the population and mean of the population for a 100-city TSP problem. Of course, changing one parameter at a time is not a satisfactory approach either. The point is that different genetic parameter settings can greatly influence the search process. For this reason, developers spend substantial time searching for the best set of parameters for the problem instance under consideration.

However, recent research suggest that no set of parameters remains the best throughout

the run. That is, a certain set of parameters that proves to be good at a given generation may become less effective during subsequent generations. The fact that EAs sample the search space and that this sampling incorporates random components explains why the search process should be regarded a dynamic process that changes from one generation to another—even when the environment is static. Consequently, variable parameters, rather than fixed parameters, are the appropriate choice for dynamic problems.

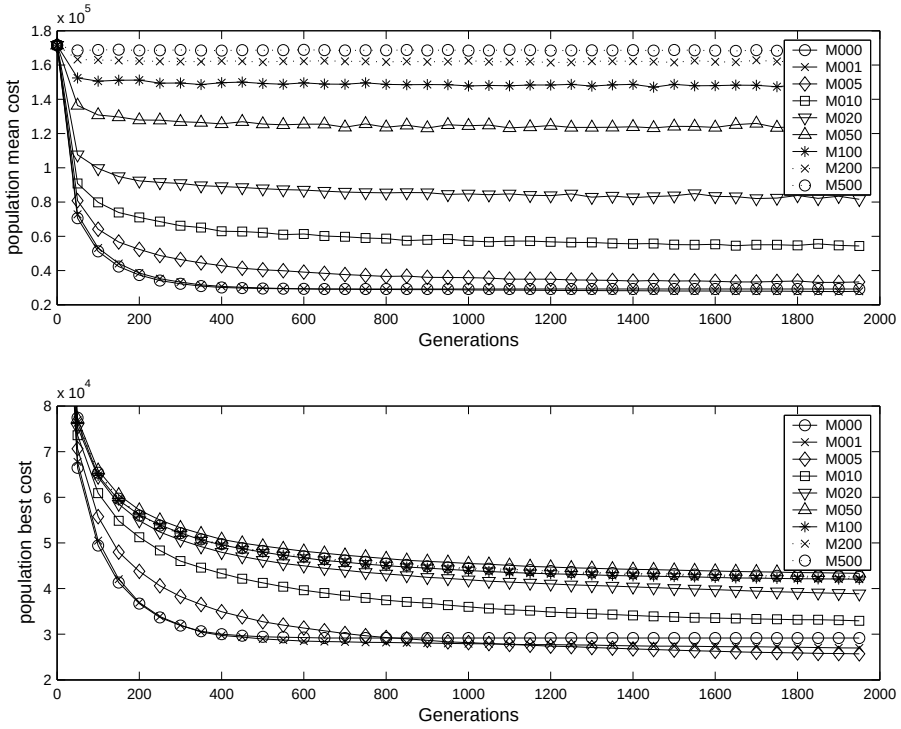


Fig. 6.1: Effect of mutation rate on solution (K100 problem). Values in the legend give mutation rate per thousand, e.g., M005 means mutation rate of 0.005

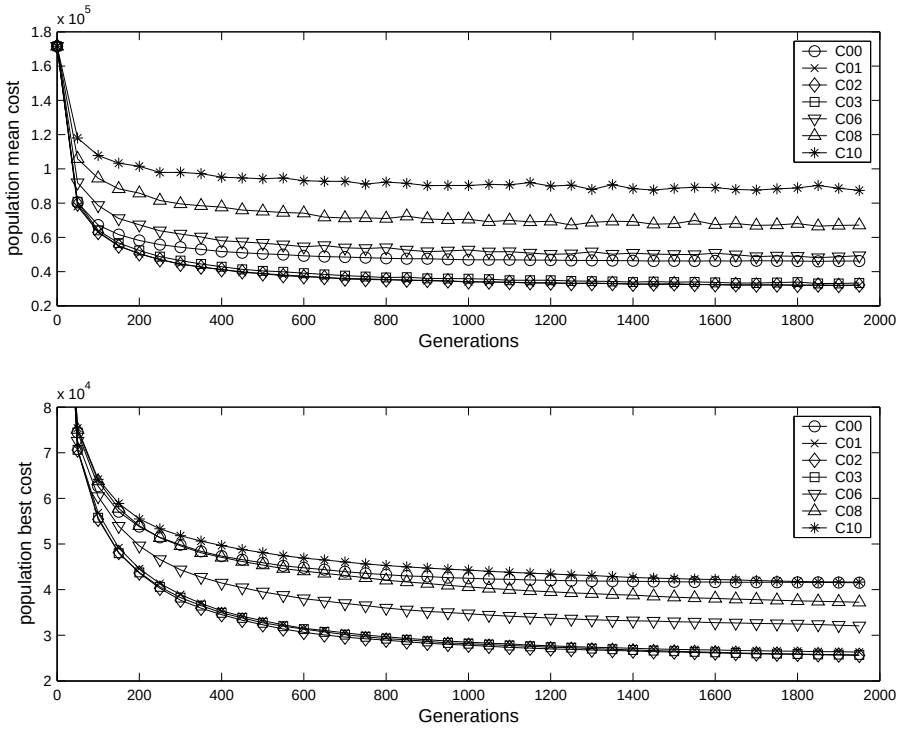


Fig. 6.2: Effect of crossover rate on solution (K100 problem). Values in the legend give crossover rate per ten; e.g., C06 means crossover rate of 0.6

6.2.2 Undesirable population convergence

Traditional GA operators work in a manner that reduces diversity of the search after a promising region is discovered, to focus the search on that region. In an ideal search, solution quality enhances with time while population diversity gradually decreases. The double plots in Figure 6.3 show the evolution of solution quality and population diversity for different values of genetic parameters. It can be seen that both measures go hand-in-hand with time in almost all the subplots (except when the mutation rate is very large). Thus, a reduction of population diversity is an indication of the EA achieving its goal of finding the optimal solution. Some researchers even use diversity reduction as a criterion

for terminating the search process.

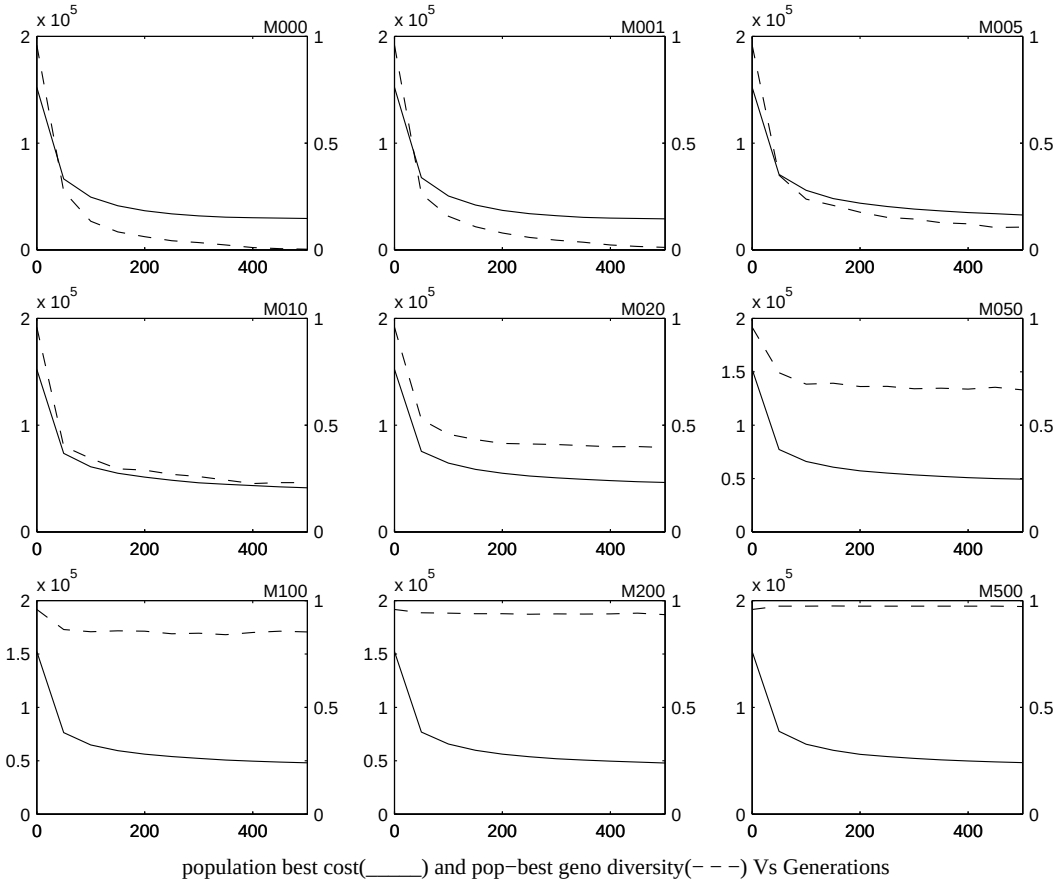


Fig. 6.3: Evolution of population best and diversity (k100).

However, the actual search is almost always non-ideal, and convergence takes place around suboptimal solutions. This phenomenon makes promoting population diversity a central issue both in static and in dynamic optimization. In static optimization, loss of diversity is often blamed for what is known as *premature convergence*, which describes the situation when the search terminates at suboptimal solutions. In dynamic optimization, loss of diversity is blamed for the algorithm's inability to further track the shifting optima.

The latter situation will be termed *obsolete convergence* in this study, since it is outdated by subsequent changes in environment .

Definition 6.1 (obsolete convergence) *A state of the search process in which the population has converged around an optimal or suboptimal solution to an instance that is replaced by a newer one due to some change in the environment.*

It is important to note that real-world problems seldom change completely at once[1]; hence, some of the information gathered from the past are re-usable in present and future instances. Therefore, for a GA to be efficient in a dynamic environment, it should be able to persevere after obsolete convergence and at the same time overcome premature convergence. The DS should thus be able to transfer as much knowledge from one instance in the dynamic problem to the next instance without biasing the search excessively around old solutions. This can be achieved by adaptively balancing the GAs ability to exploit old information with the much needed ability to explore the changing search space. This balance can be manipulated by controlling the genetic operators that have some influence on the population diversity; e.g, the selection and mutation operators. Hence, this chapter investigates a model that changes genetic parameters specifically to tackle dynamic problems. The model reacts to environmental changes by biasing the search process towards exploration. Once the environment becomes static again, the model reduces the explorative forces gradually with time.

6.3 Linear model

The idea of using a linear model (LM) to change genetic parameters for dynamic environments is intuited from the experiments of Grefenstette [1999] on genetically controlled

[1]In real life, events happen only to part of the problem. For example, vehicles do not break down all at once and weather changes affect only parts of roads.

mutation, where mutation rate is controlled by the algorithm itself rather than by the user. He showed that the level of hypermutation increases significantly following an environmental change then reduces gradually with time, as shown in Figure 6.4. In this way, the population is enabled to re-converge around a promising region of the search space after every environmental change. The main problem, however, with genetic control is that the algorithm might require an excessively large number of generations to evolve to its final values. Although slow rates of adaptation might be acceptable in some static applications, they are certainly unacceptable in dynamic applications. Hence, the proposed LM model explicitly specify the rate of change of the mutation rate instead of leaving it up to the genetic process. Furthermore, the LM model is not limited to controlling the mutation rate only; it can be used to control other genetic parameters too.

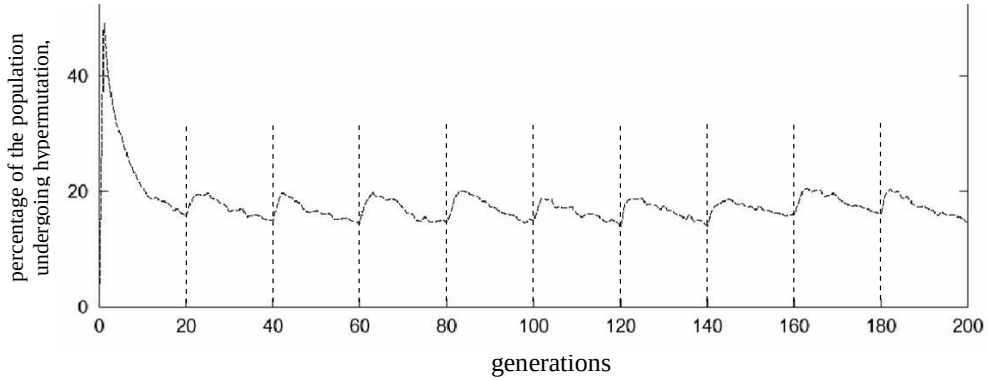


Fig. 6.4: Genetically controlled hypermutation in a problem that changes every 20 generations. Mutation rate rapidly increases after environmental changes (vertical dotted lines) and gradually decreases during static phases of the problem [Grefenstette 1999]

6.3.1 Adaptive mutation

The LM model aims to diversify the search whenever an environmental change is detected, and to progressively reduce diversification during quiescent phases. A simple way to achieve

this is by linearly changing the mutation rate repeatedly. When the environment changes (say at time $t = t_m$), a cycle begins and the mutation rate $\mu(t)$ is set to an upper limit $\overline{\mu}$. Subsequently, it is reduced with time until, after a period ρ, it reaches to a base value $\underline{\mu}$. The current cycle terminates at the next environmental change (at time $t = t_{m+1}$), and a new cycle begins, as depicted in Figure 6.5.

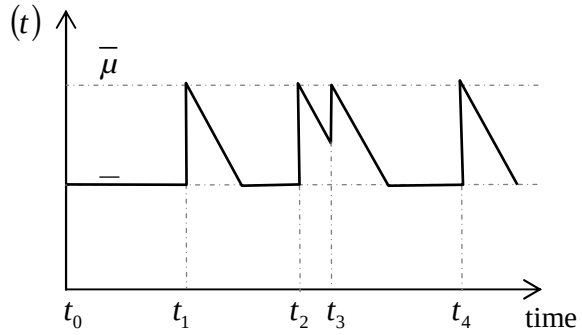


Fig. 6.5: Variable mutation rate

The following formula gives the variation of mutation rate $\mu(t)$ in the cycle between two consecutive environmental changes; i.e., when $t_m \leq t < t_{m+1}$.

$$\mu(t) = \begin{cases} \overline{\mu} - \dfrac{\overline{\mu} - \underline{\mu}}{\rho}(t - t_m), & t_m \leq t < t_m + \rho \\ \underline{\mu}, & t_m + \rho \leq t < t_{m+1} \end{cases} \tag{6.1}$$

6.3.2 Adaptive crossover

In addition to changing the mutation rate, the LM changes the rate of crossover and the probability of selection. Crossover has effects similar to those of mutation on diversity, in that higher rates of crossover tend to increase population diversity as shown Figure 6.6. Thus, the LM changes the rate of crossover linearly in a similar manner to that of the mutation. The following formula gives the variation of crossover rate $\chi(t)$ in the cycle

between two consecutive environmental changes; i.e., when $t_m \leq t < t_{m+1}$.

$$\chi(t) = \begin{cases} \overline{\chi} - \dfrac{\overline{\chi} - \underline{\chi}}{\rho}(t - t_m), & t_m \leq t < t_m + \rho \\ \underline{\chi}, & t_m + \rho \leq t < t_{m+1} \end{cases} \tag{6.2}$$

where $\overline{\chi}$ and $\underline{\chi}$ respectively are upper and lower limits of crossover rate.

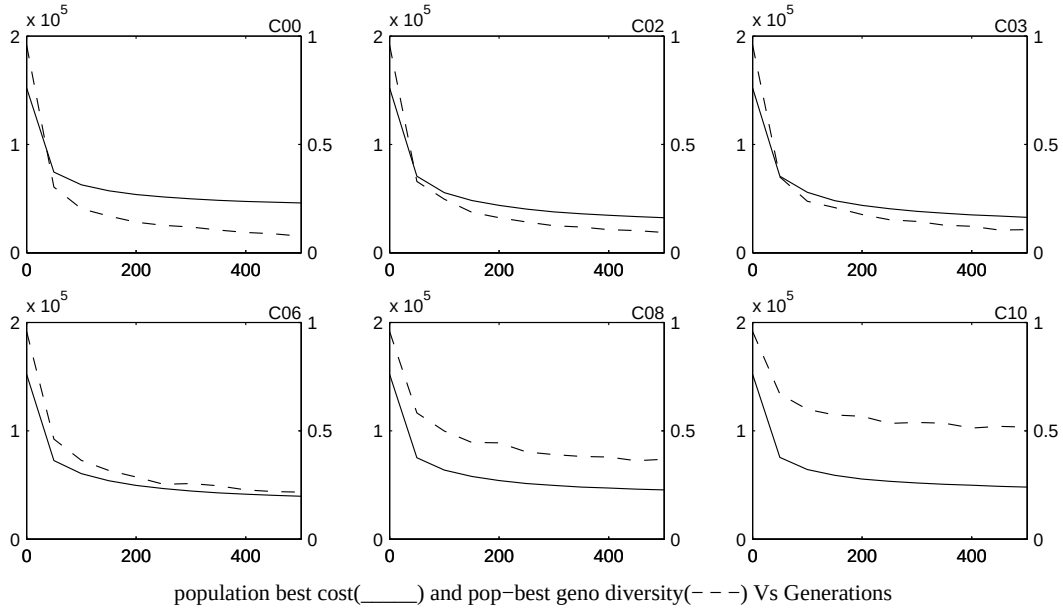


Fig. 6.6: Comparing the evolution of population best and diversity under different rates of crossover. On each subplot crossover rate is given in tenths, e.g., C06 means crossover rate =0.6.

6.3.3 Adaptive selection

In terms of selection, tournament selection is one of the widely used techniques. It works in a manner similar to ranking selection (which avoids the pitfalls of proportionate selection methods, such as roulette wheel) and at the same time is easier to implement and less

time consuming than ranking methods which require sorting individuals according to their fitness. A simple tournament scheme selects one individual from a group of individuals (typically two) based on its relative fitness. This scheme is often modified by injecting it with some degree of randomness that would lead to the selection of the less-fit individual from time-to-time (instead of selecting the fitter individual all the time). In the modified tournament selection scheme, the fitter individual is selected at a fixed probability s in the range between 0.5 and 1.0. Thus, selection probability is an additional input parameter that affects the outcome of the search process and hence has to be tuned with the rest of the algorithm parameters. Figure 6.7 shows the influence of probability of selection on the solution evolution through time on a typical COP. However, similar to other genetic parameters, selection probability can not be directly related to solution quality.

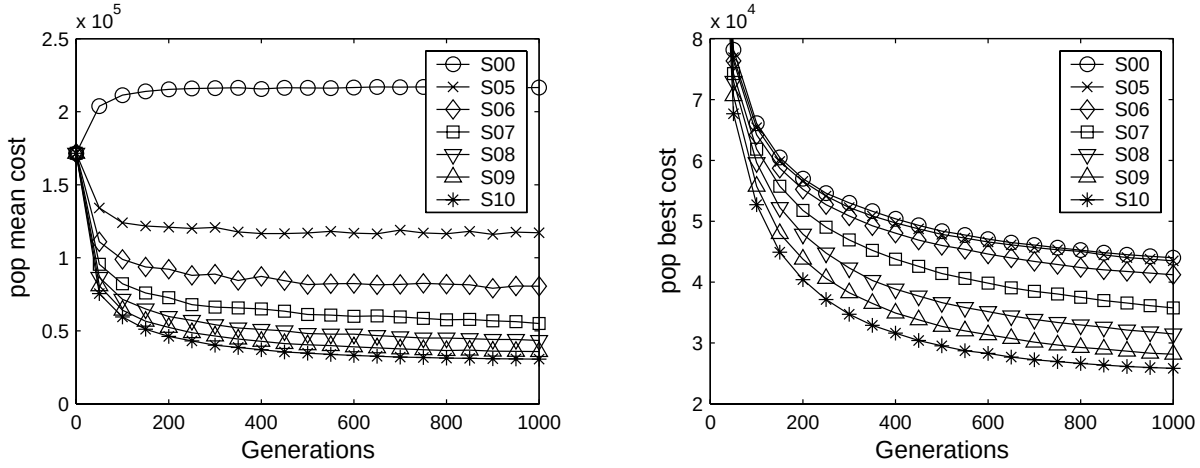


Fig. 6.7: Effect of selection probability on solution (K100 problem). Values in the legend give selection probability per ten, e.g., S06 means the probability is 0.6

Still, we can have more direct relations between selection probability and population diversity. The correlation between selection probability and population diversity is evident in Figure 6.8, where increasing the probability of selection from one subplot to another

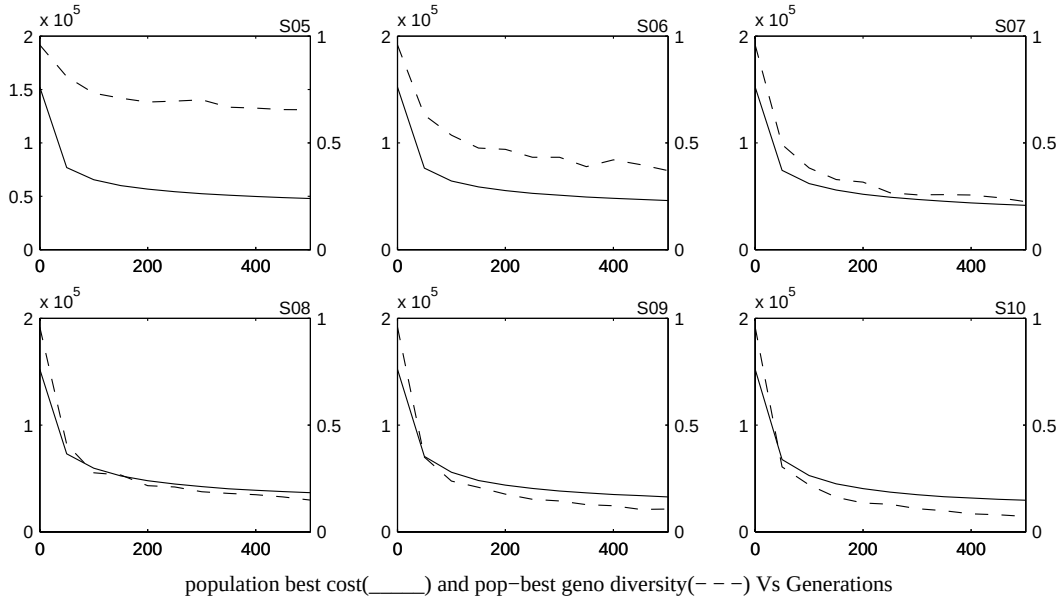


Fig. 6.8: Comparing the evolution of population best and diversity under different values of selection probability. Selection probability is given in tenths, e.g., S08 means selection probability =0.8.

reduces diversity. Therefore, selection pressure and consequently the rate of population convergence can be controlled by changing selection probability. Thus, the LM changes the probability of selection linearly in a manner similar to that of of mutation and crossover rates but in an opposite sense. The following formula gives the variation of selection probability rate $s(t)$ in the cycle between two consecutive environmental changes; i.e., when $t_m \leq t < t_{m+1}$.

$$s(t) = \begin{cases} \underline{s} + \dfrac{\overline{s} - \underline{s}}{\rho}(t - t_m), & t_m \leq t < t_m + \rho \\ \overline{s}, & t_m + \rho \leq t < t_{m+1} \end{cases} \tag{6.3}$$

where $\overline{s}$ and $\underline{s}$ respectively are upper and lower limits of selection probability.

With this scheme, the varying selection probability will have a typical shape like that

of Figure 6.9.

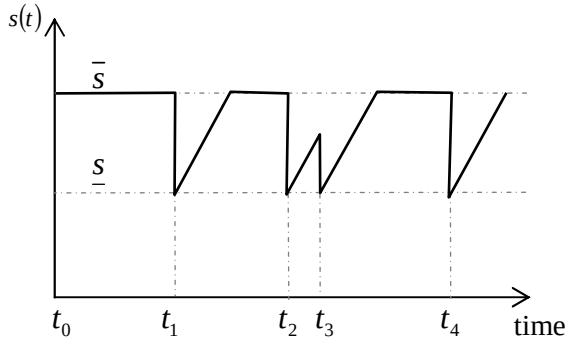


Fig. 6.9: Variable selection probability

6.3.4 Algorithm structure

The adaptive operators of the previous sections are combined in vector form by denoting the the controlled genetic parameters by $\mathbf{P}(t)$, their respective explorative values by $\mathbf{P}_r$, and their respective exploitative values by $\mathbf{P}_t$. Thus the complete linear model becomes

$$\mathbf{P}(t) = \begin{cases} \mathbf{P}_r - \dfrac{t - t_m}{\rho}(\mathbf{P}_r - \mathbf{P}_t), & t_m \leq t < t_m + \rho \\ \mathbf{P}_t, & t_m + \rho \leq t < t_{m+1} \end{cases} \tag{6.4}$$

where $\mathbf{P}(t) = [\ \mu(t) \quad \chi(t) \quad s(t)\ ]^T$, $\mathbf{P}_r = [\ \overline{\mu} \quad \overline{\chi} \quad \underline{s}\ ]^T$, and $\mathbf{P}_t = [\ \underline{\mu} \quad \underline{\chi} \quad \overline{s}\ ]^T$.

This model is used to convert a standard genetic algorithm into a dynamic TSP solver as shown in Figure 6.10. In this algorithm, chromosome representation is a straight forward path representation, where values of the genes are the city numbers, and the relative position of the genes represent city order in the tour.

The algorithm is generational in that the whole population is replaced by the new offspring at every generation. A stochastic tournament selection is used to select surviving individuals at every generation. The pseudo code for selection is given in Figure 6.11.

```
Procedure LM
    i = 0;   // initiate instances counter
    g = 0;   // initiate generations counter
    Generate(pop);   // generate initial population
    Evaluate( pop );
    repeat
        while quiescent environment
            g = g+1;
            TournamentSelect( pop );
            Cross( pop );
            Mutate( pop );
            best_of_generation[g] = Evaluate( pop );
            params = LinearlyAdaptParam( params , g );   // apply LM, Eqn 6.4
        endwhile
        i = i+1;
        AdaptPopulation( pop , strategy );   // apply LM, case t = t_m in Eqn 6.4
    until terminating condition
```

Fig. 6.10: Pseudo code for a linear dynamic solver

Edge crossover [Whitley et al. 1991] is employed to recombine solutions in the population. This operator has the advantage of preserving most edges of the parent solutions without producing infeasible child solutions.

The mutation operator is a pair-wise node swap. It introduces changes in the chromosome by swapping the positions of two randomly selected nodes. This operator produces little change in the individual, since no more than four edges are replaced in each mutation step. More drastic changes can be applied by simply increasing the rate of mutation.

```
Procedure TournamentSelect( pop )
    a = 0;    // initiate counter of individuals
    while a < pop·size
        b = rand( 0 , pop_size-1 )
        indA = pop·individual[a];
        indB = pop·individual[b];
        if rand(0,1) ≤ selection_probability;
            pop·individual[a] = Best( indA , indB );
        else
            pop·individual[a] = Worst( indA , indB );
        endif
        a = a+1;
    endwhile
```

Fig. 6.11: Pseudo code for stochastic tournament selection

6.4 Experimentation

This section describes experiments carried out mainly to test the linear model against the restart strategy.

6.4.1 Test problems

The dynamic test problems are based on a 100-city problem, kroA100, borrowed from the TSP library Reinelt [1991]. Dynamic changes are introduced in three modes that reflect the methods described in the generalized benchmark generator of Chapter 5: an edge change mode (ECM), an insert/delete mode (IDM) and a vertex swap mode (VSM). Problem dynamics are basically controlled by two parameters: frequency and severity. The frequency of change is determined by the number of generations between succeeding environmental changes, and the severity of change is determined by the number of elementary steps applied at every environmental change. An elementary step is the smallest possible change

in the problem that causes the new instance to have a different optimal solution from the previous one.

In ECM, the problem is changed by increasing the length of an edge randomly selected from the best found tour, or decreasing the length of an edge randomly selected but not from the best found tour. This scheme ensures that environmental changes affect optimal solutions. The elementary step of the change is the change in the cost of a single edge.

In IDM, environmental changes are imposed by adding new cities to the problem or by removing some of the existing cites. The elementary step of the change in this mode is the addition (or the deletion) of a single city. This mode might prove to be the most difficult since it entails variable representation to reflect the changing number of cities.

In VSM, the labels of two randomly selected vertices (cities) are interchanged in the mapping function that maps the chromosome into solutions. VSM offers an efficient way to create dynamic problems with known optima without the need of re-optimization.

In the current experimentation, each benchmark problem includes 200 successive changes to the base problem, that is, there is a sequence of 200 static problems for each of the three modes of environmental shifts. Each sequence of static problems is translated into nine dynamic test problems, by combining three degrees of severity (1, 10, 100 steps per shift) and three periods of change (10,100, 1000 generations between shifts). In short, there are twenty-seven dynamic test problems each constructed from 200 static instances.

6.4.2 Competing strategies

The dynamic test problems are used to compare the performance of the LM model against four other models: *FM, RM, RIM10 and RIM20*.

FM The ignore strategy is modelled by a fixed model (*FM*). This model is basically a traditional GA with fixed operator rates. It ignores environmental changes. That is, it does not apply any specific measures to tackle dynamism in the problem and

hence is expected to perform poorly. Still this strategy might produce good solutions if changes in the dynamic problem are insignificant.

RM The restart strategy is the most straightforward strategy to handle dynamic problems. This strategy depends totally on exploration to find new solutions and does not make use of any past knowledge. The restart model (RM) randomly re-generates the population whenever the problem changes. That is, the RM model regards each change in the environment as a new problem to be solved independently of the previous instances.

RIM10 The random immigrants strategy can be viewed as a partial restart, since only a fraction of the population is re-initialized at every environmental change. In random immigrant model (RIM10), only 10% of the population is replaced with random immigrants (new individuals).

RIM20 This model also uses random immigrants but replaces 20% of the population at each environmental change.

To make comparisons fair, all models use a population size of 50, a crossover rate of 0.9, a selection probability of 1, and a mutation rate of 0.025. For the LM model the lower limits of mutation rate, crossover rate, and selection probability are 0.025, 0.9, and 0.9 respectively, and their respective upper limits 0.05, 1.0, and 1.0.

6.4.3 Results

Our primary goal is to compare the linear model with the restart model.

We investigate the behavior of an individual algorithm by plotting the evolution of the best of generation through time, together with two additional curves that serve as lower and upper bounds on the algorithm's performance. The lower bound is the goal (values given by the BG), and the upper bound is taken as the results from the RM model. Figures 6.12,

6.13, and 6.14 show detailed tracking of optima through time by the adaptive mutation model. Each figure considers both rapidly changing environments (every 10 generations) and slowly changing environments (every 1000 generations). The environmental changes are ECM type in Figure 6.12, IDM type in Figure 6.13, and VSM type in Figure 6.14.

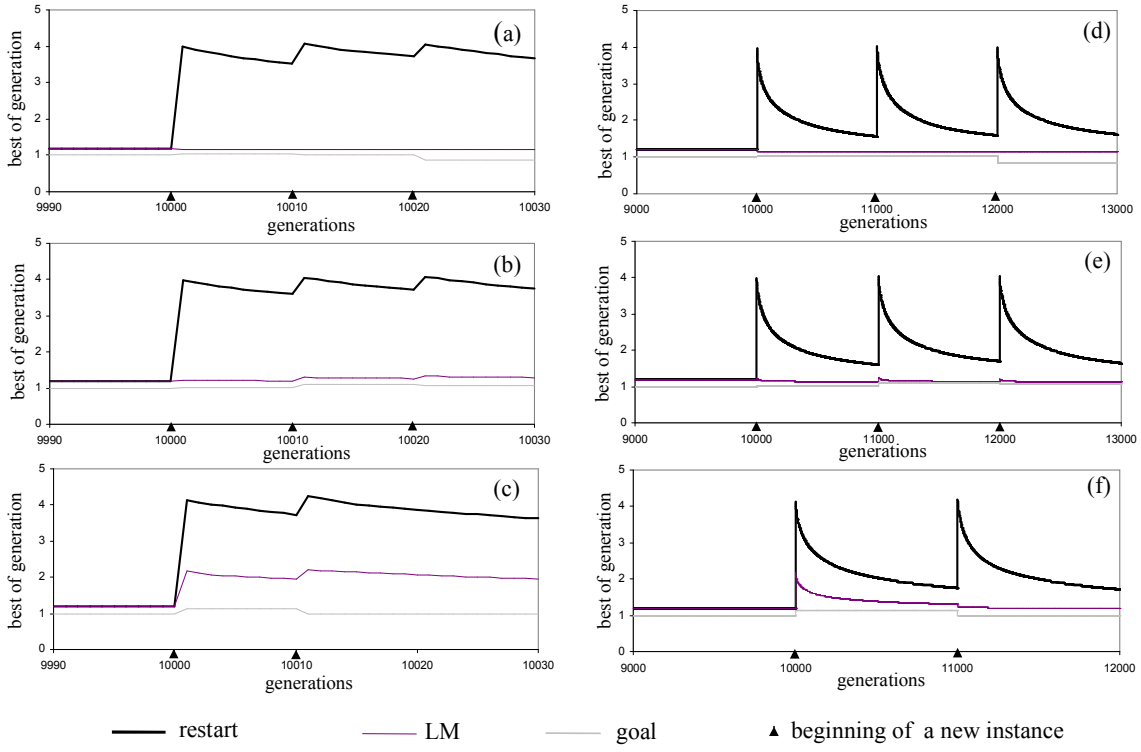


Fig. 6.12: Evolution of best of generation on the k100 problem in the ECM mode. In subplots (a), (b), and (c) the problem changes every 10 generations with severities of 1, 10, and 100 steps per shift respectively. In subplots (d), (e), and (f) the problem changes every 1000 generations with severities of 1, 10, and 100 steps per shift respectively.

It can be seen that the accuracy of the tracking of the optima depends both on the speed and the severity of environmental changes. In rapidly changing environments (left hand subplots in the three figures), the benefits of adapting old solutions are very clear when contrasted with the restart strategy (the cost of a restart solution can be as much as four

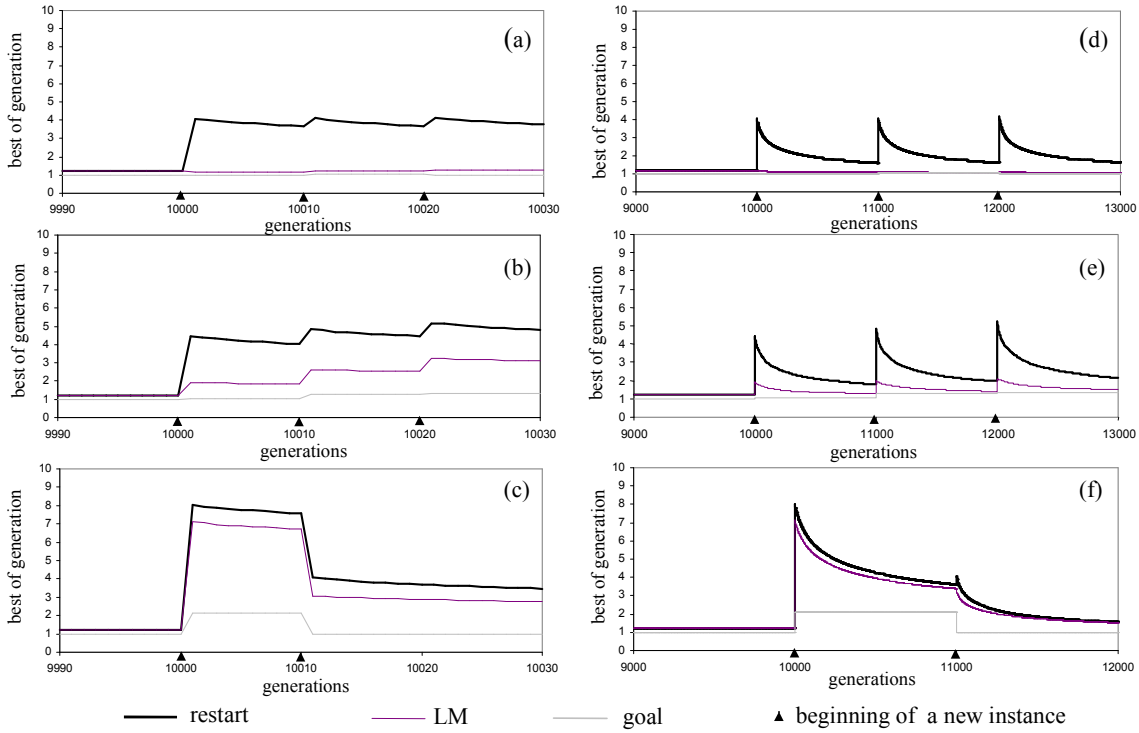


Fig. 6.13: Evolution of best of generation on the k100 problem in the ADM mode. In subplots (a), (b), and (c) the problem changes every 10 generations with severities of 1, 10, and 100 steps per shift respectively. In subplots (d), (e), and (f) the problem changes every 1000 generations with severities of 1, 10, and 100 steps per shift respectively.

times that of the adaptation strategies when the changes are small and the computation time is limited). Since the restart strategy re-initializes the population completely after each environmental change, the new population tends to be scattered over the search space and is hence very likely to be away distant from optimal solutions. This explains why the figures indicate a sudden deterioration in performance of the RM model after each change. Unlike the RM model, the linear model does not disturb the entire population but rather mutates a small part of it. Hence, even after a change in the environment, the population retains some individuals that are very likely to be in the vicinity of good quality solutions

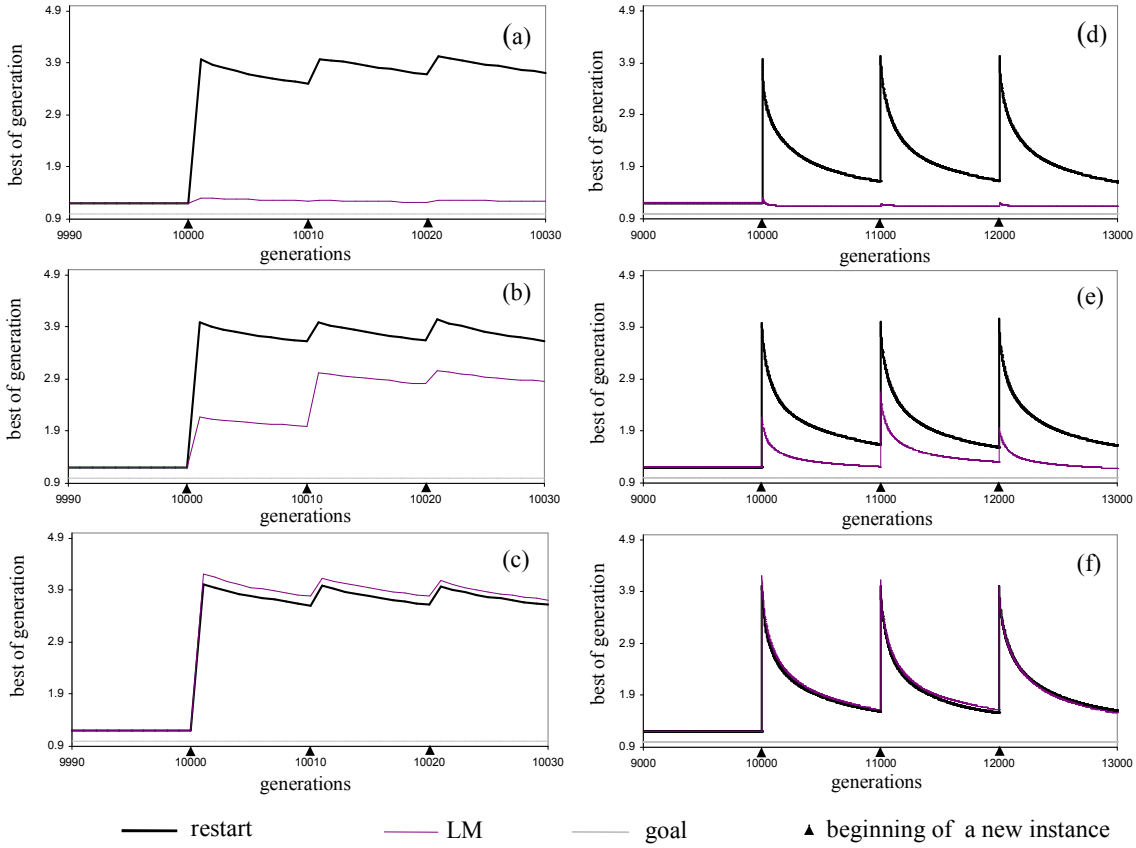


Fig. 6.14: Evolution of best of generation on the k100 problem in the VSM mode. In subplots (a), (b), and (c) the problem changes every 10 generations with severities of 1, 10, and 100 steps per shift respectively. In subplots (d), (e), and (f) the problem changes every 1000 generations with severities of 1, 10, and 100 steps per shift respectively.

(unless the environmental changes are very drastic).

In the slowly changing environments (right hand subplots in the three figures), the algorithms are given practically unlimited computation time, allowing even the relatively slow restart strategy to display a better performance. Still it is inferior to the LM model, where the difference between both strategies is the greatest when the severity of change is small.

As mentioned in Chapter 4, it is not practical to use curves of evolution to compare several algorithm. Thus, the five competing models are compared using mean best of generation. Selected results are shown in Figure 6.15 for different rates of base mutation (i.e., upper limit for LM and conventional rate for other models). As well, different ranges of severity of environmental changes and the three modes of changes (ECM, IDM, and VSM) are investigated. Each point in these plots is the average of ten independent runs using different random initial populations. In every run, the environment is kept quiescent for the first 10000 generations, then allowed to change according to the specified severity and period of change. The prolonged initial static phase gives the GA sufficient time to reach initial convergence and thus make later performance less dependent on the initial population. This scheme is essential since the major challenge to the dynamic solver is to be able to explore the new search space after converging around some high quality solution under the preceding environment. Since the solution values are likely to vary with time, mean best of generation (MBG) values are reported as ratios of the solution to the base problem.

It can be noted that adaptation strategies (LM, RIM10 and RIM20) tend to be closest to the goal line, which is formed from the best solutions previously computed by the BG. As well, it seems to be a good idea to keep rates of base mutation in the vicinity of 0.1. The RM model shows the poorest performance when the problems change moderately. Indeed, RM results are so large that they do not show in some subplots of the figure. Restart does comparatively well only when the changes are relatively large, as in Figure 6.15(f) and Figure 6.15(i).

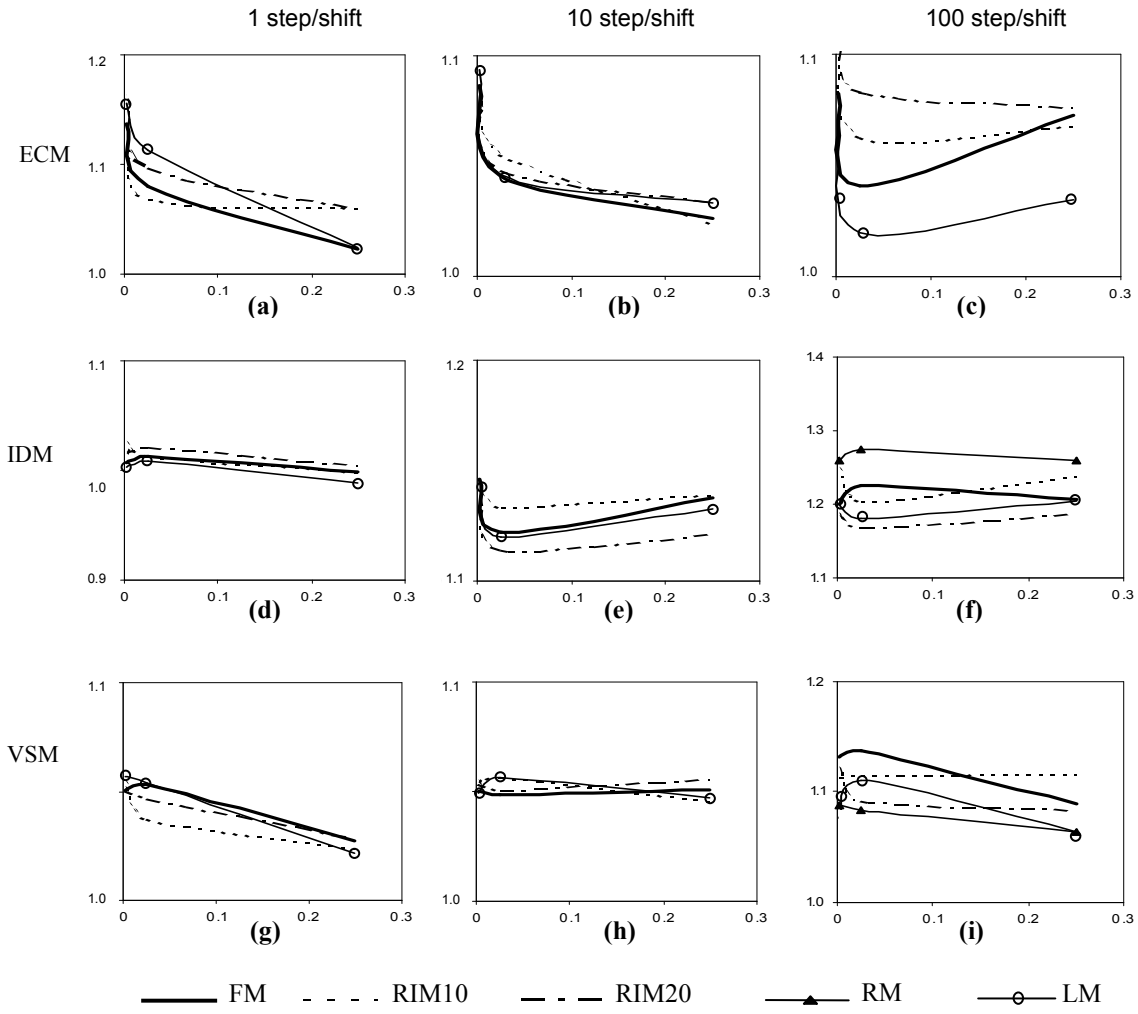


Fig. 6.15: Strategy comparison. Mean best of generation (vertical axis) is plotted against base mutation rate (horizontal axis) using mean best of generation. Note that the results from applying the restart strategy are of low quality (too large to appear with other models in most figures).

6.5 Summary

This chapter presented an EA model that changes genetic parameters to increase population diversity at environmental changes and gradually changes the parameters towards low population diversity during quiescent phases of the environment.

In order to enable fair comparisons, a dynamic solver was developed to track the moving optima in dynamic problems. It can switch between different models that apply strategies of tackling dynamic problems.

The mapping swap benchmarking scheme can be used to replace other modes. Since the behavior of the dynamic solver was found to be similar in the three types of benchmarking.

The restart strategy produced solutions of low quality suggesting that it is very important not to discard knowledge from past solutions.

The dynamics of the problem greatly influence the effectiveness of an algorithm. The results of the previous experiments simply state that when the environmental changes are slight, the problem can be treated as static (a fixed base rate of mutation address such problems). At the other extreme, the restart strategy can produce the best results if the problem changes completely. Between these extremes lie the majority of real world cases that benefit most from adaptive strategies. Hence, adaptation is viewed as a tradeoff between the restart strategy and the ignore strategy, as illustrated in Figure 6.16. In other words, the level of diversity imparted to the dynamic problem should be proportional to the severity of environmental change.

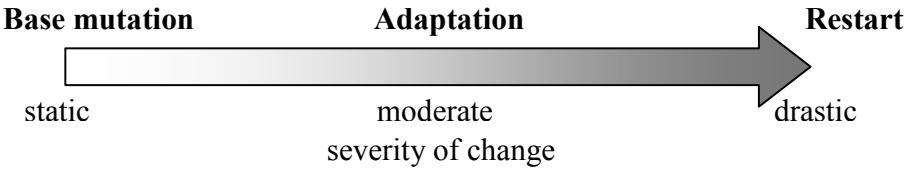


Fig. 6.16: Tradeoff between restart and ignore strategies

However, the developed model suffers a significant disadvantage. It aggravates the difficulty of parameter tuning with the need of finding two values for each parameter (upper and lower limits). Furthermore, one would naturally wonder: if a genetic parameter produces better results when it varies linearly with time than when it is fixed, then it might produce the best results when it varies in some nonlinear fashion. However, it would be more perplexing, more time consuming, and generally more difficult to try to tune a nonlinear function relating parameter to time.

Chapter 7

Adaptive Dynamic Solvers

Although all EAs possess the necessary operators for intensification and diversification, many EA implementations lack a mechanism to control the balance between these two factors.

Sörensen and Sevaux [2004]

7.1 Introduction

In the previous chapter, it was asserted that deciding on an appropriate set of parameter values for an evolutionary algorithm is a non-trivial task; what might seem to be an optimal set of at one generation may become less good at another generation. Even if an acceptable set was discovered (often after considerable effort of tuning) for a given problem instance, this set is not guaranteed to work well with other problem instances. Thus, schemes that seek to control genetic parameters in response to some feature of the search can be more effective and certainly less demanding in terms of tuning efforts, especially in dynamic problems where time for tuning is often limited.

The issue of parameter tuning is closely related to that of balancing exploration and exploitation during the search process. A direct link to both issues is population diversity. At one end of the link, increasing diversity drives the search towards exploration while decreasing diversity focuses the search on a specific promising region. At the other end of the link, genetic operators tend to either increase or decrease population diversity. Thus genetic operators can be used to manipulate population diversity and consequently the search status between exploration and exploitation.

The use of diversity to control evolutionary algorithms on dynamic COPs is investigated in this chapter. Different measures of diversity are examined and compared in Section 7.2. An *adaptive diversity model* in which genetic parameters vary in response to measured diversity is introduced in Section 7.3. An *adaptive islands model* that extends the idea of measuring and controlling diversity to multiple populations is presented in Section 7.4. Both models are hybridized with local search into an *adaptive hybridized diversity model* and an *adaptive hybridized island model* in Section 7.5.

7.2 Investigating diversity measures

The term *population diversity* or simply *diversity* appears frequently in the literature of evolutionary algorithms without definition [Burke et al. 2004]. Generally, diversity is taken as a characteristic of the population that reflects how different the individuals in the population are. A common definition for population diversity is given by Barker and Martin [2000] as "the sum over all pairwise distances between individuals in the population". Such a definition suggests that a diversity measure cannot be resolved unless the underlying distance measure is identified.

7.2.1 Distance measures

A distance measure can be thought of as a function that associates a non-negative real value with a pair of solutions such that similarity between the solutions increases as the their distance decreases. There are certain properties that are expected to exist in a proper distance measure $dist(x, y)$ between a pair of solutions x and y [Kapur and Kesavan 1992].

- Distance is non-negative.

$$dist(x, y)) \geq 0 \tag{7.1}$$

- Distance is symmetric.

$$dist(x, y) = dist(y, x) \tag{7.2}$$

- Distance is zero between identical solutions.

$$dist(x, y) = 0 \quad \text{iff } x = y \tag{7.3}$$

- The distance between any two solutions can not exceed the sum of the distances between each of them and a third solution.

$$dist(x, y) \leq dist(x, v) + dist(v, y) \tag{7.4}$$

To measure population diversity, three types of distance measures come into consideration, namely genotypic, phenotypic, and algorithmic measures.

Genotypic distance

Genotypic distance measures are based on the difference in the genome of the individuals under consideration. The most commonly used genotypic distance is the Hamming distance, $hamm\_dist(u, v)$ which compares corresponding genes in two strings u, v.

$$hamm\_dist(u, v) = \sum_{i=1}^{l} abs(\, sgn(u[i] - v[i])\,) \tag{7.5}$$

where $u[i]$ is the i^{th} gene in string u and $v[i]$ is the i^{th} gene in string v. However, the Hamming distance is unsuitable for permutation-based representations where the relative positions are important. For example in TSP and VRP, the *edge distance* is commonly used to indicate the number of uncommon edges—not genes—in two tour strings. Denoting the set of edges in a string u by $arcs(u)$, edge distance can be given by the following formula:

$$edge\_dist(u, v) = |arcs(u) \backslash arcs(v)| \tag{7.6}$$

which translates as the number of edges in $arcs(u)$ but not in $arcs(v)$.

Phenotypic distance

Phenotypic distance measures use some features of the solution to measure differences between solutions. They are usually based on values of a fitness or objective function. A commonly used fitness-based measure is given by the following formula.

$$fitness\_dist(u, v) = abs(\, sgn(fitness(u) - fitness(v)\,)$$

Algorithmic distance

Algorithmic distance measures give distance between solutions in terms of the number of moves a certain algorithmic operator uses to skip from one solution to another. Given a specific algorithmic operator, this measure most faithfully reflects the "efforts" or costs needed to traverse between solutions. However, this kind of measure is not useful in performing inter-algorithm comparisons and in addressing benchmarking issues, since both its meaning and its value are algorithm dependent. Furthermore, this measure is not easy to compute and can be expensive, except for simple algorithms that utilize single operators to move between solutions. For these reasons, measures of algorithmic distance are not considered in this thesis and discussion hereafter is limited to measures of genotypic and phenotypic distance.

7.2.2 Genotypic or phenotypic diversity?

Depending on the underlying distance measure, there can be a genotypic diversity measure ($\mathcal{GD}$) and a phenotypic diversity measure ($\mathcal{PD}$). The measures used in this thesis are normalized so that they fall in the range between zero and one. A zero diversity means all individuals in the population are similar (ideally identical) to each other, and a diversity of one means that they are very (ideally completely) different from each other. For example, normalized genotypic diversity based on the edge distance can be given as:

$$\mathcal{GD} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{edge\_dist(v_i, v_j)}{(l-1)(n-1)n/2} \tag{7.7}$$

where n is the population size, l is the chromosome length, and v_i and v_j are the i^{th} and the j^{th} individuals in the population. Similarly normalized phenotypic diversity based on difference in solution fitness can be given as:

$$\mathcal{PD} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{fitness\_dist(v_i, v_j)}{(n-1)n/2} \tag{7.8}$$

Phenotypic measures cannot provide the guidance we require, since they only tell whether the individuals are different or not, without determining the amount of difference. Furthermore, the idea of using fitness to measure diversity is debateable since solutions having the same fitness would be considered identical even when the solutions themselves are different. An operator like crossover can produce perfectly new solutions from same-fitness parents. Thus it is improper to consider potential parent solutions identical just because they have the same fitness. Still, distance between solutions is commonly measured in the fitness space [Sörensen and Sevaux 2004], probably because of the ease of the computation. Given that fitness evaluation is part of the standard evolutionary algorithm, phenotypic diversity measures require very little computation time in comparison with genotypic diversity (respectively given by Zhu and Liu [2004] as $O(n \log n)$ and $O(l^2 n^2)$). Compared with phenotypic diversity measures, genotypic measures are more distinct and more intimate with genetic operators. They do not precisely reflect the actual algorithmic distance between solutions, but as mentioned earlier it is neither easy nor cheap to compute algorithmic distance.

The previous arguments are illustrated by plotting the evolution of diversity measures in typical GA runs under different values of genetic parameters. Figure 7.1 compares the effect of mutation rate on genotypic and phenotypic diversity while crossover rate and selection probability are held constant. In this figure, population diversity generally declines with the number of generations, as the promising regions are discovered. Increasing mutation tends to increase diversity. However, while genotypic diversity declines consistently with generations, phenotypic diversity evolves erratically.

The effect of crossover rate on genotypic and phenotypic diversity measures is compared under constant mutation rate and selection probability as illustrated in Figure 7.2. Similar

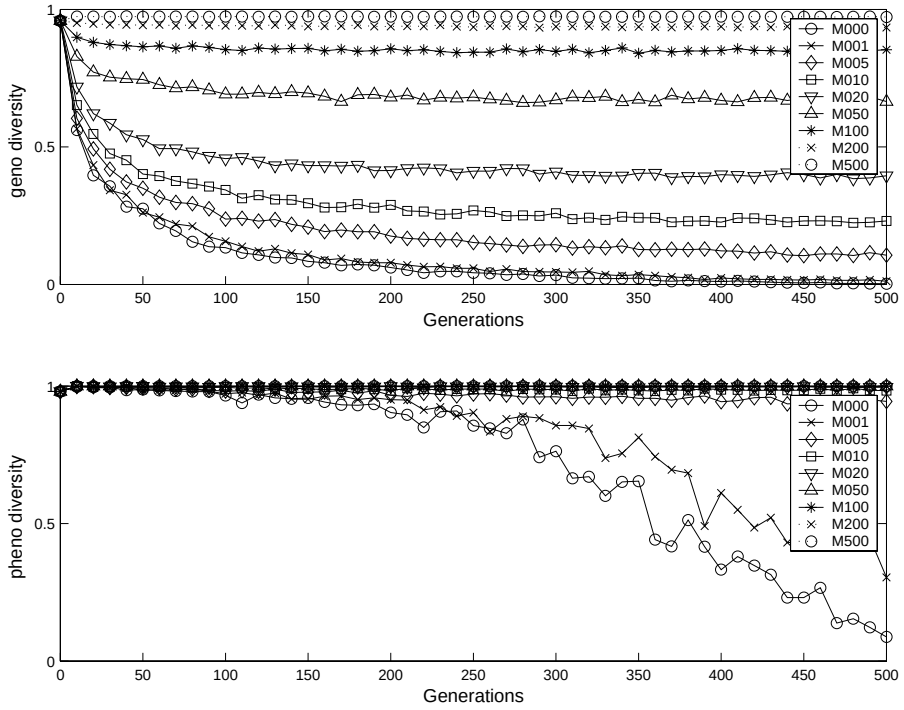


Fig. 7.1: Effect of mutation rate on measures of diversity. Values in the legend give mutation rate per thousand, e.g., M005 means mutation rate of 0.005. Genotypic diversity reflects the fact that the algorithm gradually converges with time to some solution. However, phenotypic diversity is not distinct with most mutation rates.

to mutation, crossover is a diversification operator (though less a diversifier than mutation since crossover cannot create new genetic material in the population).

The effect of selection probability on diversity is opposite to that of mutation and crossover. The evolution of both diversity measures is compared in Figure 7.3 under various selection probabilities and constant rates of crossover and mutation. Both measures of diversity increase as probability of selection decreases; still, the inconsistency of the evolution of phenotypic diversity is clear in this figure.

In summary, genotypic measures not only are convenient to assess population diversity but also reflect the values of genetic parameters more accurately than phenotypic measures.

7.2.3 Pair-wise versus population-best diversity measures

The main disadvantage of genotypic measures is their relative high computation costs. One way to reduce their computational costs is to use a single aggregation point to represent the whole population and thus reduce cost by a factor of n, where n is the population size. Riget and Vesterstroem [2002] and Ursem [2002] use an average population point for their real-value encoded individuals. However, this approach does not work for COPs, where it is hard to define an average point for the population. Another approach is to use small population size [Sörensen and Sevaux 2004], but cost reduction may not warrant the risk of hindering the search by limiting population size.

In this thesis, the population-best is used as a reference point for measuring diversity. By reserving individual v_n for the population-best, the aggregated genotypic measure (d) of a population of size n can be given as:

$$d = \sum_{i=1}^{n-1} \frac{dist(v_i, v_n)}{(l-1)(n-1)} \tag{7.9}$$

This measure has two advantages. First, the use of a single aggregation point to represent the population greatly reduces costs of computing diversity, without imposing unnecessary limitations on the population size. Second, as evolutionary algorithms are designed to converge around the population-best, it is reasonable to measure the population diversity in terms of distances from the population-best solution rather than distances from an average point.

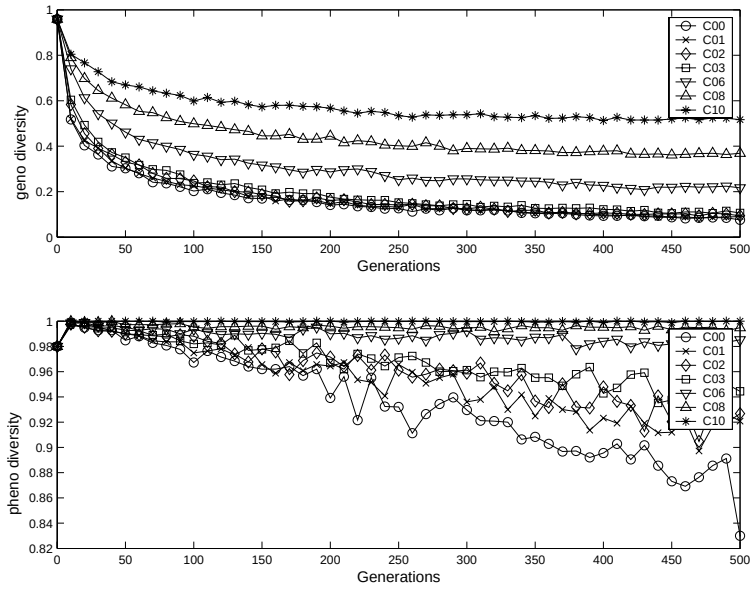


Fig. 7.2: Effect of crossover rate on measures of diversity. Values in the legend give crossover rate per ten; e.g., C06 means crossover rate of 0.6. Genotypic diversity declines smoothly with time whereas phenotypic diversity declines in a more erratic way.

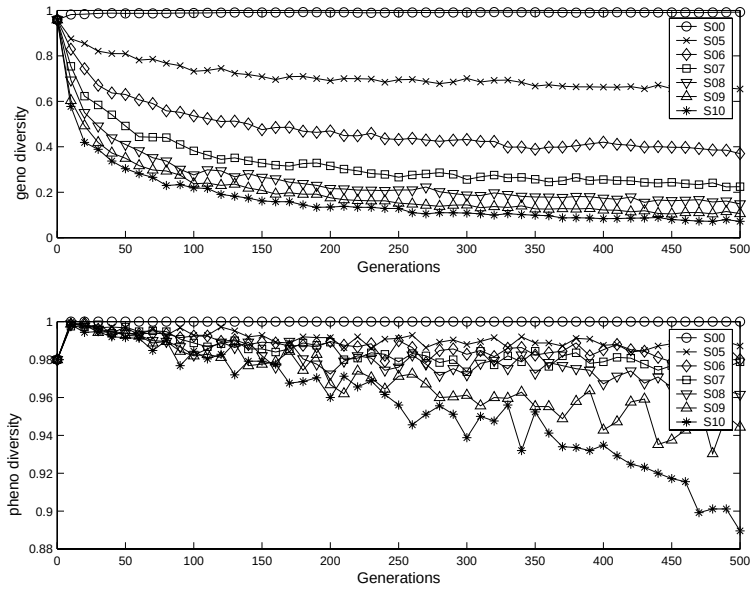


Fig. 7.3: Effect of selection probability on diversity measures. Values in the legend give selection probability per ten; e.g., S06 means selection probability of 0.6. Genotypic diversity declines smoothly with time whereas phenotypic diversity declines erratically.

The validity of the proposed diversity measure (d) is demonstrated by comparing it with the commonly used pair-wise measure $\mathcal{GD}$. Comparisons are carried out by plotting d against $\mathcal{GD}$, by comparing the evolution of both measures, and by measures, and by computing correlation coefficients. Different ranges of diversity are examined by using different combinations of mutation rate, crossover rate, and selection probability. Each parameter has three possible values labelled according to the diversity they tend to produce: `L` for parameter value causing low diversity, `M` for parameter value causing medium diversity, and `H` for parameter value causing high diversity. Thus, the combination `LLL` keeps the diversity as low as possible while the combination `HHH` drives diversity to maximum values. Parameter values used are given in Table 7.1. These values are selected so that a wide range of diversity is considered while the parameters remain close to commonly used values.

Table 7.1: Values of genetic parameters combinations

	`L`	`M`	`H`
Mutation	0.001	0.005	0.100
Crossover	0.0	0.3	1.0
Selection	1.0	0.55	0.50

Figures 7.4, and 7.5 show scatter plots of d against $\mathcal{GD}$ for ten runs of each of the 27 possible combinations of genetic parameter values. The 45° trend is unmistakeable in all slides of these figures, which indicates a close correlation between both measures. Even in those slides where the diversity is very large (indicating randomness) both measures tend to give similar indications.

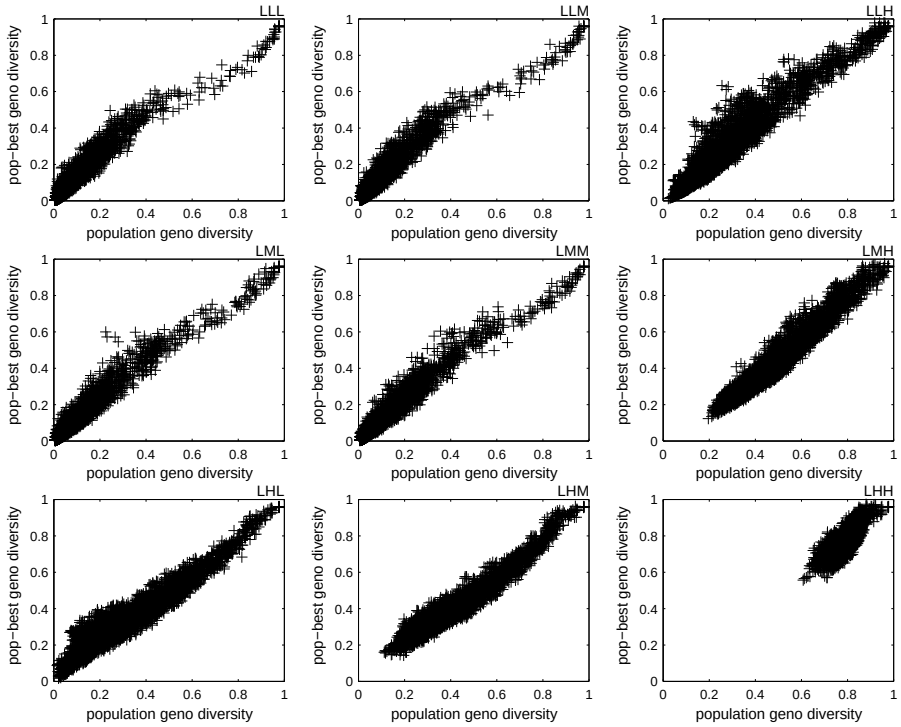


Fig. 7.4: Population-best vs. Population diversity for the K100 problem Part1

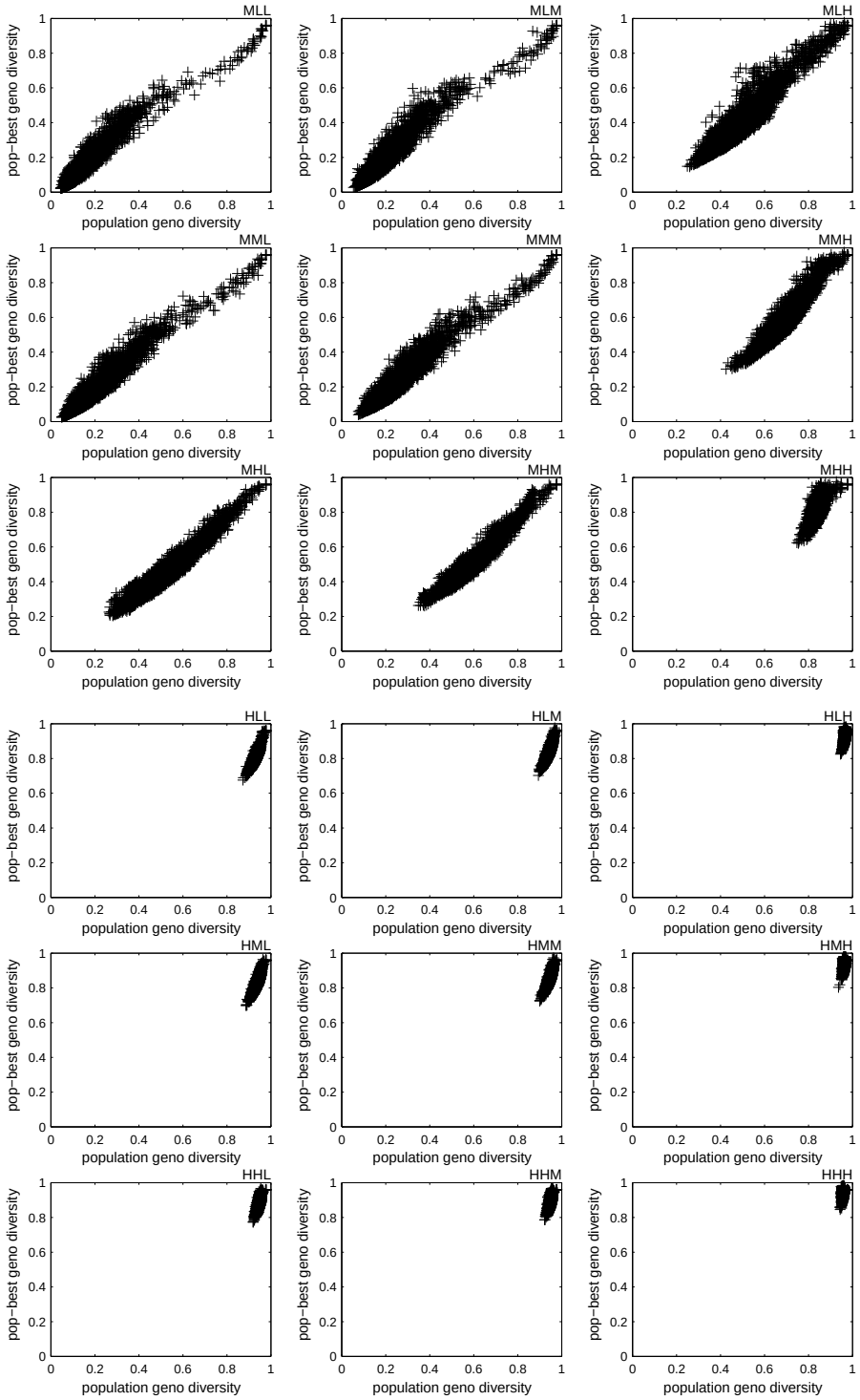


Fig. 7.5: Population-best vs. Population diversity for the K100 problem Part2

The evolution of both measures on the K100 problem is shown in Figure 7.6. The slides in the figure are the results of using three combinations of GA parameters for each diversity measure. Each slide uses ten GA runs. The figure clearly indicates that diversity diminishes as the search progresses, and that both measures of diversity give similar values for the population diversity.

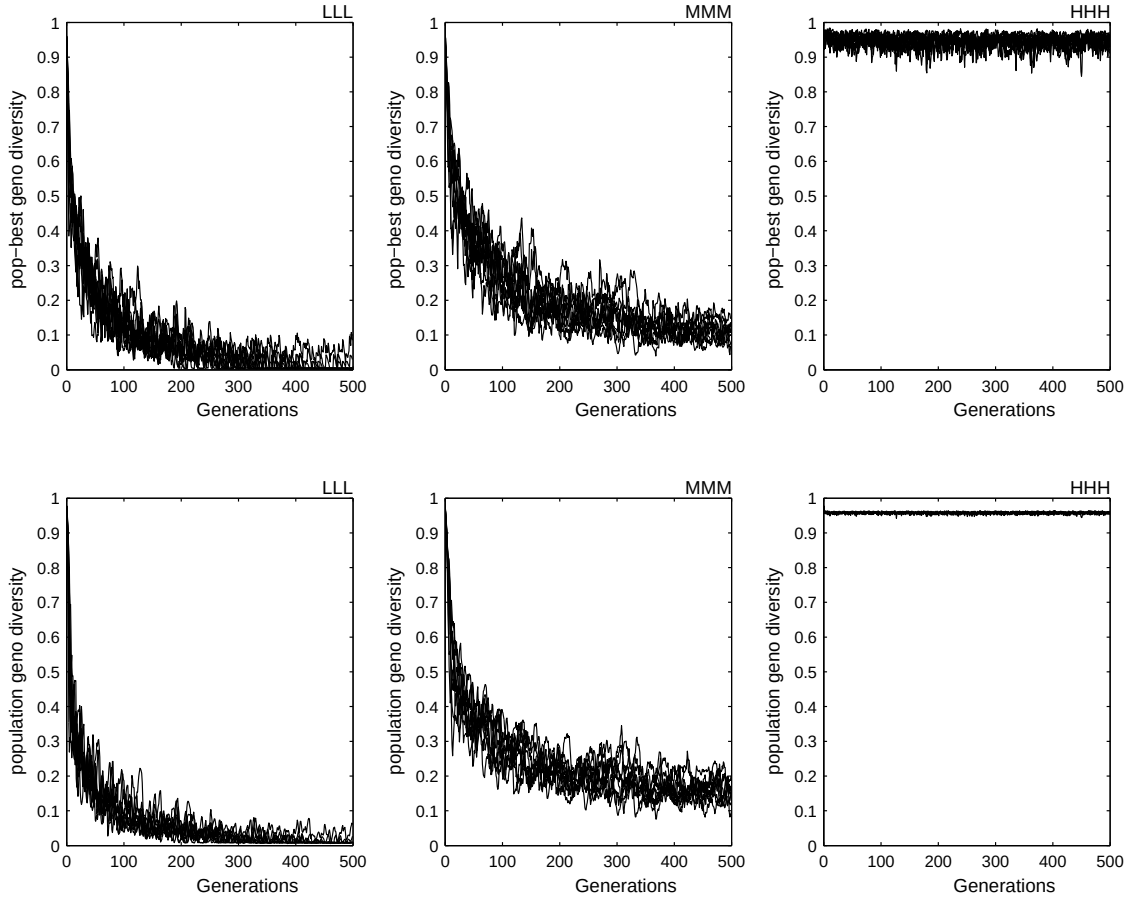


Fig. 7.6: Comparison of evolution of genotypic measures of diversity for the K100 problem. Population-best diversity measure (upper row) exhibits similar behavior to that of population pair-wise diversity measure (lower row)

The previous experiments are made in a static environment, where the problem starts from a random initial population that converges gradually with the number of generations. Two additional special cases in which both diversity measures may differ are examined here. In one case an environmental change takes place with the population already converged around some solution, as shown in Figure 7.7b. This case is imitated by a population that consists of duplicates of a good quality solution. In the other case the population is converged around some solution and a new better solution is discovered, as shown in Figure 7.7c. This case is imitated by a population in which the best solution is by far better than the rest of the individuals which are identical to each other. Correlation between both measures is compared on the three cases for different population sizes.

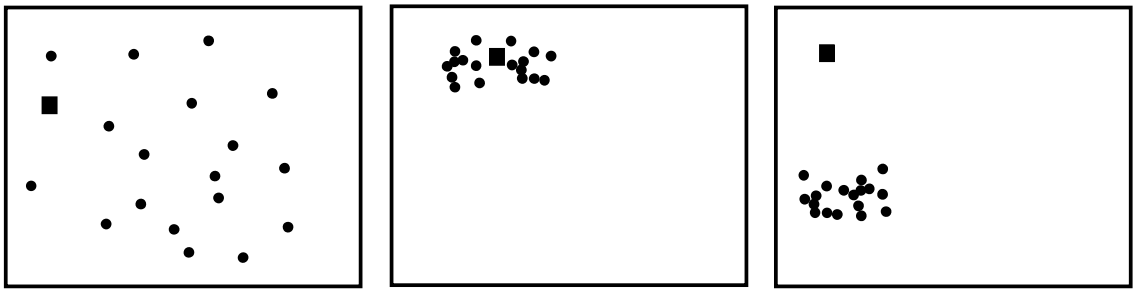


Fig. 7.7: Three cases of initial populations. In slide (a) the algorithm starts from a random initial population, in slide (b) the population is converged around a good quality solution, and in slide (c) the population is converged around a solution far from the population-best

The results are summarized in Table 7.2. The third column (pop0) gives the three cases of the initial population, 1 for random, 2 for a population converged around the population-best, and 3 for a population converged far from the population-best. The fourth column gives the commonly used Pearson's product moment-correlation coefficient (r), and the

fifth column gives Spearman's coefficient of rank correlation (r_s) [Siegel 1956; Parkess 2005] . Both measures are used to quantify correlation between two variables x and y in a sample of size n as follows:

$$r = 1 - \frac{S_{xy}}{\sqrt{S_{xx}S_{yy}}} \tag{7.10}$$

$$r_s = 1 - \frac{6\sum_{i=1}^{n}\delta_i^2}{n(n^2-1)} \tag{7.11}$$

where $S_{xx} = \sum_{i=1}^{n}(x_i - \overline{x})^2$, $S_{yy} = \sum_{i=1}^{n}(y_i - \overline{y})^2$, $S_{xy} = \sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})$, $\overline{x}$ is the sample mean of variable x, $\overline{y}$ is the sample mean of variable y and δ_i is the difference between the sample's rank of x_i and rank of y_i. Correlation coefficients can have values in the range from -1 to 1. In general, correlation coefficients between 0 and .33 indicate a weak relationship between measures, values between 0.34 and 0.66 indicate a medium strength relationship, and values over 0.67 indicate strong relationship. Negative coefficients indicate negative correlation.

Correlation between the averages of both diversity measures over a run is investigated in Table 7.2, with the number of carried out runs corresponding to the statistical sample size. The percentage correlations given in the table indicate strong correlation between both measures of diversity. The only exception is the `HHH` showing weak correlation; however, both diversity measures are so large, indicating a random search, because of the extreme values of the genetic parameters. The same evidence can be seen in the last four columns which show the average and standard deviation of both measures throughout the search process.

In short, both measures of diversity can be used interchangeably to express genotypic diversity within moderate ranges since the difference between both measures is very small in all practical ranges of the genetic parameters. Therefore, the small computational costs associated with population-best diversity measure advocate the use of this measure to control the search process, as shown in the following sections.

Table 7.2: Correlation between diversity measures under different states of convergence for the k100 problem

setting			correlation (%)		Pop-best		Population	
popsize	GA param	pop0	Pearson	Spearman	avg	sd	avg	sd
5	LLL	1	100	100	0.0006	0.02	0.0008	0.03
		2	NaN	NaN	0	0	0	0
		3	88	100	0.0006	0.02	0.0004	0.01
	MMM	1	99	81	0.0225	0.02	0.0384	0.03
		2	93	94	0.0179	0.01	0.0348	0.01
		3	89	94	0.0186	0.02	0.0353	0.02
	HHH	1	38	38	0.697	0.11	0.754	0.05
		2	58	38	0.6913	0.11	0.7508	0.05
		3	31	36	0.6932	0.11	0.7518	0.05
20	LLL	1	100	96	0.0391	0.07	0.0361	0.08
		2	69	80	0.0052	0	0.0085	0
		3	72	81	0.0099	0.06	0.0112	0.03
	MMM	1	99	95	0.0968	0.08	0.1259	0.09
		2	94	93	0.0433	0.01	0.0766	0.02
		3	66	92	0.0479	0.06	0.0788	0.04
	HHH	1	32	34	0.9018	0.04	0.9199	0.01
		2	93	30	0.8929	0.07	0.9164	0.05
		3	-17	33	0.8992	0.04	0.9174	0.04
50	LLL	1	99	96	0.0965	0.12	0.0891	0.13
		2	70	66	0.0076	0.01	0.0099	0
		3	67	67	0.0268	0.12	0.0182	0.06
	MMM	1	100	99	0.2443	0.15	0.287	0.15
		2	98	90	0.0767	0.02	0.1279	0.03
		3	55	86	0.1036	0.12	0.1452	0.07
	HHH	1	26	26	0.9479	0.02	0.9559	0
		2	96	27	0.9319	0.09	0.9472	0.07
		3	-49	13	0.9481	0.02	0.9483	0.07

7.3 Adaptive diversity model

The linear model of Chapter 6 focuses primarily on reactions to environmental changes and is inflexible during static phases since the rate of parameter change is fixed a priori. That is, it addresses obsolete convergence but ignores any premature convergence. This shortcoming is alleviated in the adaptive diversity model (ADM) presented in this section. In addition to being adaptive to environmental changes, the ADM model adapts to changes in population diversity to direct the search towards unexplored regions of the search space or towards partially tested regions that have shown promising results.

ADM is comparable in many ways to two diversity-based models from the literature, namely the *diversity-guided evolutionary algorithms* (DGEA) proposed by Ursem [2002] for continuous optimization and the *diversity-controlling adaptive genetic algorithm* (DCAGA) proposed by Zhu [2003] for VRP. DGEA uses two diversity limits to alter the search between an explorative phase and an exploitative phase; however, such an approach may not be suitable for dynamic problems as it is likely to require comparatively more generations and hence solution time. DCAGA uses one target diversity to control genetic parameters; however, targeting one value for diversity does not seem the best idea for two reasons. First, diversity measures are not distinct and their values carry some ambiguity. Second, for the same reasons we believe that an optimal set of genetic parameters does not exist (or at least changes during the run), we conclude that the optimal (target) diversity is also dynamic. Consequently, ADM uses two diversity limits to regulate genetic parameters during static phases of the search without reducing the search to *pure* exploitation or *pure* exploration.

7.3.1 The diversity approach

Considering the mutation operator for a start, ADM can be described as follow. When an environmental change is detected (at $t = t_m$), the mutation rate is set to an upper

limit $\overline{\mu}$ as was done in the LM model. Subsequently (while the environment is static) the ADM measures the population diversity $d(t)$ and compares it to two reference values, an upper limit d_h and a lower limit d_l. A mutation rate $\mu(t)$ for the current generation can then be computed according to this comparison. The following formula gives the variation of mutation rate in the cycle between two consecutive environmental changes (i.e. $t_m \leq t < t_{m+1}$):

$$\mu(t) = \begin{cases} \overline{\mu}, & t = t_m \\ Z_l \cdot \big(\overline{\mu} - \mu(t-1)\big) + \mu(t-1), & t \neq t_m,\ d(t) < d_l \\ \mu(t-1) - Z_h \cdot \big(\mu(t-1) - \underline{\mu}\big), & t \neq t_m,\ d(t) > d_h \\ \mu(t-1), & t \neq t_m,\ d_l \leq d \leq d_h \end{cases} \tag{7.12}$$

$$\text{where } Z_l = min\left\{\frac{d_l - d(t)}{D}, 1\right\},\ Z_h = min\left\{\frac{d(t) - d_h}{D}, 1\right\},\ D = d_h - d_l$$

The ADM is extended to include adaptive crossover rate and selection probability as well. The formula for adaptive crossover rate $\chi(t)$ is similar to that of mutation and is given by the following equation.

$$\chi(t) = \begin{cases} \overline{\chi}, & t = t_m \\ Z_l \cdot \big(\overline{\chi} - \chi(t-1)\big) + \chi(t-1), & t \neq t_m,\ d(t) < d_l \\ \chi(t-1) - Z_h \cdot \big(\chi(t-1) - \underline{\chi}\big), & t \neq t_m,\ d(t) > d_h \\ \chi(t-1), & t \neq t_m,\ d_l < d(t) < d_h \end{cases} \tag{7.13}$$

where $\underline{\chi}$ and $\overline{\chi}$ are the lower and the upper limits of crossover rate respectively. Z_l and Z_h are as given earlier in the mutation formula 7.12.

The formula for adaptive selection $s(t)$ is given by Equation 7.14. Here, selection probability is changed in an opposite manner to that of mutation and crossover since high selection pressure tends to reduce population diversity.

$$s(t) = \begin{cases} \underline{s}, & t = t_m \\ s(t-1) - Z_l \cdot \big(s(t-1) - \underline{s}\big), & t \neq t_m, d(t) < d_l \\ Z_h \cdot \big(\overline{s} - s(t-1)\big) + s(t-1), & t \neq t_m, d(t) > d_h \\ s(t-1), & t \neq t_m,\ d_l < d(t) < d_h \end{cases} \tag{7.14}$$

where $\underline{s}$ and $\overline{s}$ are the lower and the upper limits of selection probability respectively; and Z_l, and Z_h are as given earlier in the mutation formula 7.12.

Figure 7.8 illustrates the general principle of the ADM, and how it drives genetic parameters toward exploration or exploiting in response to measured diversity. In this figure, P can be the value of any of the controlled genetic parameters μ, χ or s. P_r corresponds to maximum exploration values; i.e., $\overline{\mu}$, $\overline{\chi}$ or $\underline{s}$, whereas P_t corresponds to maximum exploitation values; i.e., $\underline{\mu}$, $\underline{\chi}$, or $\overline{s}$.

The pseudo code for a dynamic solver using the ADM is given in Figure 7.9.

7.3.2 Diversity model investigation

Under the ADM model, diversity and the genetic parameters go hand-in-hand as shown in Figure 7.10, which compares the evolution of population diversity and mutation rate in the LM and ADM models. Thus, with the ADM model, one needs to specify two values for population diversity (d_l and d_h), and two values (a minimum limit and maximum limit) for each of the adaptive genetic parameters. The graphs in Figure 7.11 show the relation between adjusted parameter values and old parameter values under different ranges of unbiased diversity (d_h - d_l) and measured diversity d.

This increase in the number of input parameters does not mean an increase in the efforts and time of tuning. In fact, the use of ADM can help reduce tuning efforts. Actually, only the two limits of diversity need to be tuned, while limits of genetic parameters can be lenient, since their role now is to avoid extreme values of the corresponding parameter. In

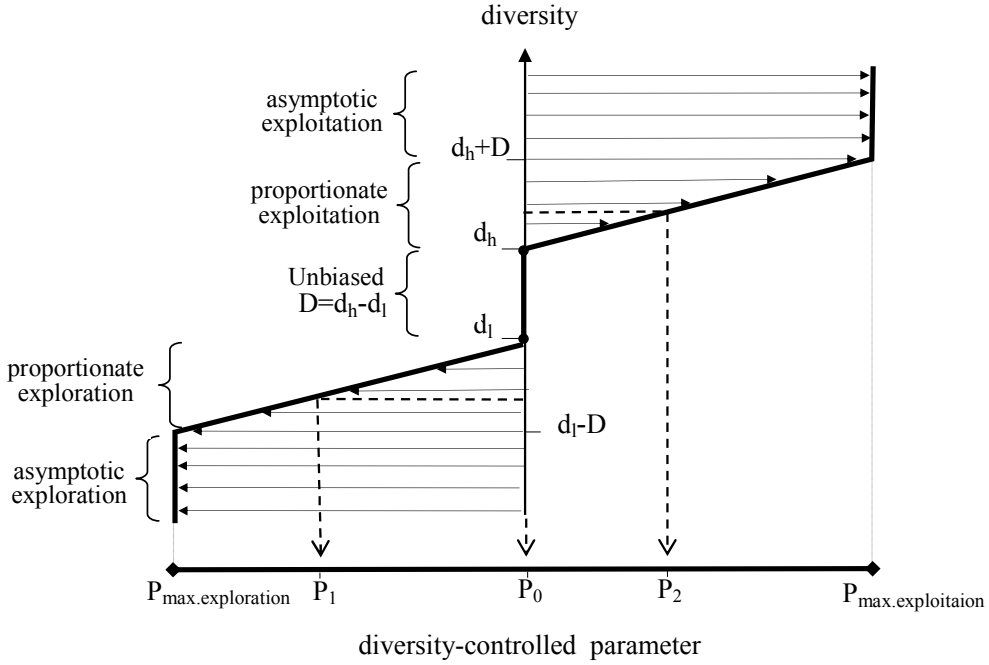


Fig. 7.8: Diversity range is divided into five regions. Low diversity maps the genetic parameter into a more explorative value (e.g., P_1) and high diversity maps it into a less explorative value (e.g., P_2). Diversity values between d_l and d_h do not change the current values of the genetic parameters (the parameter is mapped into its original value P_0). The farther the diversity is from the unbiased range, the more change to the genetic parameter. Diversity in the asymptotic regions maps the parameter into one of its extreme values ($P_{\text{max.exploration}}$ or $P_{\text{max.exploitation}}$) .

other words, since the primary factor in determining the values of genetic parameters is diversity, efforts will be directed to the tuning of diversity limits rather than the tuning of genetic parameters. Limits on the genetic parameters can be determined by conducting controlled experiments using fixed parameter values to exclude parameter ranges where performance deteriorates noticeably.

The effect of both limits of diversity on the performance of the ADM model is inves-

```
Procedure ADM
    i = 0;   // initiate instances counter
    g = 0;   // initiate generations counter
    pop = Generate(0);   // generate initial population
    Evaluate( pop );
    repeat
        while quiescent environment
            g = g+1;
            Select( pop );
            Cross( pop );
            Mutate( pop );
            best_of_geration[g] = Evaluate( pop );
            pop_div = MeasureDiversity( pop );
            params = DiversityAdaptParam( params , pop_div );    // apply ADM Eqns
        endwhile                                                  7.12, 7.13, 7.14
        i = i+1;
        AdaptPopulation( pop , strategy );
    until terminating condition

Procedure AdaptPopulation( pop , strategy )
    Evaluate(pop);
    Repair(pop);
    ApplyDynamicStrategy(pop);    // apply ADM, case t = t_m in Eqns 7.12, 7.13, 7.14
```

Fig. 7.9: Pseudo code for a diversity controlled dynamic solver

tigated under different environmental dynamics. In order to apply the ADM model to a specific dynamic problem, a static instance of the problem (or a few instances from similar problems) is chosen to construct dynamic problems using the mapping swap scheme described in Section 5.4. The dynamic problems are solved using different values of diversity limits.

As an example, a 100-city TSP problem (K100) is used to construct nine dynamic

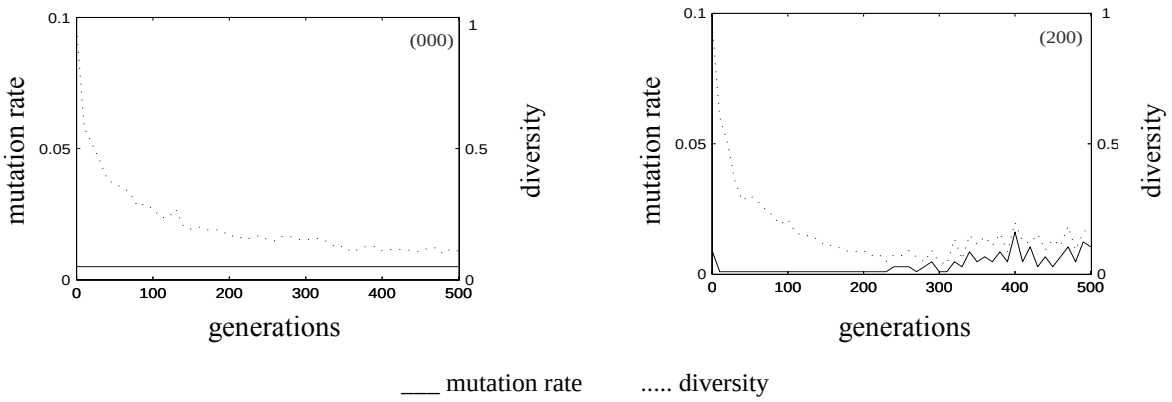


Fig. 7.10: Diversity-controlled mutation rate. Under traditional constant mutation rate (LH graph) diversity declines steadily with time. But with controlled mutation (RH graph), large diversity at the beginning of the run drives the rate of mutation to its lowest limit, and when diversity is small, the rate of mutation is increased. Thus mutation rate closely follows population diversity throughout the search.

problems using three values of severity and three periods of change. Each dynamic problem is solved by the ADM model using forty-eight combinations of diversity limits (six values of lower diversity limit ranging from 0.0 to 0.5 and eight values of upper diversity limits ranging from just above the lower limit to 1.0). The results of these experiments are summarized in Figure 7.12 for different levels of severity at an average period of change and in Figure 7.13 for different periods at an average severity of change. These results show that the ADM model gives the same performance under a wide range of diversity limits.

As well, Figure 7.12 suggests increasing both limits with the severity of change. This suggestion confirms the generally accepted view that population diversity should be increased to higher levels to continue the search after an environmental change.

On the other hand, Figure 7.13 suggests reducing diversity limits with the period of change. Therefore, instead of using fixed lower and upper diversity limits (d_l and d_h) in the

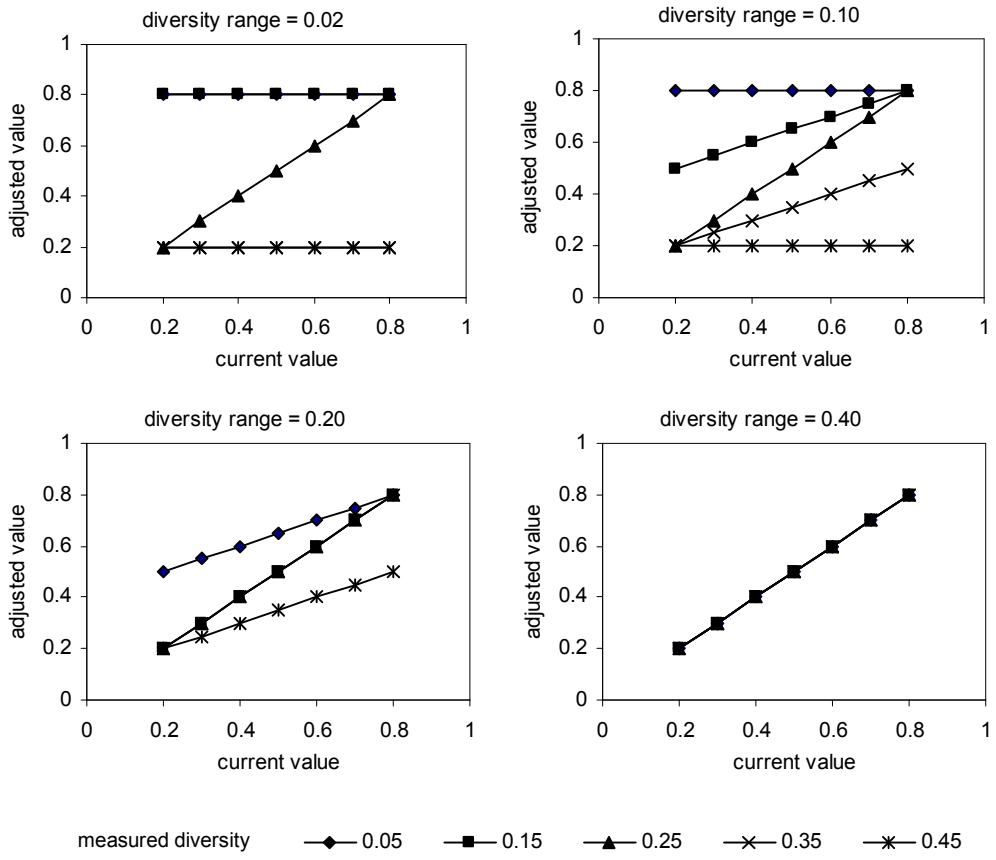


Fig. 7.11: The four slides are for different sizes of unbiased diversity range with a target diversity of 0.25. Graphs on these slides show the relation between the diversity-adjusted value of the parameter and the parameter's original value. Reducing the unbiased range makes the search oscillate more severely between exploitation and exploration.

ADM genetic control formulae 7.12, 7.13, and 7.14, it is more promising to use variable limits $d_l(t))$ and $d_h(t))$. That is, when the environment changes (say at time $t = t_m$), the lower diversity limit $d_l(t)$ is set to a large initial value d_{l0}. Subsequently, it is reduced with time until, after a period ρ, it reaches a final value d_{lf}. The following formula gives the variation of $d_l(t))$ in the cycle between two consecutive environmental changes (i.e.,

$t_m \leq t < t_{m+1}$)

$$d_l(t) = \begin{cases} d_{l0} - \dfrac{d_{l0} - d_{lf}}{\rho}(t - t_m), & t_m \leq t < t_m + \rho \\ d_{lf}, & t_m + \rho \leq t < t_{m+1} \end{cases} \tag{7.15}$$

Similarly, $d_h(t)$) is varied from a large initial value d_{h0} to a smaller final value d_{hf} by the following formula:

$$d_h(t) = \begin{cases} d_{h0} - \dfrac{d_{h0} - d_{hf}}{\rho}(t - t_m), & t_m \leq t < t_m + \rho \\ d_{hf}, & t_m + \rho \leq t < t_{m+1} \end{cases} \tag{7.16}$$

Under variable diversity limits, the ADM model is expected to work in a less random way and with a better overall performance. Initial and final values of diversity limits are set by controlled experimentation where the limits and period of change are varied systematically to determine their appropriate values. It is worth noting here that unlike the usual inconclusive tuning of genetic parameters, tuning of diversity limits is expected to be easier and more accurate.

7.4 Adaptive island model

In this section, an adaptive island model (AIM) is introduced to enable the well known island genetic algorithm to tackle dynamic problems. Traditionally an IGA divides the population into several subpopulations or islands allocated to one or more processor. These islands evolve independently from each other for a period of time, called the migration interval. At the end of each migration interval, a few individuals are exchanged between islands. IGAs reportedly gave better solution quality and less computation time even when the model was implemented in a serial manner, as mentioned in Chapter 3. IGAs can maintain population diversity without destroying individuals of good quality whereas

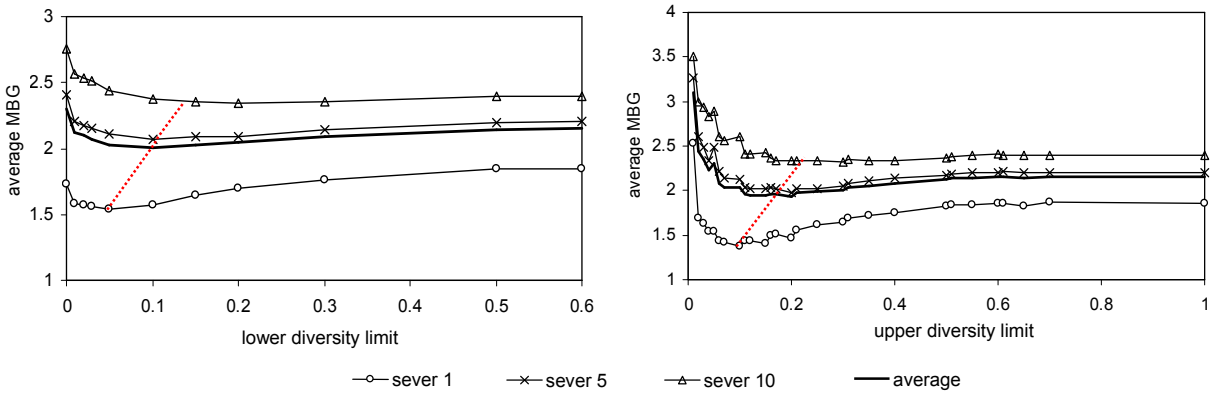


Fig. 7.12: Tuning diversity limits at average period of change. Dotted lines are drawn to show the trend of the variation of the best diversity limits.

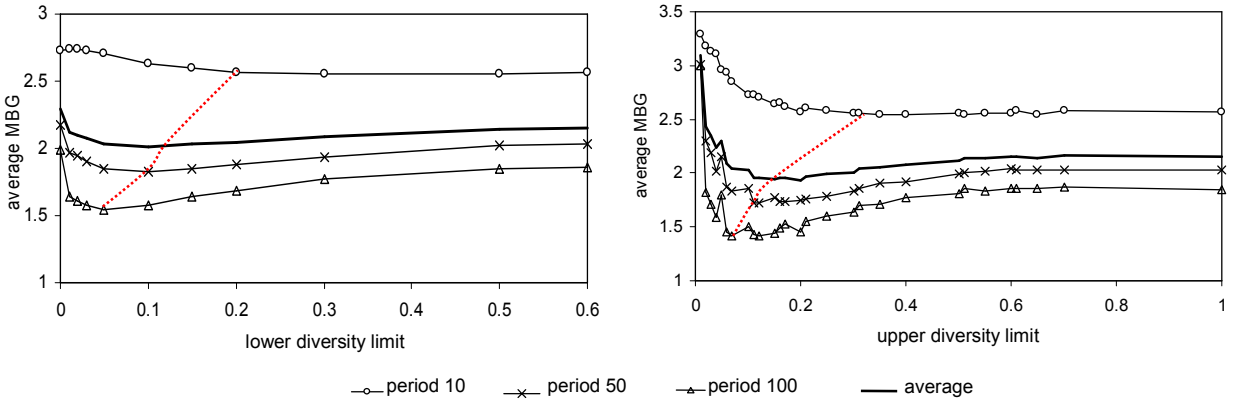


Fig. 7.13: Tuning diversity limits at average severity of change. Dotted lines are drawn to show the trend of the variation of the best diversity limits.

techniques relying on large mutation rates often lose good quality individuals. Furthermore, IGAs are more suitable to dynamic problems since they act as implicit memory by retaining individuals found in the vicinity of peaks.

AIM shares many features with other multiple population evolutionary algorithms that have been used successfully in dynamic problems. However, unlike SBGA

[Oppacher and Wineberg 1999] and SOS [Branke et al. 2000], AIM uses a fixed number of same size islands. As well, no specific island is given the role of base or core island in AIM; the island that contains population-best is considered the current base island. AIM maintains several good solutions at a time, each of which is the center of an island. In this way all islands participate in exploring the search space and at the same time exploit good individuals. AIM is more like MGA [Ursem 2000], but still does not rely on the continuity nature of the variables to guide the search process. In addition, AIM uses controlled diversity genetic operators, in a way similar to that described in the previous section. To avoid of confusion, the term *population* hereafter is reserved to refer to all individuals in all islands under consideration while the term *island* is used to refer to one subpopulation.

7.4.1 The island approach

The general idea of AIM is to use a fixed number of islands guided by two measures that are based on the genotypic distance between individuals, an island diversity measure and a population diversity measure. Island diversity is measured as the sum of distances from individuals in the island to the island-best, whereas population diversity is based on distances from island-best to global-best (the best individual in all islands).

Each island is basically a small population that evolves under the control of its own diversity independently from other islands. In this way an island plays the role of a niche, as it consists of individuals that are close to each other. The best individual in the island is used as an aggregate point for measuring island diversity and as a representative of the island in measuring inter-island diversity (or simply population diversity).

With the islands charged with maintaining population diversity, the algorithm becomes less reliant on the usual (destructive) high rates of mutation. Furthermore, mutation now is required to maintain diversity with individual islands (not within entire population), thus lower rates of mutation are needed. Therefore, mutation rate in AIM, though still diversity dependent, has now a lower upper limit.

In order to avoid premature convergence due to islands being isolated from each other, individuals are forced to migrate from one island to another at pre-defined intervals in a ring-like scheme, as illustrated in Figure 7.14. This scheme helps impart new genetic material to destination islands and increase survival probability of high fitness individuals.

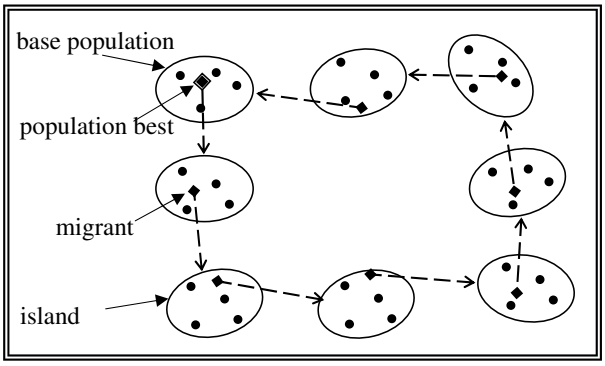


Fig. 7.14: Ring migration scheme, with the best individuals migrating among islands

On the global level, AIM is required to keep islands in different parts of the search space. This requirement is achieved by measuring inter-island diversity before migration and by mutating duplicate islands. If two islands are found very close to each other, one of them is considered duplicate, and consequently its individuals are mutated to cover a different region of the search space. Elite solutions consisting of the best individual from each island are retained throughout the isolation period. During migration, elite solutions are not lost since best individuals are forced to migrate to new islands.

At environmental changes, each island is re-evaluated and its genetic parameters are reset to their respective upper diversity values. During quiescent phases of the environment, genetic parameters are changed in response to individual island diversity measures. That is, the parameter control formulae (7.12 through 7.16) are applied to each island separately. A pseudo code for AIM is given in Figure 7.15.

```
Procedure AIM
    i = 0;   // initiate instances counter
    g = 0;   // initiate generations counter
    pop = Generate(isl[1], isl[2], ..., isl[n_islands]);
    Evaluate( pop );
    repeat
        while quiescent environment
            EvolveIslands( pop , g );
            PerformMigration( pop );
        endwhile
        i = i+1;
        AdaptPopulation( pop , strategy );
    until terminating condition

Procedure EvolveIslands( pop , g )
 for  (k=1 to n_islands)
    repeat
      g = g+1;
      Select( isl[k] );
      Cross( isl[k] );
      Mutate( isl[k] );
      best_of_isl[k] = Evaluate( isl[k] );
      isl_div = MeasureDiversity( isl[k] );
      diversity_limits = LinearlyAdapt( diversity_limits , g );
      params[k] = DiversityAdaptParam( params[k] , isl_div );    // apply ADM Eqns
    until end of isolation period                                  7.12, 7.13, 7.14
      pop_best = Best(pop_best,  best_of_isl[k] );
 endfor

Procedure AdaptPopulation( pop , strategy )
 Evaluate(pop);
 Repair(pop);
 ApplyDynamicStrategy(pop);    // apply ADM, case t = t_m in Eqns 7.12, 7.13, 7.14

Procedure PerformMigration( pop )
 pop_div = MeasureDiversity( isl[1], isl[2], ..., isl[n_islands] );
 for  (k=1 to n_islands)
    MutateIsland( isl[k], pop-div);    // mutate current island if it is too close to another island
    Migrate( isl[k] );
 endfor
```

Fig. 7.15: Pseudo code for AIM

7.4.2 Island model investigation

The effectiveness of island models is commonly investigated with respect to three basic parameters: migration rate (number of individuals sent from islands to the base population), migration interval (number of generations between subsequent migrations), and the number of islands constituting the entire population. In general, large intervals of migration and small numbers of immigrants lead to isolated islands that may suffer premature convergence, whereas frequent migrations and large numbers of immigrants lead to global mixing of individuals where the islands virtually reduce to a single large population. Still, reports on static implementation of IGAs [Whitley et al. 1999] assert the difficulty of tuning these parameters.

For dynamic problems, however, AIM relies on the notion that each island is represented by its best individual, thus only the best individual in each island migrates. This restriction on migration has the advantage of reducing computational costs and tuning efforts. Thus investigations in this section are limited to the role of the number of islands and migration interval. Two sets of experiments are reported here using a population of 50 individuals and fixed genetic parameters. The first set uses various numbers of islands (ranging from 2 to 25) with a constant migration interval. The second set uses a fixed number of islands (five) to investigate different intervals of migration (the smallest interval equals ten evaluations and the largest interval equals the period of environmental change).

The results of these experiments on the k100 problem (made dynamic using a mapping benchmark scheme) are reported in this section. Figure 7.16 shows the effect of changing the number of islands on solution quality. We first note that problem dynamics have little effect on the best number of islands. All slides in the figure suggest avoiding a large number of islands (i.e., islands of small size). These figure suggests that the optimal number of islands is five. Repeating the experiments on the be52 problem and the p442 problem (see Section 8.2.2 for details of these problems) give nearly the same recommendation on the best number of islands. Of course this number might change with other problems or

if a different overall population size is used; however, experiments on p442 and be52 in addition to those on k100 suggest that there is always an optimal number of islands that can be found by tuning the number of islands with a mapping benchmark scheme as is done in the current experiments.

Figure 7.17 shows the effect of changing the length of migration interval on solution quality. All slides in this figure agree on avoiding intermediate intervals: For slowly changing environments, frequent migration outperforms rare migration, but for fast environmental changes, it is much better to prevent migration. These results can be explained as follows. In a fast changing environment, it is important to quickly converge even at the risk of premature convergence around a moderately good solution, thus the fewer migrations the better. But in slowly changing environments frequent migrations help avoiding premature convergence and give a better chance of finding good solution.

The experiments are repeated using different base problems but the previously described trends of migration interval and number of islands are consistent. Hence, the implementations of AIM use five islands with adaptive migration. That is, migration is prevented during the initial period following an environmental change. Once island diversity d becomes smaller than the lower diversity limit d_l, migration is performed at every generation until the next environmental change.

As well, this scheme will also be awarding if several processors are used. With each island allocated to a single processor, the rate of migration is a primary factor that determines the cost of communication between processors. The proposed AIM model reduces this cost when the environment is fast changing; i.e., when reduction in run time is very desirable.

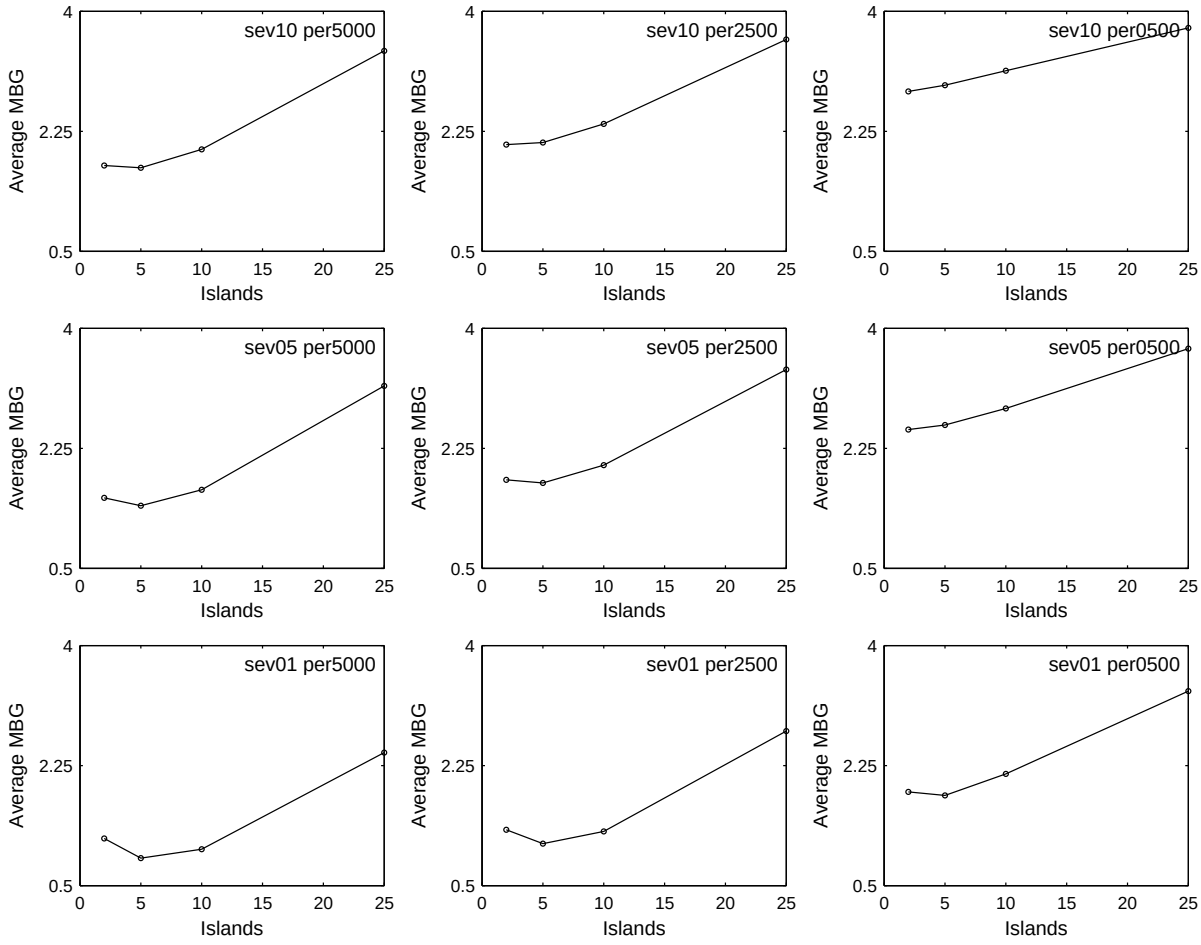


Fig. 7.16: Effect of number of islands on solution quality under different problem dynamics. Each of the nine slide corresponds to a specific severity of change and period of change, e.g., sev10 per5000 slide is for a problem with a severity of 10 shifts per environmental change and 5000 evaluations per environmental change.

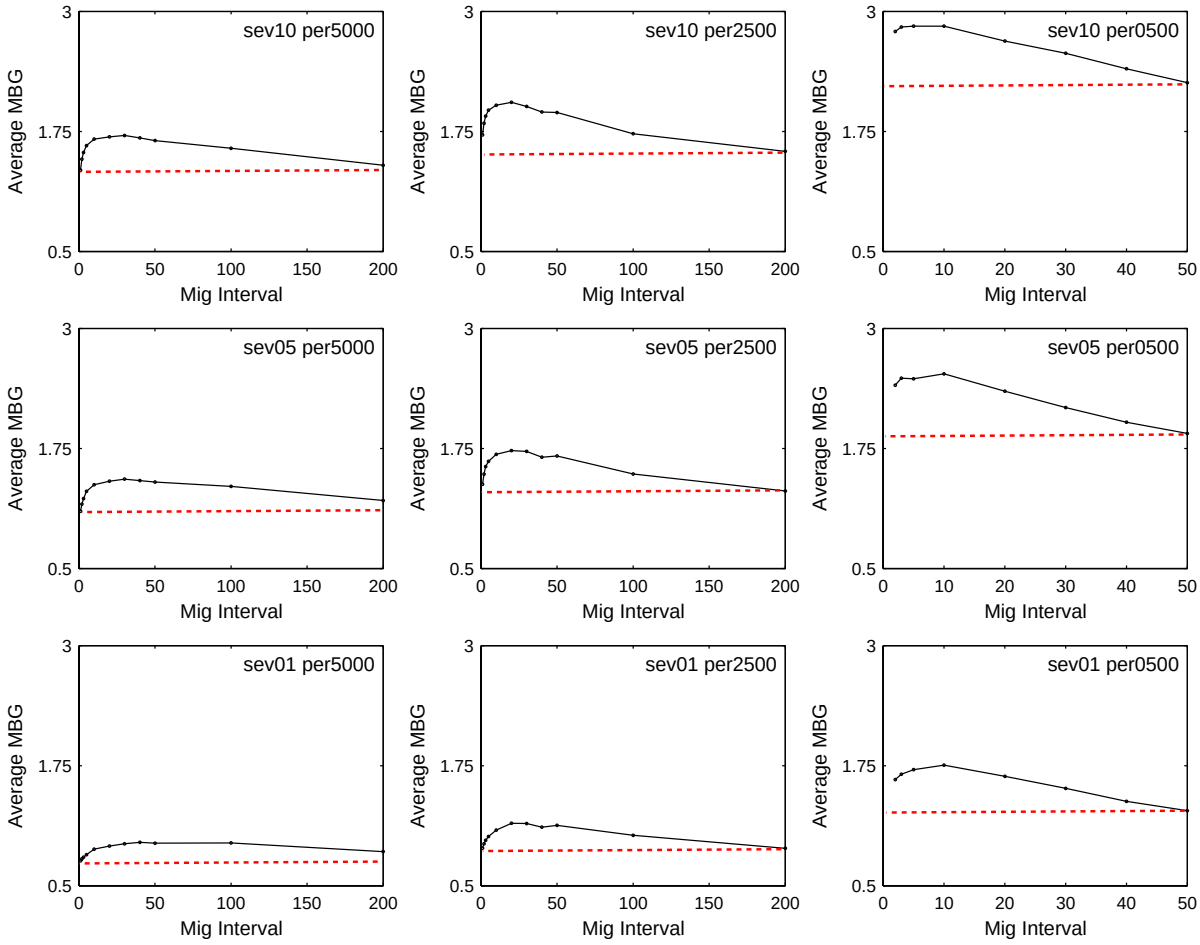


Fig. 7.17: Effect of migration interval on solution quality under different problem dynamics. Migration interval is given in tens of individual evaluations. Dotted lines are drawn to show the relatively good performance at small and large migration intervals in contrast to poor performance at intermediated intervals. Each of the nine slide corresponds to a specific severity of change and period of change, e.g., sev10 per5000 slide is for a problem with a severity of 10 shifts per environmental change and 5000 evaluations per environmental change.

7.5 Adaptive hybridized models

One of the most important implications of the no-free-lunch theorems is that the effectiveness of a general algorithm on a given problem can be increased by integrating problem-specific knowledge in the algorithm. Indeed, the effectiveness of EAs on COPs can often be greatly enhanced by hybridizing with some local search techniques [Maniezzo V. 2002]. Such hybrids are known as *hybrid genetic algorithms* [He and Mort 2000], *genetic local search* [Merz and Freisleben 1997], or *memetic algorithms* (MAa) [Moscato 1999].

Typically, local search is used to enhance the solutions produced by crossover and mutation before they are retained in the population, and most MAs in the literature [Krasnogor and Smith 2005] apply local search to all individuals in the population. In fact, Merz and Freisleben's MAs, which are among the most successful metaheuristics, use *exhaustive* local search, in which all individuals in the population undergo a local search that terminate in a local optimum. Under this approach, the end-of-generation population consists entirely of local optima. However, the user of an exhaustive local search approach presumes that there is "unlimited" time for algorithm execution, since for a lot of COPs the number of local search iterations to local optimality is not polynomially bounded [Johnson et al. 1988]. Furthermore, this approach can rapidly lead to a profound loss of diversity. Therefore, while exhaustive local search might be appropriate for solving static problems, it does not seem to be the best choice for dynamic problems, where both time and diversity are of utmost importance. Hence, it is worth investigating alternatives to exhaustive local search.

Most MA design issues that are deemed important for static problems are centered on finding the best tradeoff between global evolutionary search and embedded local search [Krasnogor and Smith 2005]. The models presented in this section for dynamic problems sidestep these issues by clearly specifying the task of the local search in contrast to that of the evolutionary algorithm. The EA is charged with handling the dynamism at environ-

mental changes whereas local search is employed in static phases to enhance the solutions. In this way, the adaptive ability of ADM and AIM models is retained while abundant literature on local search is directly exploited. Consequently, of the design issues addressed in in Krasnogor and Smith [2005], the allocation of local search iterations has the utmost importance in the proposed methodology.

7.5.1 The hybrid approach

In this section, we propose to hybridize the evolutionary algorithm with local search heuristics in the following manner: At the end of each generation individuals in the population undergo a local search that uses the *best-accept* strategy; that is, the best solution in the *searched* neighborhood of the individual under consideration is compared with the current solution. Then, the best of the two solutions is retained as the current solution. Only a fraction of the neighborhood of the current solution is explored, and only a limited number of local search iterations per generation are executed.

Integrating local search with EA in this manner, two new models (AHDM, and AHIM) can be built from the previously introduced ADM and AIM models (see Figure 7.18 for the pseudo code of AHIM). In both models, the evolutionary algorithm adapts to environmental changes in the same way suggested for the ADM and AIM models. Thus, the main task in this section is to establish an efficient scheme to integrate local search in the host EA.

There are several ways to improve the efficiency of a local search heuristic. Merz [2004] shows many techniques that use domain knowledge for improving the efficiency of a local search. However, for dynamic problems there will always be a limit for the number of local search iterations. The question that arises with this limitation is: What is the best way to invest the available iterations? For example, should we distribute them evenly among all individuals of the population or allocate them to the population-best individual only?

```
Procedure AHIM
    i = 0;   // initiate instances counter
    g = 0;   // initiate generations counter
    pop = Generate(isl[1], isl[2], ..., isl[n_islands]);
    Evaluate( pop );
    repeat
        while quiescent environment
            EvolveIslands( pop , g );
            PerformMigration( pop );
        endwhile
        i = i+1;
        AdaptPopulation( pop , strategy );
    until terminating condition

Procedure EvolveIslands( pop , g )
 for  (k=1 to n_islands)
    repeat
      g = g+1;
      Select( isl[k] );
      Cross( isl[k] );
      Mutate( isl[k] );
      LocalSearch( isl[k] );
      best_of_isl[k] = Evaluate( isl[k] );
      isl_div = MeasureDiversity( isl[k] );
      diversity_limits = LinearlyAdapt( diversity_limits , g );
      params[k] = DiversityAdaptParam( params[k] , isl_div );    // apply ADM Eqns
    until end of isolation period                                          7.12, 7.13, 7.14
      pop_best = Best(pop_best,  best_of_isl[k] );
 endfor

Procedure AdaptPopulation( pop , strategy )
 Evaluate(pop);
 Repair(pop);
 ApplyDynamicStrategy(pop);    // apply ADM, case t = t_m in Eqns 7.12, 7.13, 7.14

Procedure PerformMigration( pop )
 pop_div = MeasureDiversity( isl[1], isl[2], ..., isl[n_islands] );
 for  (k=1 to n_islands)
    MutateIsland( isl[k], pop-div);    // mutate current island if it is too close to another island
    Migrate( isl[k] );
 endfor
```

Fig. 7.18: Pseudo code for AHIM

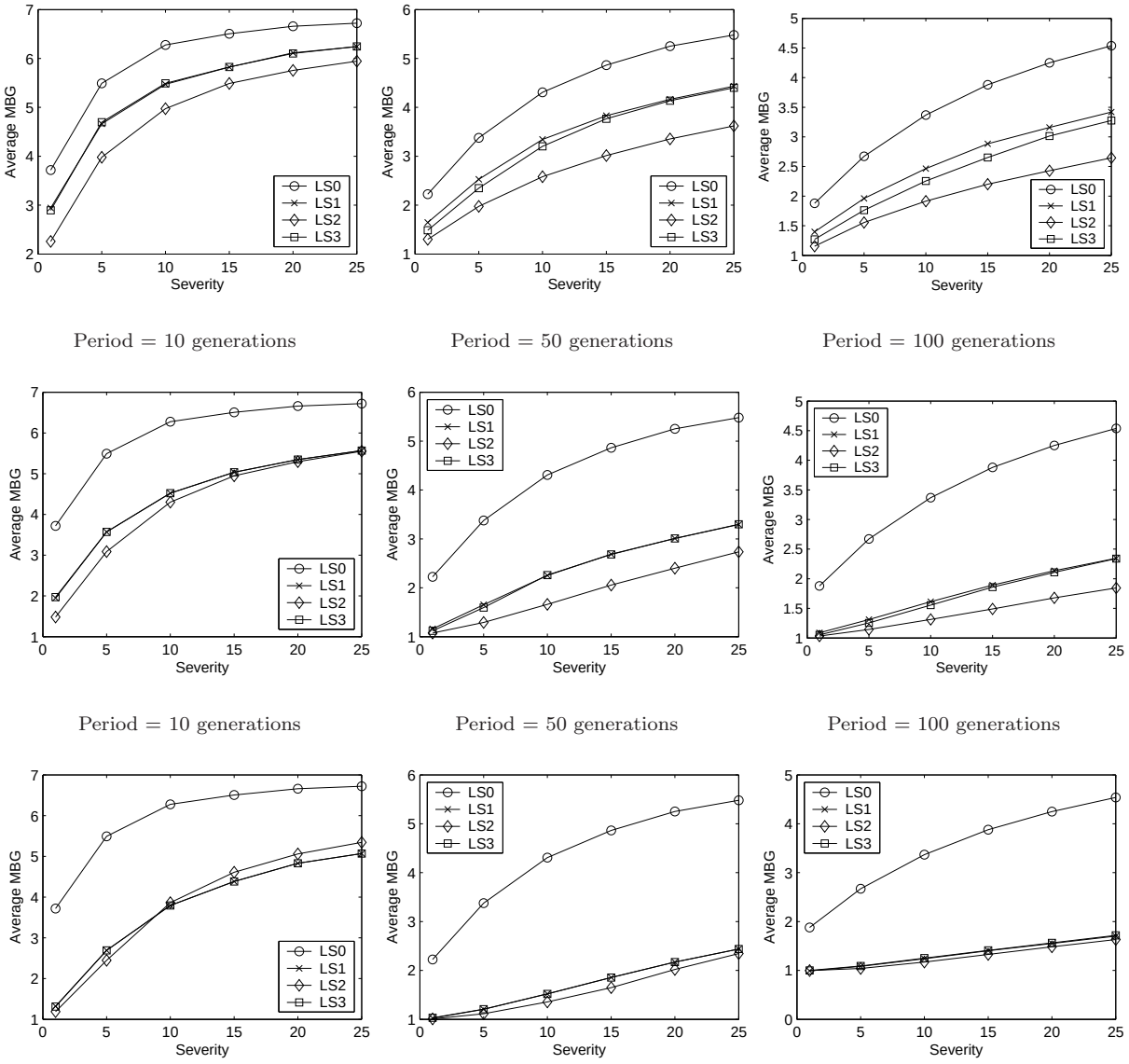
7.5.2 Allocating local search evaluations

Given a certain number of local search evaluations, we would like to determine a strategy to allocate them. To this end, four local search strategies (LS0, LS1, LS2, LS3) are integrated in the ADM model. Thus ADM+LS0 denotes a dynamic solver without local search. ADM+LS1 applies local search to all individuals in the population. ADM+LS2 applies local search to best individual in the population. ADM+LS3 adaptively applies local search, that is, uses LS1 at environmental changes to distribute the search on all individuals in the population, and uses LS2 at static phases to concentrate on further enhancing good individuals.

Experiments are carried out on different dynamics of the k100 problem using the same settings described in the previous sections. The only exception here is the inclusion of the well-known 2-opt heuristic within LS1, LS2 and LS3 strategies. The experiments are repeated using different numbers of allowable local search evaluations. The results summarized in Figure 7.19, which clearly shows that strategies employing local search outperform the pure EA model regardless of the problem dynamics and the allowable number of local search evaluations. Among the three local search strategies, LS2 is the best. That is, with a limited number of iterations it is better to concentrate on fine-tuning the search near the population best rather than "waste" valuable resources on the entire population. Strategies LS1 and LS3 give comparable performance in all slides.

Repeating the experiments on the be52 problem and the p442 problem does not reveal any new trends—only that differences between LS1 and LS3 are more distinguishable in the case of p442 as shown in Figure 7.20.

In the comparative experiments of the next chapter, local search is applied to the best individual in the island only, without pre-requiring local optimality to terminate. However, if a local optimum is encountered, the excess local search budget (unused evaluations) is evenly distributed among other individuals of the population.

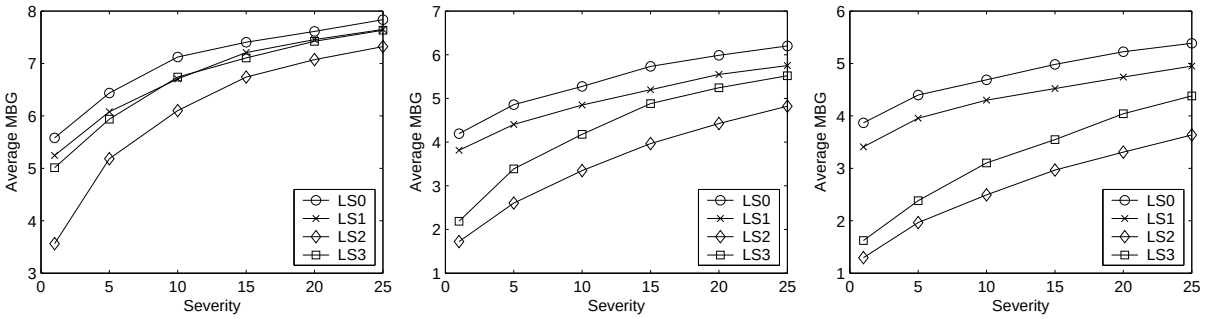


Period = 10 generations | Period = 50 generations | Period = 100 generations

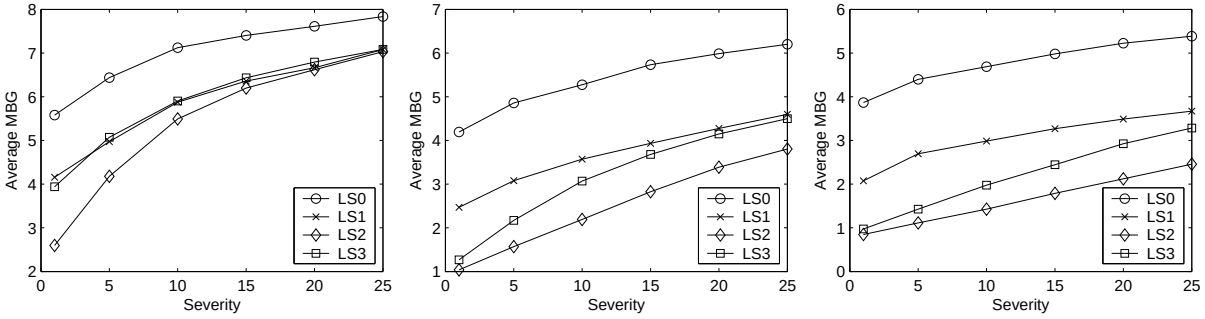
Period = 10 generations | Period = 50 generations | Period = 100 generations

Period = 10 generations | Period = 50 generations | Period = 100 generations

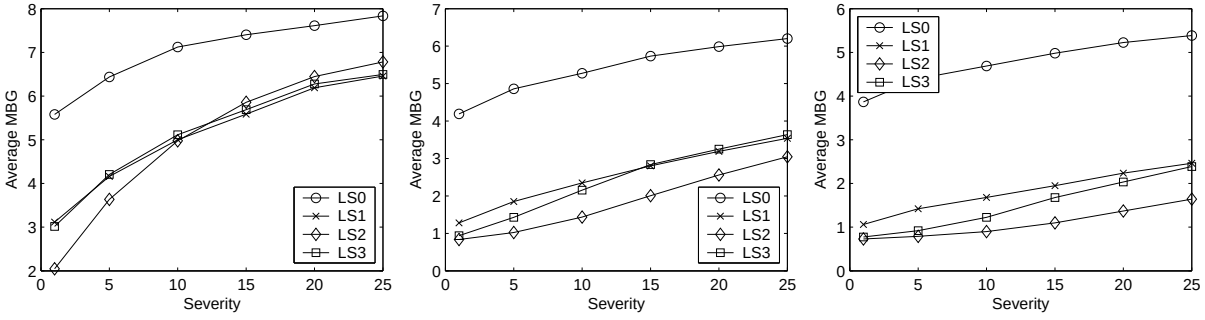
Fig. 7.19: Investigating local search on k100. Local search budget in evaluations: upper raw =50, med raw=500, bottom raw=5000.



Period = 10 generations Period = 50 generations Period = 100 generations



Period = 10 generations Period = 50 generations Period = 100 generations



Period = 10 generations Period = 50 generations Period = 100 generations

Fig. 7.20: Investigating local search on p442. Local search budget in evaluations: upper raw =50, med raw=500, bottom raw=5000. This problem was included because the difference between strategies is more distinct. Nevertheless the trends do not differ from the k100 problem

7.6 Summary

This chapter has introduced several dynamic solvers that use diversity to control the evolutionary search. The ADM model manipulates the evolutionary parameters in response to measured diversity, and the AIM model enhances the ADM model through the use of islands. To further enhance solution quality, the EA models were hybridized with local search.

Efficiency of these models relies to a large degree on the methods they employ to measure diversity, control parameters, migrate solutions, and embed local search. Hence, a substantial amount of investigation and comparative tests was dedicated to finding the proper methods. The final adaptive hybridized island model, AHIM, is expected to work well on many dynamic COPs.

In the next chapter, the performance of the developed algorithms is compared on several dynamic benchmarks that were developed in this thesis.

Chapter 8

Empirical study and analysis

We have saddled algorithmic researchers with the burden of exhibiting faster and better algorithms in each paper, a charge more suited to software houses, while expecting them to advance our knowledge of algorithms at the same time.

Hooker [1996]

8.1 General

The benchmark generation schemes presented in Chapter 5 and the adaptive strategies developed in Chapter 7 are applied to two dynamic combinatorial problems (TSP and FMS). Mathematical formulation, test problems, modes of environmental changes, and results of testing adaptive models on these problems are described in this chapter.

The two evolutionary models (ADM and AIM) and their hybridized counterparts (AHDM and AHIM) are compared against FM, RM, and RIM. In order to reduce the number of pair-wise combinations of models, comparisons are given in two groups. The first group involves comparisons of 'pure' evolutionary models (ADM and AIM against FM, RM and RIM). The second group involves comparisons against hybridized models (ADM and AIM against AHDM and AHIM). This approach also helps in contrasting the results of these models since the quality of results of ADM and AIM is midway between that of FM, RM, and RIM (on one side) and AHDM and AHIM (on the other side).

The evolutionary algorithm code was developed on a DAYTEK CT6730-C27 machine, AMD Athlon XP 2700+ (2.16 GHz). The code was written in C and compiled using GNU/Linux gcc version 3.3.5.

The mean best of generation (MBG), which is based on the objective function rather than on the fitness (see Chapter 4), is used as a basis of comparison for the numerical results presented in this chapter.

8.2 The dynamic travelling salesman problem

Any task of sequencing objects with the objective of minimizing the total cost or time can be regarded as a TSP instance, hence the wide range of application of TSP in science and engineering. Furthermore, it has often been the case that progress on the TSP has led to progress on other combinatorial problems. TSP can simply be stated as: if a travelling

salesman wishes to visit each of a list of cities exactly once and then return to the home city, what is the shortest route the travelling salesman can take?

8.2.1 Mathematical formulation

There are many different formulations for the travelling salesman problem. One common formulation is the integer programming formulation, which is given in [Rardin 1998] as follows:

$$
\begin{aligned}
min \quad & \sum_{i} \sum_{j>i} d_{i,j} x_{i,j} && && (8.1)\\
\text{s.t.} \quad & \sum_{j<i} x_{j,i} + \sum_{j>i} x_{i,j} && = 2 \quad \text{for all } i \\
& \sum_{i \in S} \sum_{j \notin S, j>i} x_{i,j} + \sum_{i \notin S} \sum_{j \in S, j>i} x_{i,j} && \geq 2 \quad \text{for all proper point subsets } S, |S| \geq 3 \\
& x_{i,j} = 0 \text{ or } 1 && \quad \text{for all } i; \quad j > i
\end{aligned}
$$

where $x_{i,j} = 1$ if link (i, j) is part of the solution, and $d_{i,j}$ is the distance from point i to point j.

8.2.2 Dynamic benchmarks

Static problems of sizes comparable to those reported in the literature [Guntsch et al. 2001; Eyckelhof and Snoek 2002; Younes et al. 2003] are used in the comparative experiments of these section. These problems are given in the TSP library [Reinelt 1991] as berlin52, kroA100, and pcb442. In this thesis they are referred to as *be52*, *k100*, and *p442* respectively. Dynamic versions are constructed from these problems in three ways (modes): an edge change mode (ECM), an insert/delete mode (IDM) and a vertex swap mode (VSM).

Edge change mode This mode reflects one of the real-world scenarios, a traffic jam. Here, the distance between the cities is viewed as a time period or cost that may

change over time, hence the introduction and the removal of a traffic jam, respectively, can be simulated by the increase or decrease in the distance between cities. The change step of the traffic jam is the increase in the cost of a single edge. The strategy is as follows: If the edge cost is to be increased then that edge should be selected from the best tour. However, if the cost were to be reduced then the selected edge should not be part of the best tour.

The BG starts from one known instance and solves it to find the best or the near best tour. Then an edge is selected randomly from the best tour, and its cost is increased by a user defined factor creating a new instance which will likely have a different best tour.

Insert/delete mode IDM reflects the addition and deletion of new assignments (cities). This mode works in a manner similar to the ECM mode. The step of the change in this mode is the addition or the deletion of a single city. This mode generates the most difficult problems to solve dynamically since they require variable chromosome length to reflect the increase or decrease in the number of cities from one instance to the next.

Vertex swap mode VSM is another way to create a dynamic TSP by interchanging city locations. It offers a simple, quick and easy way to test and analyze the dynamic algorithm. The locations of two randomly selected cities are interchanged; this does not change the length of the optimal tour but does change the solution (this is analogous to shifting the independent variable(s) of a continuous function by a predetermined amount). The change step (the smallest possible change) in this mode is an interchange of costs between a pair of cities; this can be very large in comparison with the change steps of the previous two modes.

In the experiments conducted, each benchmark problem is created from an initial sequence of 1000 static problems inter-separated by single elementary steps. Depending on

the specified severity, a number of intermediate static problems will be skipped to construct one test problem.

Each sequence of static problems is translated into 21 dynamic test problems by combining seven degrees of severity (1, 5, 10, 15, 20, 25 steps per shift, and random) and three periods of change (500, 2500, and 5000 evaluations per shift, which correspond to 10, 50, and 100 generations per shift based on a population of 50 individuals).

8.2.3 Algorithm settings

In all tested models, the underlying GA is generational with tournament selection in which selection pressure can be altered by changing a selection probability parameter. Edge crossover [Whitley et al. 1991] is used throughout. The mutation operator is a pair-wise node swap. It sweeps down the list of bits in the chromosome, swapping each with a randomly selected bit if a probability test is passed. That is, for a problem of size l and a mutation rate of μ the expected number of swaps on an individual is μl. This operator produces little change in the individual, since no more than four edges are replaced in each swap. Nevertheless, more drastic changes can be applied by simply increasing the rate of mutation.

A population of 50 individuals is used throughout. The FM, RM and RIM models use a crossover rate of 0.9 and a selection probability of 1.0. These values are determined by tuning on the be52 problem in the VSM mode. A mutation rate of $\frac{1}{4l}$ is used. This value corresponds to the commonly used rate of mutation [Gen and Cheng 1999; Mitchell 1996; Reeves and Rowe 2002] (inverse of the chromosome length) and to the number of edges (four) replaced in each swap. For the ADM, AIM, AHDM, and AHIM models, the previous values represent one limit on the corresponding operator. For these models, the other limit is 1.0, 0.9, and $\frac{1}{2l}$ on crossover, selection and mutation respectively.

Individuals in the population are assigned to five islands in the AIM and AHIM models. The well-known 2-opt heuristic is used in the hybridized models (AHDM and AHIM) for

local search, with the number of local search evaluations per generation limited to 50.

8.2.4 Results

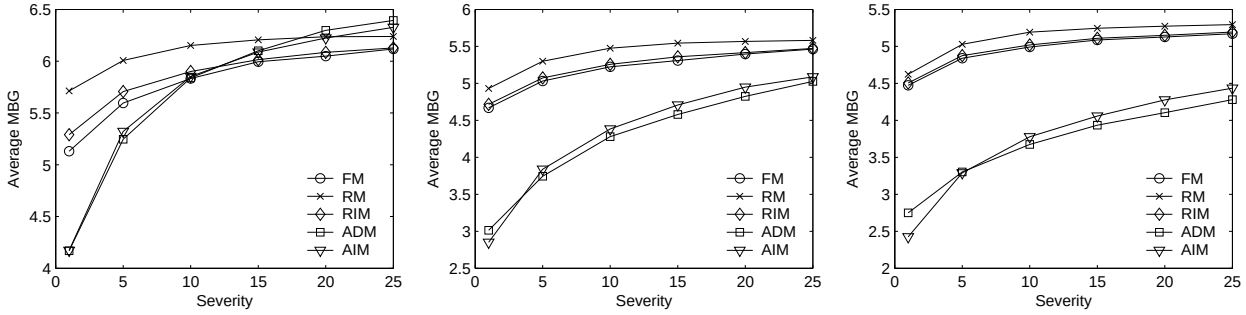
The results of experiments on the be52, k100, and p442 problems in the three modes of environmental change are given in Appendix A. In this section, we discuss the results of the k100 problem, which are similar to those of the other two problems.

Comparison of evolutionary models

Experimental results on the dynamic k100 problem in the VSM mode under three different periods of change are summarized in Figure 8.1, where the mean best of generation (averaged over ten runs) is plotted against severity of change. The ADM and AIM models outperform the other models in almost all cases. The other three models give comparable results to each other in general, with differences tending to decrease as the severity of change increases. Only when the change severity is 10 steps per shift or more, may the other models give slightly better performance than ADM and AIM. Keeping in mind that in this 100 vertex problem, a severity of 10 in the VSM mode amounts to changing (4×10) edges; that is, about 40% of the edges in an individual are replaced, which constitutes a substantial amount of change. As we are interested in small environmental changes (which are the norm in practice), we can safely conclude that the experiments attest to the superiority of the ADM and AIM over the other three models in the range of change of interest.

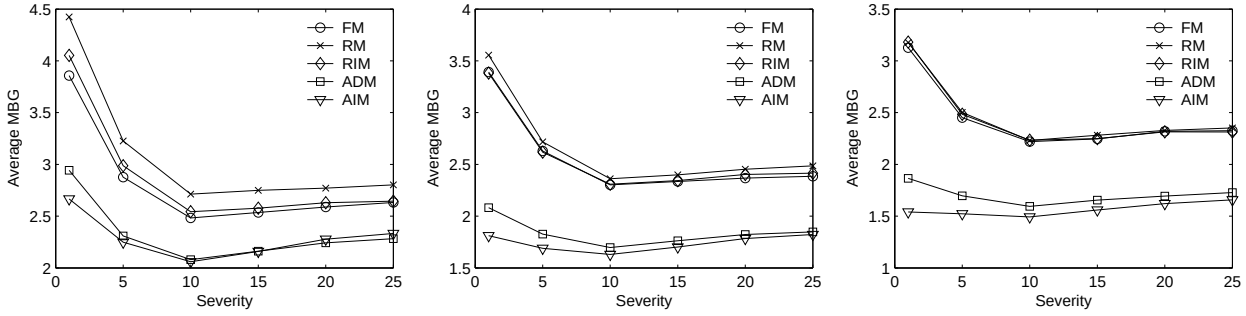
Running the benchmark generator in either the ECM mode or the IDM mode gives similar results as illustrated in Figure 8.2 and Figure 8.3 respectively. It can be seen that ADM and AIM outperform the other models in most considered dynamics.

The RM model produces the worst results in all conducted experiments (even though this model has been modified to retain the best solution in the hope of obtaining better



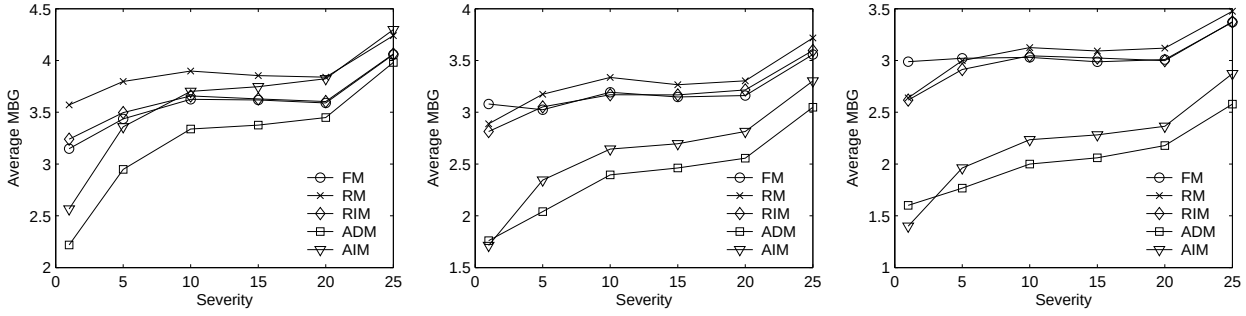
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. 8.1: Comparison of evolutionary models (k100_VSM)



Period = 10 generations Period = 50 generations Period = 100 generations

Fig. 8.2: Comparison of evolutionary models (k100_ECM)



Period = 10 generations Period = 50 generations Period = 100 generations

Fig. 8.3: Comparison of evolutionary models (k100_IDM)

results than those obtained in Chapter 6).

It is not easy to definitely settle on which of the two models (ADM and AIM) is the best, since both give very comparable results in almost all cases. However, where more than one processor can be used, AIM is the best of the two models since it can be easily parallelized by allocating different islands to different processors and consequently reducing computation time drastically.

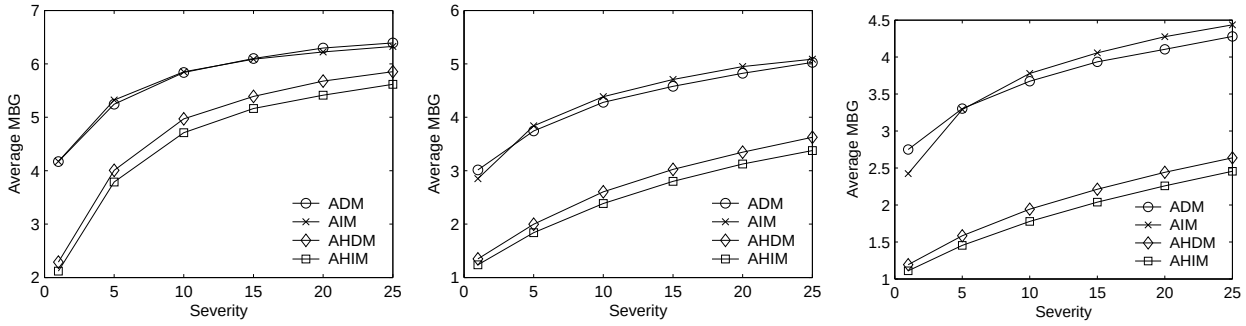
Comparison of hybridized models

Results of comparisons of AIM and ADM against their hybridized counterparts (AHIM and AHDM) are shown in Figure 8.4, 8.5, and 8.6, where average MBG is plotted against different values of severity. First, we observe that in general all hybridized models are by far better than pure evolutionary models in the three modes of environmental change and over all the considered dynamics.

As with the pure evolutionary models, the performance of AHIM is comparable to that of AHDM. Therefore, the same argument of the previous section can be used to conclude that AHIM is better than AHDM if several processors are used.

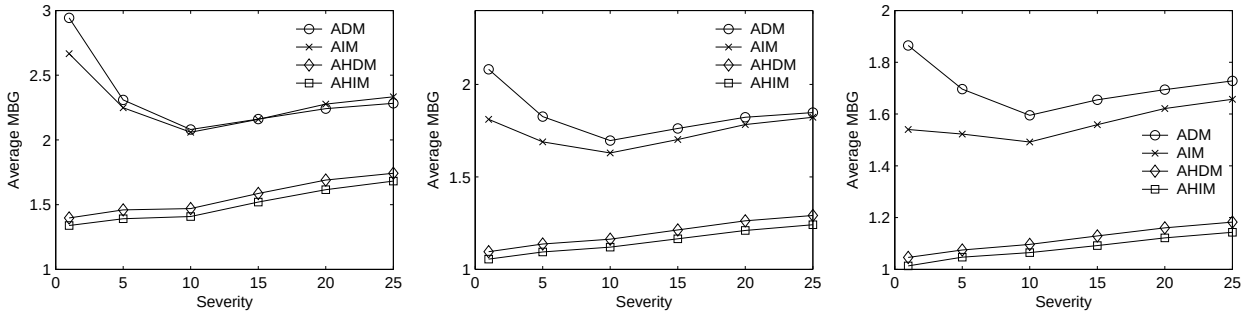
Individual model results

More details on the conducted experiments are given in Table 8.1 for the ADM model. The number of evaluations needed to reach the best solution after an environmental change is averaged for all instances of the problem. The adaptation columns give the average and the standard deviation (over the conducted runs) of the number of evaluations per instance, including those of the local search when applicable. The MBG columns give the worst, the average, the best, and the standard deviation of the mean best of generation over the conducted runs. The last column gives the mean CPU time per instance (from an environmental change till the discovery of the best solution); $mean\ CPU = \frac{T}{R} \times \frac{E}{E_t}$, where T is the total CPU time, R is the number of conducted runs, E is the average number



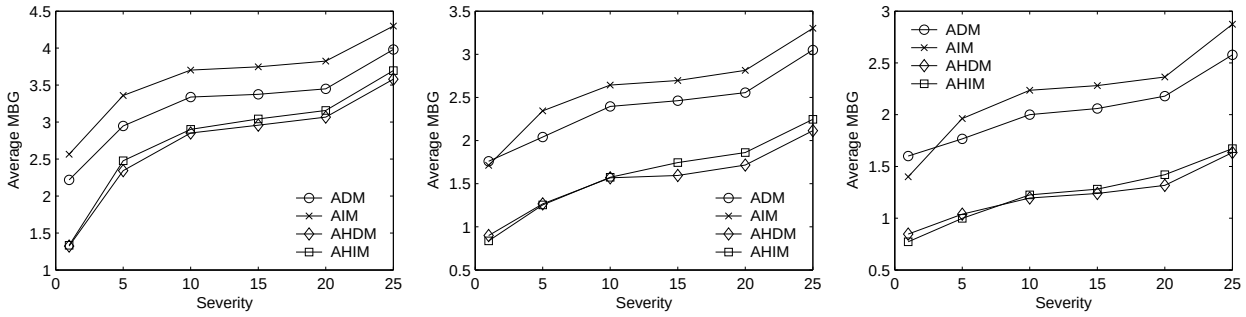
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. 8.4: Comparison of hybridized models (k100_VSM)



Period = 10 generations Period = 50 generations Period = 100 generations

Fig. 8.5: Comparison of hybridized models (k100_ECM)



Period = 10 generations Period = 50 generations Period = 100 generations

Fig. 8.6: Comparison of hybridized models (k100_IDM)

Table 8.1: Results of ADM (k100_SW)

		Adaptation			MBG			CPU
Per.	Sev.	avg	std	worst	avg	best	std	(sec)
	1	312.	12.79	4.2544	4.1696	4.0287	0.08	0.02
	5	363.	7.85	5.3227	5.2459	5.1925	0.04	0.02
	10	353.	7.36	5.9298	5.8394	5.7440	0.06	0.02
10	15	339.	10.07	6.1893	6.1003	6.0180	0.06	0.02
	20	332.	7.33	6.3907	6.2976	6.2411	0.05	0.02
	25	334.	11.89	6.5481	6.3933	6.2391	0.08	0.02
	rnd	351.	10.12	5.5935	5.5343	5.4797	0.04	0.02
	1	1711.	53.35	3.0732	3.0133	2.9342	0.05	0.08
	5	1918.	43.19	3.7874	3.7452	3.6892	0.03	0.09
	10	1863.	36.44	4.3391	4.2801	4.2505	0.03	0.10
50	15	1814.	22.38	4.6390	4.5802	4.5202	0.04	0.10
	20	1783.	32.59	4.8829	4.8245	4.8007	0.03	0.11
	25	1733.	22.03	5.0818	5.0274	4.9780	0.04	0.11
	rnd	1882.	21.65	4.0343	4.0005	3.9567	0.03	0.10
	1	3614.	89.30	2.8018	2.7503	2.7136	0.03	0.17
	5	3998.	66.90	3.3303	3.3027	3.2719	0.02	0.20
	10	3910.	57.48	3.7088	3.6736	3.6406	0.02	0.20
100	15	3829.	38.00	3.9868	3.9355	3.8898	0.03	0.20
	20	3791.	66.66	4.1625	4.1043	4.0576	0.03	0.21
	25	3699.	44.49	4.3260	4.2796	4.2371	0.03	0.21
	rnd	3935.	49.00	3.5032	3.4738	3.4230	0.02	0.19

Table 8.2: Results of AHDM (k100_SW)

		Adaptation			MBG			CPU
Per.	Sev.	avg	std	worst	avg	best	std	(sec)
	1	893.	5.62	2.3349	2.2873	2.2261	0.03	0.02
	5	878.	5.63	4.0525	4.0075	3.9159	0.04	0.02
	10	851.	5.12	5.0538	4.9714	4.9032	0.04	0.02
10	15	836.	7.94	5.4601	5.3907	5.3143	0.05	0.02
	20	824.	7.37	5.7285	5.6783	5.5756	0.04	0.02
	25	812.	8.53	5.9179	5.8538	5.7898	0.05	0.02
	rnd	866.	3.45	4.5445	4.5038	4.4536	0.03	0.01
	1	4462.	25.81	1.3611	1.3482	1.3259	0.01	0.10
	5	4442.	13.54	2.0160	1.9973	1.9785	0.01	0.11
	10	4371.	11.74	2.6233	2.6044	2.5881	0.01	0.11
50	15	4353.	19.54	3.0398	3.0252	3.0139	0.01	0.11
	20	4325.	17.30	3.3710	3.3453	3.2880	0.03	0.11
	25	4308.	25.93	3.6641	3.6254	3.5791	0.03	0.11
	rnd	4408.	11.12	2.3210	2.2894	2.2554	0.02	0.10
	1	8605.	93.62	1.2079	1.1925	1.1767	0.01	0.20
	5	8982.	41.58	1.6010	1.5833	1.5717	0.01	0.22
	10	8841.	27.45	1.9616	1.9429	1.9190	0.01	0.21
100	15	8735.	22.78	2.2282	2.2129	2.1939	0.01	0.22
	20	8686.	30.71	2.4622	2.4423	2.4218	0.01	0.22
	25	8630.	35.90	2.6636	2.6367	2.6199	0.01	0.22
	rnd	8921.	34.49	1.7668	1.7484	1.7342	0.01	0.21

Table 8.3: Results of AIM (k100_SW)

		Adaptation		MBG				CPU
Per.	Sev.	avg	std	worst	avg	best	std	(sec)
	1	408.	21.44	4.2992	4.1726	4.0818	0.08	0.02
	5	474.	12.20	5.3547	5.3220	5.2555	0.03	0.02
	10	481.	8.12	5.8915	5.8523	5.8206	0.02	0.02
10	15	486.	12.35	6.1579	6.0884	5.9828	0.06	0.02
	20	480.	16.88	6.2926	6.2238	6.1508	0.04	0.02
	25	485.	9.09	6.4055	6.3264	6.2265	0.05	0.02
	rnd	480.	13.22	5.6800	5.5825	5.5110	0.04	0.02
	1	2349.	49.00	2.9043	2.8517	2.8064	0.03	0.10
	5	2632.	21.92	3.8977	3.8398	3.8178	0.03	0.11
	10	2674.	20.50	4.4105	4.3847	4.3541	0.02	0.11
50	15	2691.	23.67	4.7632	4.7081	4.6559	0.04	0.10
	20	2672.	14.27	5.0191	4.9500	4.8684	0.04	0.11
	25	2670.	26.18	5.1663	5.0885	4.9842	0.05	0.11
	rnd	2641.	35.28	4.1381	4.0995	4.0561	0.03	0.11
	1	4802.	60.15	2.4715	2.4250	2.3379	0.05	0.19
	5	5341.	35.45	3.3482	3.2904	3.2519	0.04	0.21
	10	5415.	30.34	3.8214	3.7780	3.7408	0.02	0.22
100	15	5427.	41.46	4.1028	4.0575	4.0008	0.03	0.22
	20	5440.	40.08	4.3082	4.2763	4.2107	0.03	0.22
	25	5426.	35.96	4.4778	4.4356	4.3971	0.03	0.22
	rnd	5369.	25.16	3.5473	3.5075	3.4733	0.02	0.22

Table 8.4: Results of AHIM (k100_SW)

		Adaptation		MBG				CPU
Per.	Sev.	avg	std	worst	avg	best	std	(sec)
	1	2684.	14.85	2.1390	2.1185	2.0809	0.02	0.02
	5	2710.	6.55	3.8330	3.7872	3.7361	0.03	0.02
	10	2698.	15.60	4.7669	4.7126	4.6497	0.03	0.02
10	15	2687.	15.03	5.2039	5.1646	5.1190	0.03	0.02
	20	2677.	20.91	5.4836	5.4134	5.3489	0.05	0.03
	25	2693.	24.25	5.6729	5.6164	5.5372	0.04	0.03
	rnd	2705.	11.02	4.2872	4.2485	4.2051	0.03	0.02
	1	14125.	174.61	1.2454	1.2370	1.2320	0.00	0.11
	5	15022.	34.68	1.8466	1.8390	1.8283	0.01	0.12
	10	15076.	18.75	2.3985	2.3856	2.3677	0.01	0.11
50	15	15071.	15.50	2.8249	2.7994	2.7693	0.02	0.12
	20	15057.	10.54	3.1699	3.1255	3.0955	0.03	0.12
	25	15026.	21.26	3.4111	3.3778	3.3336	0.02	0.12
	rnd	15014.	22.89	2.1173	2.1010	2.0805	0.01	0.12
	1	26499.	460.55	1.1218	1.1105	1.0996	0.01	0.20
	5	30083.	63.61	1.4671	1.4528	1.4454	0.01	0.23
	10	30356.	65.87	1.7877	1.7781	1.7674	0.01	0.24
100	15	30433.	42.68	2.0628	2.0377	2.0223	0.01	0.23
	20	30444.	41.34	2.2736	2.2591	2.2436	0.01	0.24
	25	30443.	41.28	2.4855	2.4556	2.4327	0.02	0.24
	rnd	30017.	133.77	1.6091	1.6015	1.5939	0.01	0.23

of evaluations to best per instance, and E_t is the average total number of evaluations per run.

An interesting observation here is that there are no significant differences between values of the worst, average and best of runs. This is also evident in the small values of standard deviation. The obvious explanation is that the number of instances involved is large enough to give a reasonably accurate estimate of the algorithm's performance with a reduced number of runs.

Similar details can be found in Table 8.2 for AHDM, in Table 8.3 for AIM and in Table 8.4 for AHIM. Examining these tables individually does not reveal new trends in addition to those discussed in reference to Table 8.1. Pair-wise comparisons, however, show that the instance runtime (CPU column) does not change significantly when local search is embedded in the evolutionary model. The efficiency of local search is a result of the limitations imposed on the number of local search evaluations, which when applied effectively (to the population best) enhance the solution quality without significantly increasing runtime, as can be seen from comparing Table 8.1 to Table 8.2, and Table 8.3 to Table 8.4.

Statistical Analysis

Statistical results reported in this section are obtained using a significance level of 5% to construct 95% confidence intervals on the difference in the mean best of generation (see Section 4.5.3 for details on how the statistical tests are carried out). Table 8.5, 8.6, and 8.7 show the results of multiple post ANOVA comparison test for the three modes of change (respectively, ECM, IDM, and VSM). Each table covers 21 combinations of problem dynamics (three periods of change and seven levels of severity of change including one level with a random severity).

Table 8.8, 8.9, and 8.10 show the results of multiple post ANOVA comparison test on the hybridized models for the three modes of change and the 21 combinations of dynamics.

The entries in these tables are interpreted as follows. An entry of 1 signifies that the

Table 8.5: Multiple comparison test of evolutionary models (k100-VSM)

period	10							100							1000						
severity	1	5	10	15	20	25	r	1	5	10	15	20	25	r	1	5	10	15	20	25	r
FM−RM	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	0	-1	0	-1	-1	-1	-1	-1	-1
FM−RIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−ADM	1	1	0	0	-1	-1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
FM−AIM	1	1	0	0	-1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−RIM	1	1	1	1	0	0	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1
RM−ADM	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−AIM	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−ADM	1	1	0	0	-1	-1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−AIM	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ADM−AIM	0	0	0	0	0	0	0	0	-1	0	-1	0	0	0	1	0	-1	-1	-1	-1	0

Table 8.6: Multiple comparison test of evolutionary models (k100-ECM)

period	10							100							1000						
severity	1	5	10	15	20	25	r	1	5	10	15	20	25	r	1	5	10	15	20	25	r
FM−RM	-1	-1	-1	-1	-1	-1	-1	0	0	0	0	0	-1	0	0	0	0	0	0	0	0
FM−RIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−ADM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
FM−AIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−RIM	1	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RM−ADM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−AIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−ADM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−AIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ADM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	1

Table 8.7: Multiple comparison test of evolutionary models (k100-IDM)

period	10							100							1000						
severity	1	5	10	15	20	25	r	1	5	10	15	20	25	r	1	5	10	15	20	25	r
FM−RM	-1	-1	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	0
FM−RIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
FM−ADM	1	1	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
FM−AIM	1	0	0	0	0	0	0	1	1	1	1	1	0	1	1	1	1	1	1	0	1
RM−RIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RM−ADM	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−AIM	1	1	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−ADM	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−AIM	1	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ADM−AIM	0	-1	0	-1	-1	0	-1	0	-1	0	0	-1	0	0	0	0	0	0	0	0	0

Table 8.8: Multiple comparison test of hybridized models (k100-VSM)

period	10							100							1000						
severity	1	5	10	15	20	25	r	1	5	10	15	20	25	r	1	5	10	15	20	25	r
ADM−AIM	0	0	0	0	0	0	0	1	-1	-1	-1	0	0	-1	1	0	-1	-1	-1	-1	0
ADM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ADM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AHDM−AHIM	0	1	1	1	1	0	1	0	1	1	1	1	1	1	0	1	1	1	1	1	1

Table 8.9: Multiple comparison test of hybridized models (k100-ECM)

period	10							100							1000						
severity	1	5	10	15	20	25	r	1	5	10	15	20	25	r	1	5	10	15	20	25	r
ADM−AIM	1	0	0	0	0	0	0	1	1	1	1	0	0	1	1	1	1	1	1	1	1
ADM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ADM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AHDM−AHIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 8.10: Multiple comparison test of hybridized models (k100-IDM)

period	10							100							1000						
severity	1	5	10	15	20	25	r	1	5	10	15	20	25	r	1	5	10	15	20	25	r
ADM−AIM	-1	0	0	-1	-1	0	-1	0	-1	0	0	0	0	0	0	0	0	0	0	0	0
ADM−AHDM	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ADM−AHIM	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AHDM−AHIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

confidence interval for the difference in performance measures of the corresponding pair consists entirely of positive values, which indicates that the first model is inferior to the second model. Conversely, an entry of -1 signifies that the confidence interval for the corresponding pair consists entirely of negative values, which indicates that the first model is superior to the second one. An entry of 0 indicates that there is no significant difference between the two models.

Statistical analysis confirms the arguments made on the graphical comparisons in the previous section. As can be seen in Table 8.5, 8.7, and 8.6, there are significant differences between the performance of the adaptive models (ADM and AIM) and the other three models (FM, RM, and RIM), while there is no significant difference between ADM and AIM. As well, Table 8.8, 8.10, and 8.9 emphasize the superiority of the hybrid models over the pure evolutionary models and indicate the relatively similar performance of AHIM and AHDM.

8.3 Dynamic flexible manufacturing systems

The large number of combinatorial problems associated with manufacturing optimization [Dimopoulos and Zalzala 2000] is behind the growth in the use of intelligent techniques, such as flexible manufacturing systems, in the manufacturing field during the last decade. A flexible manufacturing systems (FMS) produces a variety of part types that are flexibly routed through machines instead of the conventional straight-line routing [Chen and Ho 2002]. The flexibility associated with this system enables it to cope with unforeseen events such as machine failures, erratic demands, and changes in product mix.

Typically, an FMS is a production system consisting of a heterogeneous group of numerically controlled machines (machines, robots, and computers) which are connected through an automated guided vehicle system. Each machine can perform a specific set of operations that may intersect with operation sets of the other machines. Production

planning and scheduling is more complicated in an FMS than it is in traditional manufacturing [Wang et al. 2005]. One source of additional complexities is associated with machine-operation versatility, since each machine can perform different operations and an operation can be performed on different alternative machines. Another source of complexity is in the material handling systems. In scheduling these systems, a number of additional decision points that affect the overall performance of the FMS are created. A third source of complexity is associated with unexpected events, such as machine breakdown, change of demand, or introduction of new products. A fundamental goal that is gaining importance is the ability to handle such unforseen events. To illustrate the kind of FMS we are focusing on, we give the following example.

Example

A simple flexible manufacturing system consists of three machines, M_1, M_2 and M_3. The three respective sets of operations for these machines are $\{O_1,\ O_6\}$, $\{O_1,\ O_2,\ O_5\}$, and $\{O_4,O_6\}$, where O_i denotes operation i. This system is to be used to process three part types P_1, P_2, and P_3, each of which requires a set of operations, respectively, given as $\{O_1,\ O_4,\ O_6\}$, $\{O_1,\ O_2,\ O_5,\ O_6\}$, and $\{O_4,\ O_6\}$.

Choice (a) For part P_1: $(O_1 \mapsto M_2,\ O_4 \mapsto M_3,\ O_6 \mapsto M_3)$; i.e, assign machine M_2 to process O_1, and assign M_3 to process O_4 and O_6. For part P_2: $(O_1 \mapsto M_1,\ O_2 \mapsto M_2,\ O_5 \mapsto M_2,\ O_6 \mapsto M_1)$; i.e assign machine M_1 to process O_1 and O_6, and assign M_2 to process O_2 and O_5. for part P_3: $(O_4 \mapsto M_3, O_6 \mapsto M_3)$; i.e, assign machine M_3 to process O_4 and O_6.

Choice (b) For part P_1: $(O_1 \mapsto M_2,\ O_4 \mapsto M_3,\ O_6 \mapsto M_1)$. For part P_2: $(O_1 \mapsto M_1,\ O_2 \mapsto M_2,\ O_5 \mapsto M_2,\ O_6 \mapsto M_3)$. for part P_3: $(O_4 \mapsto M_3,\ O_6 \mapsto M_1)$.

By comparing both choices, one notices that the first solution is biased towards minimizing transfer of parts between machines. On the other hand the second solution is biased towards balancing the operations on the machines. However, we need to consider both objectives at the same time, which may not be easy since they are conflicting.

The complexity and the specifics of the problem require revising several components of the conventional evolutionary algorithm to obtain an effective implementation on the FMS problem. In particular, we need to devise problem-oriented methods for encoding solutions, crossover, fitness assignment, and constraint handling. Appendix C contains detailed descriptions of the genetic operators and the algorithmic approach. We restrict discussion in the following sections to those details necessary to follow the approach for the dynamic problem.

8.3.1 Mathematical formulation

The assignment problem considered in this section is given in Younes et al. [2002] using the following notations:

i,l : machine index (i,l = 1,2,3,...,n_m)

j : part index (j = 1,2,3,...,n_p)

$\hat{k}_j$: is processing choice for part j (j = 1,2,3,....,n_p)

k_j : is the number of processing choices of P_j

$n_{ji\hat{k}_j}$: is the number of necessary operations required by Pj on M_i in processing choice $\hat{k}_j$, $1 \leq \hat{k}_j \leq k_j$ $t_{ji\hat{k}_j}$: is the work-load of machine M_i to process part P_j in processing choice $\hat{k}_j$.

$$x_{ji\hat{k}_j} = \begin{cases} 1 & \text{if } P_j \text{ requires } M_i \text{ in processing choice } \hat{k}_j \\ 0 & \text{otherwise} \end{cases}$$

$$q_{j\hat{k}_j} = \begin{cases} 1 & \text{if processing choice } \hat{k}_j \text{ is selected for part j} \\ 0 & \text{otherwise} \end{cases}$$

Using this notation, the three objective functions of the problem (f_1, f_2, and f_3) are given as follows:

1. Minimization of part transfer (by minimizing the number of machines required to process each part):
$$f_1 = min_{\hat{k}_j} \sum_{i=1}^{n_m} q_{j\hat{k}_j} x_{ji\hat{k}_j}, \forall_j \tag{8.2}$$

2. Load Balancing by minimizing the cardinality distance between the workload of any pair of machines:
$$f_2 = min_{\hat{k}_j} \sum_{j=1}^{n_p} q_{j\hat{k}_j} \sum_{i=1}^{n_m} \sum_{l=(i+1)}^{n_m} |x_{ji\hat{k}_j} t_{ji\hat{k}_j} - x_{jl\hat{k}_j} t_{jl\hat{k}_j}| \tag{8.3}$$

3. Minimization of the number of necessary operations required from each machine over the possible processing choices:
$$f_3 = min_{\hat{k}_j} \sum_{i=1}^{n_m} q_{j\hat{k}_j} x_{ji\hat{k}_j} n_{ji\hat{k}_j}, \forall_j \tag{8.4}$$

An overall multi-objective mathematical model of FMS can then be formulated as follows:

$$solve for \quad f_1, f_2, f_3$$

s.t.

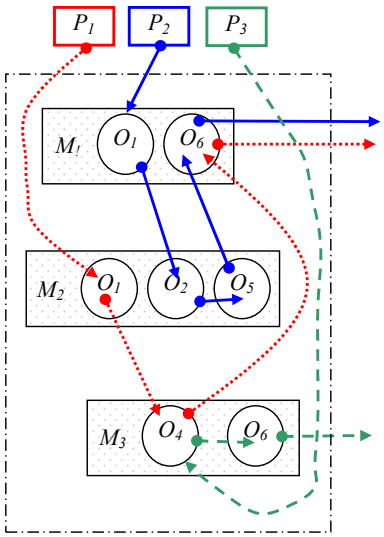
$$\sum_{\hat{k}_j=1}^{k_j} q_{j\hat{k}_j} = 1$$

$$x_{ji\hat{k}_j} \in \{0,1\};\ \ q_{j\hat{k}_j} \in \{0,1\};\ \ n_{ji\hat{k}_j} \geq 1;\ \ t_{ji\hat{k}_j} \geq 0$$

The above model is the basis of the scheme of solution encoding and evaluation described in the next section.

8.3.2 Solution representation and evaluation

An individual solution is represented by a series of operations for all parts involved. Each gene in the chromosome represents a machine type that can possibly process a specific operation. Figure 8.7 illustrates a chromosome representation of a possible solution to the example given in Section 8.3. The advantages of this representation scheme are the simplicity and the capability of undergoing standard operators without producing infeasible solutions (as long as parent solutions are feasible).

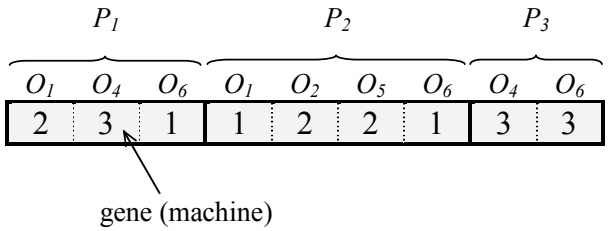


(a)

P_1: $O_1 \mapsto M_2,\ O_4 \mapsto M_3,\ O_6 \mapsto M_1$

P_2: $O_1 \mapsto M_1,\ O_2 \mapsto M_2,\ O_5 \mapsto M_2,\ O_6 \mapsto M_1$

P_3: $O_4 \mapsto M_3, O_6 \mapsto M_3$



(b)

Fig. 8.7: Chromosome representation. A schematic diagram of the possible choice of part routing in (a) is represented by the chromosome in (b)

The effectiveness of a solution has to be measured in terms of minimizing part transfer, balancing the workload of the machines and minimizing the number of different operations per machine. These objectives are conflicting. A solution that minimizes one objective function can be of poor quality for the other two objective functions. In what follows, normalized formulas for the three objective functions are given in order to enable a weighted sum approach for the evaluation of solutions.

Part transfer If γ_j is the number of different machines assigned to part P_j, then part transfer for P_j is $(\gamma_j - 1)$. An upper bound on part transfer is $(\phi_j - 1)$ where ϕ_j is the number of operations for part P_j. Accordingly, a normalized part transfer function for all the parts can be given as:

$$F_1 = \frac{\sum_{j=1}^{n_p}(\gamma_j - 1)}{\sum_{j=1}^{n_p}(\phi_j - 1)} \tag{8.5}$$

where n_p is the total number of parts

Load balance Machine imbalance can be measured in terms of the sum of the squared deviations ssd from the average number of operations.

$$ssd = \sum_{i=1}^{n_m}(\lambda_i - \frac{\sum_{i=1}^{n_m}\lambda_i}{n_m})^2$$

where λ_i is the number of operations performed by machine M_i and n_m is the total number of machines. An upper bound for the sum of the squared deviations can be computed by assuming a case in which one machine does all the operations and the remaining machines are idle as follows:

$$\begin{aligned} ssd_{max} &= (n_p - \frac{n_p}{n_m})^2 + (n_m - 1)(0 - \frac{n_p}{n_m})^2 \\ &= \frac{n-1}{n} \cdot n_p^2 \end{aligned}$$

Accordingly, a normalized measure of machine imbalance can be given by

$$F_2 = \frac{ssd}{ssd_{max}} \tag{8.6}$$

Machine operations This objective can be expressed in terms of the sum Λ of the number of operations allocated to each machine; i.e., $\Lambda = \sum_{i=1}^{n_m}(\lambda_i)$

An upper bound Λ_{max} on the machine operations can be computed by considering the worst case, in which each machine performs all operations, $\Lambda_{max} = n_m \cdot n_p$

For this objective function, a non-zero lower bound Λ_{min} can also be computed for the case when no operation overlap exists between machines; i.e, $\Lambda_{min} = n_p$.

Accordingly, a normalized measure of machine operations can be given by

$$F_3 = \frac{\Lambda - \Lambda_{min}}{\Lambda_{max} - \Lambda_{min}} = \frac{\sum_{i=1}^{n_m}(\lambda_i) - n_p}{n_m n_p - n_p} \tag{8.7}$$

Based on these normalizations, the proposed fitness assignment uses a weighted cost function C, which can be given for an individual v as follows:

$$C(v) = w_1 F_1 + w_2 F_2 + w_3 F_3 \tag{8.8}$$

where w_1, w_2, and w_3 are the weights assigned to the three objective functions respectively, and $w_1 + w_2 + w_3 = 1$.

By varying the weights in the cost function (8.8), one can plot different solutions in the objective function space. Figure 8.8 illustrates the conflicting behavior of the three functions on the rnd1 problem (11 machines, 20 parts, and 9 operations).

In the current implementation, the three objectives are lexicographically ordered, such that minimizing part transfer has the highest priority and minimizing the number of operations has the lowest priority.

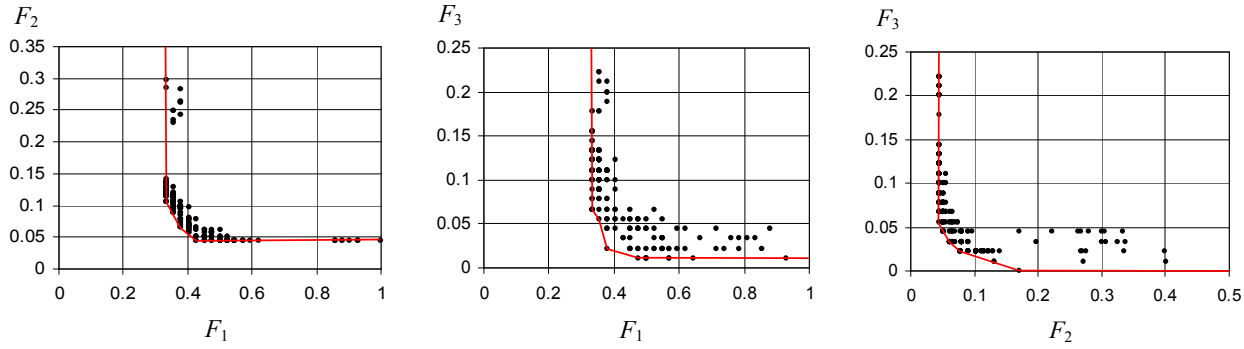


Fig. 8.8: Solutions in the criterion space

8.3.3 Dynamic benchmarks

Four instances of sizes comparable to those used in the literature [Chen and Ho 2002; Younes et al. 2002] are used in the comparative experiments of these section.

Three of these instances (20 agents, 200 jobs), (20 agents, 100 jobs) and (10 agents, 100 jobs) were used in Chu and Beasley [1997]. The data describing these problems can be found in the *gapd* file in the OR-library [Beasley 1990]. In this thesis they are referred to as *gap1*, *gap2*, and *gap3* respectively. As described in Chen and Ho [2002], agents are considered as machines, jobs are considered as operations, and each part is assumed to consist of five operations. In these instances, a machine is assumed capable of performing all the required operations. However, in general machines may have limited capabilities; that is, each machine can perform a specific set of operations that may or may not overlap with those of the other machines. To enable this feature, a machine-operation incidence matrix is generated for each instance as follows: If the cost of allocating a job to an agent is below a certain level, the corresponding entry in the new incidence matrix is equal to one to indicate that the machine is capable of performing the corresponding operation. Alternatively, if the cost is large, the corresponding entry in the incidence matrix is zero to indicate that the job is not applicable to the machine. The final lists that associate

parts with operations and machines with operations are used to construct the dynamic problems. These lists are provided in Appendix D.

The forth problem instance is randomly generated. It was specifically designed and used to test FMS systems with overlapping capabilities in Younes et al. [2002]. This instance consists of 11 machines, 20 parts, and 9 operations. In this thesis, it will be referred to as *rnd1*.

In terms of the number of part operations (chromosome length) and the number of machines (alleles), the dimensions of these problems are 200×20, 100×20, 100×10, and 62×11 for gap1, gap2, gap3, and rnd1 respectively.

Dynamic problems are constructed from these instances in three ways (modes): a machine delete mode (MDM), a part add mode (PAM), and a machine swap mode (MSM) .

Machine delete mode This mode reflects one of the real-world scenarios in which a machine suddenly breaks down. The change step of this mode is the deletion of a single machine.

Part add mode This mode reflects the addition and deletion of new assignments (parts). The step of change in this mode is the addition or the deletion of a single part. This mode requires variable representation to reflect the increase or decrease in the number of operations associated with the changing parts.

Machine swap mode This mode is a direct application of the mapping-based benchmark generation scheme, which is introduced in Section 5.4. By interchanging machine labels, a dynamic FMS can be generated easily and quickly. The change step in this mode is an interchange of a single pair of machines. As a mapping change scheme, this mode does not require computing a new solution after each change. We only need to swap the machines of the current optimal solution to determine the optimum of the next instance.

In the current experimentation, each benchmark problem is created from an initial sequence of 100 static problems inter-separated by single elementary steps. Depending on the specified severity, a number of intermediate static problems will be skipped to construct one test problem.

Each sequence of static problems is translated into 18 dynamic test problems by combining seven degrees of severity (1, 2, 3, 5, 10 steps per shift, and random) and three periods of change (500, 2500, and 5000 evaluations per shift, which correspond to 10, 50, and 100 generations per shift based on a population of 50 individuals).

8.3.4 Algorithm settings

In all tested models, the underlying GA is generational with tournament selection in which selection pressure can be altered by changing a selection probability parameter. Structured crossover and simple mutation are used throughout (see Appendix C for more details on the underlying genetic operator and the pseudo code specifically designed for the multiple objective FMS).

A population size of 50 individuals is used throughout. For the FM, RM and RIM models, a crossover rate of 0.9, a selection probability of 1.0, and a mutation rate of $\frac{1}{l}$ are used. For the ADM, AIM, AHDM, and AHIM models, the previous values represent one limit on the corresponding operator; the other limit is 1.0, 0.9, and $\frac{2}{l}$ for crossover, selection and mutation respectively.

Individuals in the population are assigned to five islands in the AIM and AHIM models. AHDM and AHIM use a local search heuristic based on a simple neighborhood definition, in which two individuals are considered neighbors if any one of them can be generated by altering the value of one gene in the other individual. The number of local search evaluations per generation is limited to 50.

8.3.5 Results

The results of experiments conducted on the rnd1, gap1, gap2, and gap3 problems in the three modes of environmental change are given in Appendix B. In this section, we focus on the gap1 problem, the largest and presumably the hardest, and on the rnd1 problem, the most distinct.

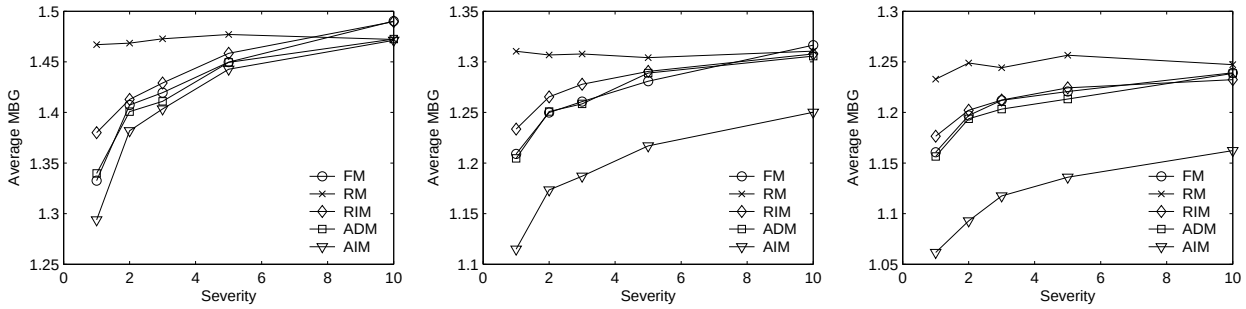
Comparison of evolutionary models

Results of comparisons among the pure evolutionary models on the rnd1 problem in the MSM mode are shown in Figure 8.9, where the average MBG (over ten runs) is plotted against different values of severity.

First, we notice that results of the RM model are inferior to those of the other models when the change severity is small. As severity increases, RM results become comparatively better, and at extreme severities RM outperforms the other models. This trend is consistent over different periods of environmental change confirming our notion that restart strategies are best used when the problem changes completely; i.e., when no benefits are expected from re-using old information.

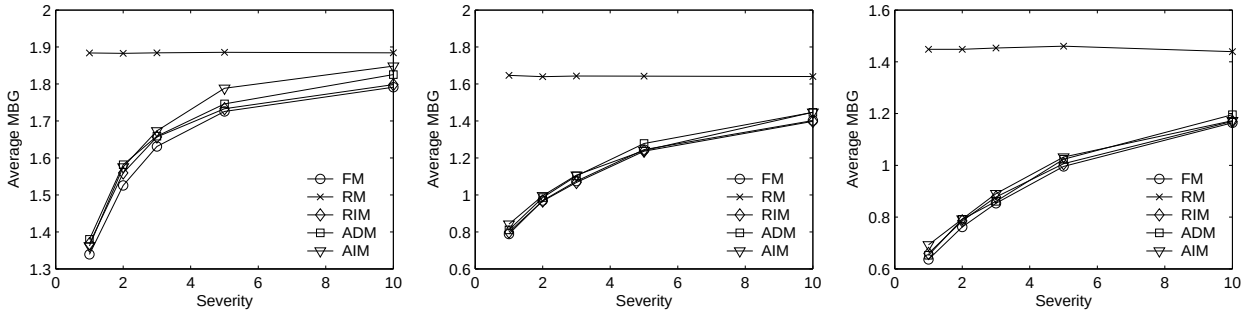
Starting with the ten generation period, we notice that models that reuse old information (all models except for RM) give comparable performance. However, as the period of change increases, differences between their performance become more apparent. This trend can be explained as follows: when the environmental change is fast, the models do not have sufficient time to converge, and hence they give nearly the same results. When allowed more time, the models start to converge, and those using the best approach to persevere after obsolete convergence produce the best results.

The inferior performance of the the RM model is more apparent in the larger problem (i.e., gap1), as illustrated in Figure 8.10. Whereas the performance of the other models deteriorates with change severity, the performance of the RM model is consistently poor



Period = 10 generations Period = 50 generations Period = 100 generations

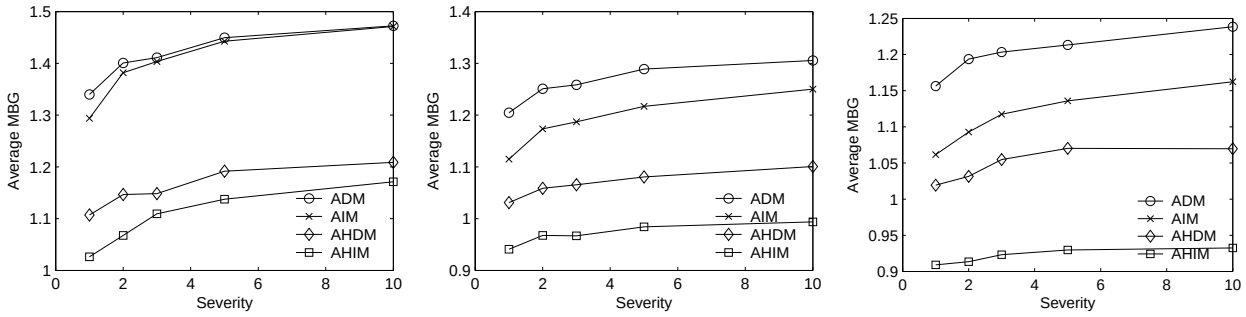
Fig. 8.9: Comparison of evolutionary models (rnd1_MSM)



Period = 10 generations Period = 50 generations Period = 100 generations

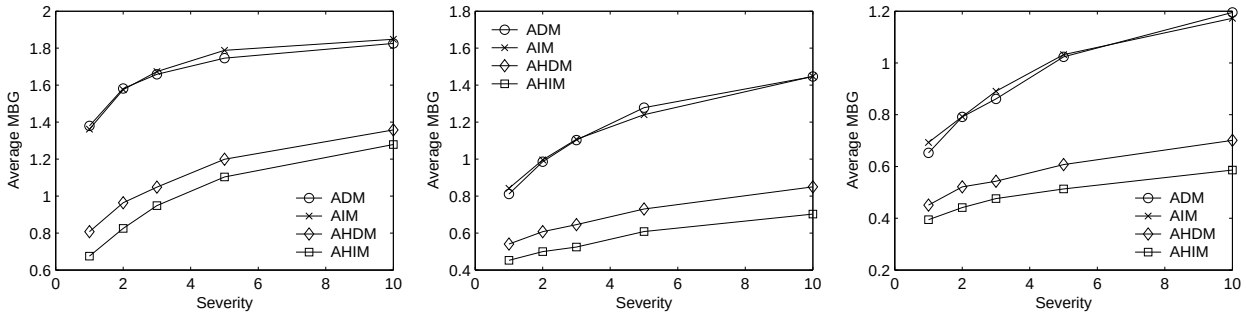
Fig. 8.10: Comparison of evolutionary models (gap1_MSM)

across the problem dynamics. We note here that the performance of RM (relative to the other models) seems worse than that of rnd1; this can be explained by examining change severity. Although values of severity are numerically the same in both cases, relative to problem size they are different, since gap1 is larger than rnd1. In other words, the severity range used in the experiments on gap1 is virtually less than that used in the experiments on rnd1.



Period = 10 generations Period = 50 generations Period = 100 generations

Fig. 8.11: Comparison of hybridized models (rnd1_MSM)



Period = 10 generations Period = 50 generations Period = 100 generations

Fig. 8.12: Comparison of hybridized models (gap1_MSM)

Comparison of hybridized models

Results of comparisons of AIM and ADM against their hybridized counterparts (AHIM and AHDM) are shown in Figure 8.11 for the rnd1 problem in the MSM mode. First, we observe that in general all hybridized models are by far better than pure evolutionary models over all the considered dynamics. The superiority of the island models is noticeable too (compare AIM to ADM and AHIM to AHDM).

Comparisons on the gap1 problem in the MSM mode are given in Figure 8.12. Whereas

ADM and AIM give similar performance, AHDM and AHIM produce more distinct performance.

The superiority of the AHIM model over all other models is obvious on all the dynamics, modes, and instances considered. This good performance is not surprising since AHIM, in addition to using local search to fine-tune the search, maintains diversity without destroying individuals of good quality.

It is worth mentioning here that the previous discussion (on the results of the evolutionary models and hybridized models) is based on the MSM mode only. It is sufficient to use this mode to investigate relative performance of all models. This is another demonstration of the effectiveness of the mapping-based benchmarking scheme in generating test problems. A quick glance at the figures in Appendix B.2 confirms this observation.

Individual model results

More details on the conducted experiments are given in Table 8.11 for the ADM model and in Table 8.12 for its hybridized counterpart (AHDM). For the AIM and AHIM, the results are given in Table 8.13 and 8.14.

The number of evaluations needed to reach the best solution after an environmental change is averaged for all instances of the problem. The adaptation columns give the average and the standard deviation (over the conducted runs) of the number of evaluations per instance, including those of the local search when applicable. The MBG columns give the worst, the average, the best, and the standard deviation of the mean best of generation over the conducted runs. The last column gives the mean CPU time per instance (from an environmental change till the discovery of the best solution); $mean\ CPU = \frac{T}{R} \times \frac{E}{E_t}$, where T is the total CPU time, R is the number of conducted runs, E is the average number of evaluations to best per instance, and E_t is the average total number of evaluations per run.

There are no significant differences between values of the worst, average and best of

Table 8.11: Results of ADM (gap1_MSM)

		Adaptation		MBG				CPU
Per.	Sev.	avg	std	worst	avg	best	std	(sec)
	1	753.	13.13	1.4053	1.3798	1.3546	0.02	0.09
	2	754.	24.10	1.5971	1.5816	1.5653	0.01	0.09
	3	741.	20.20	1.6788	1.6590	1.6338	0.02	0.09
10	5	758.	26.00	1.7786	1.7461	1.6975	0.03	0.10
	10	753.	31.58	1.8484	1.8253	1.7780	0.03	0.10
	rnd	755.	5.12	1.6401	1.5954	1.5516	0.04	0.09
	1	3774.	68.96	.8528	.8109	.7740	0.04	0.41
	2	3896.	48.27	1.0380	.9858	.9079	0.05	0.42
	3	3968.	19.18	1.1349	1.1030	1.0588	0.04	0.43
50	5	3988.	44.99	1.2932	1.2780	1.2454	0.02	0.44
	10	4015.	24.65	1.5033	1.4469	1.4055	0.04	0.45
	rnd	3884.	90.99	1.0319	1.0089	.9635	0.03	0.42
	1	7252.	189.67	.6728	.6524	.6360	0.01	0.77
	2	7593.	113.62	.8226	.7914	.7463	0.03	0.81
	3	7651.	302.99	.9214	.8610	.8038	0.04	0.83
100	5	7977.	99.09	1.0482	1.0237	1.0047	0.02	0.86
	10	7979.	98.57	1.2247	1.1954	1.1743	0.02	0.87
	rnd	7622.	371.17	.8287	.7897	.7278	0.04	0.82

Table 8.12: Results of AHDM (gap1_MSM)

		Adaptation		MBG				CPU
Per.	Sev.	avg	std	worst	avg	best	std	(sec)
	1	1163.	25.56	.8336	.8075	.7810	0.02	0.06
	2	1177.	27.67	.9944	.9637	.9186	0.03	0.07
	3	1188.	49.02	1.0964	1.0483	1.0077	0.04	0.07
10	5	1201.	40.15	1.2431	1.1992	1.1670	0.03	0.07
	10	1220.	31.61	1.3980	1.3577	1.3122	0.04	0.08
	rnd	1156.	47.81	1.0151	.9606	.9178	0.04	0.06
	1	4985.	448.03	.5518	.5408	.5248	0.01	0.26
	2	5703.	203.83	.6273	.6077	.5810	0.02	0.30
	3	5830.	318.43	.6542	.6457	.6351	0.01	0.31
50	5	6126.	151.12	.7691	.7301	.7004	0.03	0.33
	10	5840.	379.96	.9042	.8495	.7961	0.04	0.32
	rnd	5754.	98.76	.6408	.6129	.5691	0.03	0.30
	1	8309.	942.71	.4767	.4510	.4063	0.03	0.43
	2	9859.	461.59	.5477	.5213	.4560	0.04	0.51
	3	10630.	732.22	.5584	.5428	.5325	0.01	0.56
100	5	11446.	632.88	.6339	.6072	.5570	0.03	0.60
	10	12121.	1305.84	.7482	.7009	.6846	0.03	0.64
	rnd	9797.	587.27	.5609	.5142	.4786	0.03	0.51

Table 8.13: Results of AIM (gap1_MSM)

		Adaptation		MBG				CPU
Per.	Sev.	avg	std	worst	avg	best	std	(sec)
	1	789.	11.28	1.3878	1.3623	1.3214	0.02	0.09
	2	787.	21.99	1.6032	1.5758	1.5244	0.03	0.09
	3	781.	18.08	1.7413	1.6733	1.6223	0.05	0.10
10	5	786.	33.80	1.8008	1.7883	1.7728	0.01	0.10
	10	803.	18.81	1.8746	1.8483	1.8262	0.02	0.11
	rnd	793.	15.57	1.6148	1.5779	1.5347	0.03	0.09
	1	4099.	87.16	.8608	.8418	.8234	0.02	0.45
	2	4184.	169.52	1.0168	.9958	.9677	0.02	0.46
	3	4377.	75.73	1.1274	1.1070	1.0760	0.02	0.48
50	5	4370.	34.12	1.2700	1.2400	1.2166	0.02	0.49
	10	4342.	57.53	1.5032	1.4476	1.4122	0.04	0.50
	rnd	4291.	95.92	1.1069	1.0368	.9806	0.05	0.47
	1	8286.	146.05	.7279	.6928	.6566	0.03	0.90
	2	8400.	379.46	.8550	.7926	.7147	0.05	0.91
	3	8595.	333.84	.9128	.8906	.8713	0.01	0.94
100	5	8941.	66.47	1.0680	1.0321	.9924	0.03	0.98
	10	8870.	90.69	1.2499	1.1727	1.1364	0.05	0.98
	rnd	8637.	211.30	.8982	.8406	.8230	0.03	0.94

Table 8.14: Results of AHIM (gap1_MSM)

		Adaptation		MBG				CPU
Per.	Sev.	avg	std	worst	avg	best	std	(sec)
	1	2796.	34.82	.7167	.6752	.6472	0.03	0.15
	2	2868.	38.66	.8837	.8249	.7947	0.04	0.15
	3	2906.	56.13	.9699	.9484	.9290	0.02	0.16
10	5	2935.	42.34	1.1282	1.1036	1.0569	0.03	0.16
	10	2868.	83.99	1.3117	1.2786	1.2177	0.04	0.16
	rnd	2897.	49.77	.9241	.8510	.8236	0.04	0.15
	1	11930.	947.74	.4957	.4523	.4189	0.03	0.62
	2	12577.	1686.69	.5385	.4996	.4684	0.03	0.65
	3	14463.	822.84	.5490	.5243	.5029	0.02	0.75
50	5	14250.	1155.78	.6413	.6082	.5763	0.03	0.74
	10	15311.	1002.54	.7652	.7025	.6791	0.04	0.80
	rnd	12429.	1721.15	.5193	.5060	.4844	0.01	0.64
	1	18911.	1014.74	.4117	.3943	.3834	0.01	0.98
	2	20889.	3539.95	.4588	.4413	.4109	0.02	1.08
	3	21604.	3869.70	.5032	.4758	.4429	0.03	1.12
100	5	23718.	3479.17	.5420	.5131	.4784	0.03	1.23
	10	27449.	3629.76	.6332	.5862	.5467	0.04	1.43
	rnd	18285.	1000.20	.4768	.4481	.4313	0.02	0.95

runs. This is also evident in the small values of the standard deviation. The obvious explanation is that the number of instances involved per run is large enough to give a reasonably accurate estimate of the algorithm's performance.

Examining instance run time with and without local search, we notice that run time does not increase after local search is embedded, as can be seen by comparing the CPU column of Table 8.11 against that of Table 8.12. Indeed, run time decreases in some cases, which attests to the efficiency gained by employing local search heuristics. We further note that the gains in run time occur despite the associated increase in the number of evaluations per instance. This is not surprising since local search evaluations (which are about half the total evaluations) are relatively cheap to perform. In local search, only the difference between the fitness of the candidate solution and that of the current solution is computed. Since only one gene is changed at a time by the local search operator, the cost of computing fitness difference is in the order of $\frac{1}{l}$ of that of evaluating an entire individual, where l is the chromosome length.

Nevertheless, runtime of AHIM is considerably longer than that of AIM, as can be seen by comparing Table 8.13 against Table 8.14. In AHIM, the number of local search evaluations increases profoundly since each island-best undergoes local search, which raises the number of allowable local search evaluations in AHIM to about five times (number of islands) the number of local search evaluations in AHDM. However, we can still favor AHIM because of the good MBG values obtainable with AHIM relative to AHDM. Moreover, if several processors are available as mentioned in the discussion of the TSP results, the AHIM model becomes absolutely the best choice, as it lends itself easily to parallelization.

Statistical Analysis

Statistical results reported in this section are obtained using a significance level of 5% to construct 95% confidence intervals on the difference in the mean best of generation (see Section 4.5.3 for details on how these tests are carried out).

Table 8.15: Multiple comparison test of evolutionary models (rnd1-MSM)

period	10						100						1000					
severity	1	2	3	5	10	r	1	2	3	5	10	r	1	2	3	5	10	r
FM−RM	-1	-1	-1	0	0	-1	-1	-1	-1	0	0	-1	-1	-1	0	0	0	-1
FM−RIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−ADM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−AIM	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
RM−RIM	1	1	1	0	0	1	1	1	0	0	0	1	1	1	0	0	0	1
RM−ADM	1	1	1	0	0	1	1	1	1	0	0	1	1	1	0	0	0	1
RM−AIM	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−ADM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RIM−AIM	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	1
ADM−AIM	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1

Table 8.16: Multiple comparison test of evolutionary models (gap1-MSM)

period	10						100						1000					
severity	1	2	3	5	10	r	1	2	3	5	10	r	1	2	3	5	10	r
FM−RM	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
FM−RIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−ADM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RM−RIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−ADM	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−AIM	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−ADM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RIM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ADM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 8.15 gives the results of multiple post ANOVA comparison test for the rnd1 problem in the MSM mode of change. This table covers 18 combinations of problem dynamics (three periods of change and six levels of severity of change including one level with a random severity). Thus, each row corresponds to 21 comparisons between a pair of models. An entry of 1 signifies that the confidence interval for the difference in performance measures of the corresponding pair consists entirely of positive values, which indicates that

Table 8.17: Multiple comparison test of hybridized models (rnd1-MSM)

period	10						100						1000					
severity	1	2	3	5	10	r	1	2	3	5	10	r	1	2	3	5	10	r
ADM−AIM	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	1
ADM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ADM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
AIM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AHDM−AHIM	1	1	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1

Table 8.18: Multiple comparison test of hybridized models (gap1-MSM)

period	10						100						1000					
severity	1	2	3	5	10	r	1	2	3	5	10	r	1	2	3	5	10	r
ADM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ADM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ADM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AHDM−AHIM	1	1	0	1	0	0	1	0	1	1	1	0	0	0	0	1	1	0

the first model is inferior to the second model. Conversely, an entry of -1 signifies that the confidence interval for the corresponding pair consists entirely of negative values, which indicates that the first model is superior to the second one. An entry of 0 indicates that there is no significant difference between the two models.

Similarly, test results on gap1 in the MSM mode are given in Table 8.16. For the hybridized models, the results are given in Table 8.17 for the rnd1 problem, and in Table 8.18 for the gap1 problem.

Collectively, the statistical tables confirm the graphical comparisons presented in the previous section. As can be seen in Table 8.15, and 8.16, there are significant differences between the performance of the RM model and all others. As well, Table 8.17 and 8.18 emphasize the superiority of the hybrid models over the pure evolutionary models, and

confirm some of the cases in which AHIM outperforms AHDM.

8.4 Summary

In this chapter, results of experimentation on dynamic TSP and FMS are presented. The four models (ADM, AIM, AHDM, and AHIM) developed in this thesis are tested on each problem.

The benchmarks are constructed from available problem instances under several modes of environmental change in accordance with the generalized framework of benchmark generation.

It is clear from the results that ADM and AIM are very competitive and can produce results better than those obtainable with traditional strategies. On the other hand, their performance is clearly inferior to that of the hybridized models, which give superior performance with little or no adverse effect on computational costs.

Although the diversity controlled models produce good results, their relative performance on TSP differ from that on FMS. For example, AHDM and AHIM give comparable results in the case of TSP, whereas AHIM outperforms AHDM on all FMS instances. Nevertheless, it is clear that local search when properly embedded enhances the performance of the host algorithm with negligible increase in computational cost.

The mapping-based scheme proved to be effective in producing test problems for comparing algorithm performance.

We also note that these results are obtained using conventional rates of mutation without attempting to fine-tune this parameter. Most tuning efforts are directed to finding appropriate limits of diversity.

Chapter 9

Conclusions and Future Work

We shall not cease from exploration
And the end of all our exploring
Will be to arrive where we started
And know the place for the first time.
From "Little Gidding", T.S. Eliot 1963

9.1 Conclusions

Real-world problems seldom change completely: changes, while frequent, remain sufficiently small, and consequently, significant benefits can be gained by adapting solutions to the changes. This thesis has concentrated on the application of evolutionary algorithms to dynamic combinatorial problems. The vast amount of research on evolutionary algorithms indicates that they have established themselves as an effective optimization tool even though their theoretical foundations are still debated.

Tackling dynamic problems entails addressing several issues related to algorithm development, implementation, and testing. Although a general treatment of static COPs may be difficult, most dynamism related issues can be addressed in a unified way. In this thesis, general methodologies of testing, benchmarking, and adaptation are developed. Results of experimentation on dynamic versions of TSP and FMS demonstrate that the methods are effective on both problems and hence have a great potential for other dynamic COPs as well.

The research conducted in this thesis leads to the following conclusions:

A unified methodology to generate benchmarks for dynamic COPs is worth developing. In this research, general methods for constructing benchmarks for different COPs have been successfully devised. It was relatively easy to construct FMS benchmarks after constructing TSP benchmarks. It should prove to be easy to apply these methods to other COPs. Furthermore, by using the same measures of dynamics, results can be easily mapped from one problem to another. existing benchmarks for combinatorial problems are of limited use as they are typically described in a case-based or application-based context.

It is important to be able to adapt to environmental changes without inflicting extra demands in terms of parameter tuning. One problem with evolutionary algorithms is that their results often depend on a number of parameters. This problem is aggravated in algorithms targeting dynamic environments, as a result of the increased problem complexity

and the increased number of algorithm parameters. By using diversity to control the EA parameters, the models developed in this thesis had significantly reduced tuning efforts.

The real issue is not just to react to environmental changes after convergence but also to obtain this convergence in the first place during the allowable time. EAs in their standard form suffer from the problem that once a population converges around an optimum it will be unable to further explore the search space after the environment shifts. This difficulty is commonly appreciated, and often cited as a main motive for seeking adaptive schemes. However, this difficulty is not so pressing as is generally believed; our first priority should be finding solutions quickly during static phases. Indeed, the results of this research show that models which merely react to environmental changes are in general inferior to models which exploit the static phases of the problem.

The purpose of measuring diversity is to assess the explorative *state of the search process to update the algorithm parameters, rather than precisely determining variety in the population as a goal in itself.* The models developed in this thesis measure population diversity in order to control the parameters of the algorithm so that diversity can be maintained within acceptable limits. However, while researchers agree on the importance of maintaining diversity, both in static and dynamic environments, they do not agree on how to maintain diversity, let alone measuring it. Some measures are accurate but expensive; others are cheaper but less accurate. Thus, a tradeoff between accuracy and cost of computation has to be sought.

Performance of evolutionary algorithms can be greatly improved by an appropriate *hybridization of these algorithms with local search heuristics.* The common practice of using exhaustive local search, in which each individual in the population undergoes local search at every generation, is not an ideal choice for dynamic problems. In fact, one can debate the use of exhaustive local search even in static problems in view of the associated computational costs and the risk of losing diversity.

9.2 Contributions

This thesis targets practical aspects in the field of evolutionary computation in dynamic and uncertain environments. Issues that hinder algorithm implementation and testing are considered of utmost importance. The treatment of these issues constitute the thesis' main contributions, which are summarized in conjunction with relevant publications in this section.

1. On the aspect of algorithm development, this thesis resulted in strategies to improve the ability of the algorithm to adapt to environmental changes, and more importantly to improve its efficiency at finding quality solutions:

    - A model that employs time dependent genetic parameters is developed. Experimental results demonstrate the benefits of proposed adaptation in enhancing the overall performance without increasing computational costs. [Younes and Basir 2002; Younes 2002; Younes et al. 2004]
    - Methods to measure diversity and to adaptively control genetic parameters throughout the search process are developed [Younes et al. 2005, 2006a].
    - A hybridized island model that seeks efficient implementation of local search in a multiple population EA is derived [Younes et al. 2006b].

2. In addressing benchmark generation, this thesis resulted in procedures that consider combinatorial problems collectively:

    - A mapping based scheme to create dynamic test problems is presented and applied [Younes and Basir 2002; Younes et al. 2003, 2005]. Unlike existing schemes this scheme can produce problem instances with known optima, without the need of solving the new instances.

- Dynamically insignificant change and misleading patterns of change, which can hinder the generation of effective benchmarks, are identified and addressed in Younes et al. [2005, 2006].
- A framework for benchmark generation for dynamic COPs is proposed [Younes et al. 2005]. By viewing the dynamic problem as a time sequence of (static) instances, dynamic benchmarks can be treated on two levels, one level to target the base static problem and the other level to target patterns of environmental changes. As the first level is well-covered in the literature of static optimization, the user can devote efforts to the second level.

3. To demonstrate the applicability of algorithmic and benchmarking ideas to more complex problems, a multiple objective dynamic FMS problem is used:

   - An EA for a (static) two-objective FMS problem is developed [Younes et al. 2002]. The algorithm produces excellent results with respect to part transfer and balancing the work among the machines.
   - An EA for a dynamic FMS problem is developed [to be submitted to European Journal of Operations Research Sept 2006]. The algorithm involves three conflicting objectives, benchmarks of different categories, and adaption strategies developed in various stages of this thesis.

Nevertheless, there are some points that would have complemented this work but were impossible to address in the given time frame. These points can be the subject of future work as described in the next section.

9.3 Future work

There are several ways to improve, extend, or apply the developed models. In this section, we briefly discuss some of these prospects.

- Diversity controlled models can use operator-specific diversity measures so that each operator is controlled by its respective diversity measure. Such measures are likely to be computationally expensive. However, if they are implemented based on the population-best, the increase in cost can be limited.

- Instead of local search, other metaheuristics can be imbedded in the evolutionary algorithm. In particular, tabu search has proven to be one of the best methods for many static COPs, and hence it is interesting to investigate its potential in dynamic problems as well.

- Another idea to explore is to adaptively update limits of diversity for the models presented in this thesis. Roughly, the idea is to use an additional processor that executes the same algorithm on a problem instance that changes under the mapping scheme. The task of the additional processor is to find the best diversity limits for the current instance and update the main model on regular intervals.

- In evaluating the performance of an algorithm on a dynamic problem, it is customary to output the performance criterion against severity of change, since severity can influence the choice of the adaptation strategy. However, change severity is commonly measured in terms of distance between solutions of successive problem instances. While this practice can be effective on benchmarks, it is hard to apply to real-world problems (with unknown solutions), where only the problem input parameters can be measured. Change in input parameters does not necessarily induce proportional change in problem solution. Hence, a possible aspect of future work is to employ tools

of reinforced learning to predict change severity, or to conduct statistical tests to determine correlation between solution-based severity and parameter-based severity.

- This thesis borrows several ideas from pioneering work on continuous optimization, for which the author is grateful. Nevertheless, in applying those ideas to dynamic COPs, several issues have been addressed, hence the developed techniques are also useful for dynamic continuous optimization and also for static COPs. For example, methods proposed to measure population diversity and to apply local search can be used to obtain efficient implementations in static optimization.

- The ability of the developed models to retain good performance over a wide range of mutation rate encourages investigating their use to reduce efforts of parameter tuning in static problems. In fact, the treatment of a dynamic problem as a series of static ones readily allows the application of the developed models to conventional static problems.

- The generalized framework of benchmarking can be extended to other COPs. Table E.1 summarizes all possible changes in three COP examples considered in this thesis. It is hoped that in the future other interesting dynamic COPs are constructed under the generalized framework and their corresponding properties are added to this table.

- The effectiveness of the developed methods on the TSP and FMS problems encourages their application to other dynamic problems, such as intelligent transportation systems, engine parameter control, scheduling of airline maintenance, and dynamic network routing. With these problems, however, several important application dependent aspects may have to be investigated. Examples are the integration of solution implementation cost in the the objective function and the the interaction between implemented solutions and system states.

- More importantly, the search of robust solutions is an integral part to the research conducted in this thesis. Robust solutions are required for cases where the application does not permit changing the solution frequently, when cost of switching between solutions is large, or when simply the dynamic solver deliver new solutions within the allowable time limits.

The ever-increasing dynamism in real world problems and competition among various enterprizes are likely to bring about newer optimization problems in which adaptation is absolutely essential while conventional methods such as re-starting after every environmental change are not an option any more.

Adapting Evolutionary Approaches for Optimization in Dynamic Environments

Appendices

Appendix A

TSP Results

This appendix lists results of experiments conducted on the dynamic TSP problem. The appendix contains sufficient detail for readers who want to replicate the work.

A.1 Individual strategy results

Details on the conducted experiments on individual models are given in this section. The number of evaluations needed to reach the best solution after an environmental change is averaged for all instances of the problem. The adaptation columns give the average and the standard deviation (over the conducted runs) of the number of evaluations per instance, including those of the local search when applicable. The MBG columns give the worst, the average, the best, and the standard deviation of the mean best of generation over the conducted runs. The last column gives the mean CPU time per instance (from an environmental change till the discovery of the best solution); $mean\ CPU = \frac{T}{R} \times \frac{E}{E_t}$, where T is the total CPU time, R is the number of conducted runs, E is the average number of evaluations to best per instance, and E_t is the average total number of evaluations per run.

Table A.1: Results of ADM (be52_SW)

		Adaptation		MBG				CPU
Per.	Sev.	avg	std	worst	avg	best	std	(sec)
	1	342.	11.02	2.2879	2.2437	2.1910	0.03	0.01
	5	328.	6.18	3.0020	2.9172	2.8537	0.04	0.01
	10	308.	6.06	3.2008	3.1650	3.1364	0.02	0.01
10	15	300.	8.06	3.3165	3.2686	3.2134	0.03	0.01
	20	296.	5.22	3.3583	3.3209	3.2841	0.03	0.01
	25	292.	7.75	3.4188	3.3584	3.3036	0.03	0.01
	rnd	319.	7.96	3.0716	3.0204	2.9765	0.03	0.01
	1	1740.	37.16	1.7068	1.6696	1.6316	0.03	0.03
	5	1782.	27.78	2.1301	2.1077	2.0810	0.01	0.04
	10	1688.	19.12	2.4294	2.4125	2.3969	0.01	0.03
50	15	1625.	23.05	2.5905	2.5641	2.5361	0.02	0.03
	20	1582.	25.95	2.6950	2.6636	2.6301	0.03	0.03
	25	1537.	21.38	2.7915	2.7550	2.7278	0.02	0.04
	rnd	1738.	31.17	2.2625	2.2297	2.1776	0.03	0.04
	1	3336.	108.11	1.5334	1.5162	1.5026	0.01	0.06
	5	3730.	40.22	1.8784	1.8411	1.8093	0.02	0.07
	10	3600.	40.91	2.0807	2.0649	2.0523	0.01	0.07
100	15	3503.	52.36	2.2117	2.1783	2.1339	0.02	0.07
	20	3490.	31.38	2.3005	2.2755	2.2561	0.01	0.07
	25	3421.	49.03	2.3757	2.3434	2.3099	0.02	0.07
	rnd	3605.	65.50	1.9289	1.9170	1.9025	0.01	0.07

Table A.2: Results of AHDM (be52_SW)

		Adaptation		MBG				CPU
Per.	Sev.	avg	std	worst	avg	best	std	(sec)
	1	802.	5.78	1.5364	1.4995	1.4720	0.02	0.01
	5	781.	3.18	2.3179	2.2604	2.2136	0.03	0.01
	10	761.	3.54	2.7029	2.6681	2.6319	0.02	0.01
10	15	749.	5.37	2.8549	2.8164	2.7557	0.03	0.01
	20	745.	6.24	2.9797	2.9104	2.8698	0.03	0.01
	25	739.	7.48	3.0170	2.9827	2.9576	0.02	0.01
	rnd	772.	4.29	2.4473	2.4130	2.3715	0.02	0.01
	1	3602.	54.31	1.1646	1.1565	1.1456	0.01	0.03
	5	3818.	25.42	1.3916	1.3788	1.3704	0.01	0.04
	10	3641.	18.16	1.6032	1.5934	1.5853	0.01	0.04
50	15	3546.	23.23	1.7386	1.7215	1.7021	0.01	0.04
	20	3494.	15.78	1.8511	1.8261	1.8077	0.01	0.04
	25	3455.	30.46	1.9241	1.9062	1.8751	0.01	0.04
	rnd	3726.	36.13	1.4723	1.4620	1.4465	0.01	0.04
	1	5899.	278.01	1.1122	1.1053	1.0974	0.01	0.05
	5	7283.	100.13	1.2423	1.2324	1.2243	0.01	0.06
	10	7402.	69.37	1.3544	1.3504	1.3452	0.00	0.06
100	15	7283.	62.60	1.4306	1.4196	1.4084	0.01	0.07
	20	7297.	93.24	1.4987	1.4777	1.4695	0.01	0.07
	25	7204.	56.03	1.5401	1.5243	1.5118	0.01	0.07
	rnd	7020.	130.94	1.2824	1.2768	1.2649	0.00	0.06

Table A.3: Results of AIM (be52_SW)

		Adaptation		MBG				CPU
Per.	Sev.	avg	std	worst	avg	best	std	(sec)
	1	384.	12.43	2.4121	2.3564	2.2963	0.04	0.01
	5	419.	9.54	2.9867	2.9229	2.8835	0.03	0.01
	10	425.	5.35	3.1760	3.1545	3.1389	0.01	0.01
10	15	427.	13.43	3.2881	3.2294	3.1989	0.03	0.01
	20	429.	11.73	3.3211	3.2695	3.2277	0.03	0.01
	25	430.	19.46	3.3590	3.3131	3.2539	0.03	0.01
	rnd	422.	7.52	3.0365	3.0078	2.9740	0.02	0.01
	1	2155.	38.91	1.6786	1.6543	1.6137	0.02	0.03
	5	2345.	19.85	2.1972	2.1692	2.1381	0.02	0.04
	10	2358.	14.24	2.4638	2.4395	2.4186	0.01	0.04
50	15	2333.	22.36	2.6020	2.5770	2.5549	0.02	0.04
	20	2334.	21.06	2.6954	2.6640	2.6139	0.03	0.04
	25	2326.	23.84	2.7690	2.7390	2.7204	0.02	0.04
	rnd	2329.	19.34	2.3045	2.2750	2.2478	0.02	0.04
	1	4362.	86.14	1.4776	1.4637	1.4470	0.01	0.07
	5	4771.	24.53	1.9002	1.8694	1.8431	0.02	0.08
	10	4791.	30.95	2.1227	2.1020	2.0908	0.01	0.08
100	15	4769.	39.08	2.2423	2.2234	2.1938	0.02	0.07
	20	4789.	58.07	2.3218	2.3023	2.2790	0.01	0.07
	25	4750.	40.30	2.3770	2.3665	2.3371	0.01	0.07
	rnd	4714.	57.45	1.9711	1.9426	1.9173	0.02	0.08

Table A.4: Results of AHIM (be52_SW)

		Adaptation		MBG				CPU
Per.	Sev.	avg	std	worst	avg	best	std	(sec)
	1	2601.	18.81	1.4369	1.4213	1.4013	0.01	0.01
	5	2657.	14.67	2.1687	2.1457	2.1230	0.01	0.01
	10	2653.	16.69	2.5533	2.5355	2.5128	0.01	0.01
10	15	2647.	18.09	2.7197	2.6951	2.6666	0.02	0.01
	20	2646.	10.01	2.8218	2.8020	2.7931	0.01	0.01
	25	2638.	11.80	2.8956	2.8665	2.8419	0.02	0.01
	rnd	2661.	14.55	2.3099	2.2941	2.2750	0.01	0.01
	1	11511.	371.90	1.1122	1.0954	1.0899	0.01	0.03
	5	14105.	84.73	1.3136	1.3095	1.3060	0.00	0.04
	10	14370.	41.20	1.5165	1.5094	1.5048	0.00	0.04
50	15	14439.	30.39	1.6374	1.6310	1.6222	0.01	0.04
	20	14431.	49.94	1.7445	1.7335	1.7264	0.01	0.04
	25	14415.	45.65	1.8297	1.8134	1.8014	0.01	0.04
	rnd	13913.	107.91	1.3934	1.3860	1.3759	0.01	0.04
	1	18507.	443.89	1.0683	1.0559	1.0477	0.01	0.05
	5	25478.	389.34	1.1766	1.1710	1.1670	0.00	0.07
	10	27030.	232.08	1.2887	1.2837	1.2796	0.00	0.08
100	15	27439.	228.56	1.3558	1.3491	1.3449	0.00	0.08
	20	27865.	212.50	1.4200	1.4151	1.4070	0.00	0.08
	25	27974.	319.42	1.4601	1.4547	1.4475	0.00	0.08
	rnd	25034.	439.90	1.2179	1.2127	1.2073	0.00	0.07

Table A.5: Results of ADM (k100_SW)

		Adaptation		MBG				CPU
Per.	Sev.	avg	std	worst	avg	best	std	(sec)
	1	312.	12.79	4.2544	4.1696	4.0287	0.08	0.02
	5	363.	7.85	5.3227	5.2459	5.1925	0.04	0.02
	10	353.	7.36	5.9298	5.8394	5.7440	0.06	0.02
10	15	339.	10.07	6.1893	6.1003	6.0180	0.06	0.02
	20	332.	7.33	6.3907	6.2976	6.2411	0.05	0.02
	25	334.	11.89	6.5481	6.3933	6.2391	0.08	0.02
	rnd	351.	10.12	5.5935	5.5343	5.4797	0.04	0.02
	1	1711.	53.35	3.0732	3.0133	2.9342	0.05	0.08
	5	1918.	43.19	3.7874	3.7452	3.6892	0.03	0.09
	10	1863.	36.44	4.3391	4.2801	4.2505	0.03	0.10
50	15	1814.	22.38	4.6390	4.5802	4.5202	0.04	0.10
	20	1783.	32.59	4.8829	4.8245	4.8007	0.03	0.11
	25	1733.	22.03	5.0818	5.0274	4.9780	0.04	0.11
	rnd	1882.	21.65	4.0343	4.0005	3.9567	0.03	0.10
	1	3614.	89.30	2.8018	2.7503	2.7136	0.03	0.17
	5	3998.	66.90	3.3303	3.3027	3.2719	0.02	0.20
	10	3910.	57.48	3.7088	3.6736	3.6406	0.02	0.20
100	15	3829.	38.00	3.9868	3.9355	3.8898	0.03	0.20
	20	3791.	66.66	4.1625	4.1043	4.0576	0.03	0.21
	25	3699.	44.49	4.3260	4.2796	4.2371	0.03	0.21
	rnd	3935.	49.00	3.5032	3.4738	3.4230	0.02	0.19

Table A.6: Results of AHDM (k100_SW)

		Adaptation		MBG				CPU
Per.	Sev.	avg	std	worst	avg	best	std	(sec)
	1	893.	5.62	2.3349	2.2873	2.2261	0.03	0.02
	5	878.	5.63	4.0525	4.0075	3.9159	0.04	0.02
	10	851.	5.12	5.0538	4.9714	4.9032	0.04	0.02
10	15	836.	7.94	5.4601	5.3907	5.3143	0.05	0.02
	20	824.	7.37	5.7285	5.6783	5.5756	0.04	0.02
	25	812.	8.53	5.9179	5.8538	5.7898	0.05	0.02
	rnd	866.	3.45	4.5445	4.5038	4.4536	0.03	0.01
	1	4462.	25.81	1.3611	1.3482	1.3259	0.01	0.10
	5	4442.	13.54	2.0160	1.9973	1.9785	0.01	0.11
	10	4371.	11.74	2.6233	2.6044	2.5881	0.01	0.11
50	15	4353.	19.54	3.0398	3.0252	3.0139	0.01	0.11
	20	4325.	17.30	3.3710	3.3453	3.2880	0.03	0.11
	25	4308.	25.93	3.6641	3.6254	3.5791	0.03	0.11
	rnd	4408.	11.12	2.3210	2.2894	2.2554	0.02	0.10
	1	8605.	93.62	1.2079	1.1925	1.1767	0.01	0.20
	5	8982.	41.58	1.6010	1.5833	1.5717	0.01	0.22
	10	8841.	27.45	1.9616	1.9429	1.9190	0.01	0.21
100	15	8735.	22.78	2.2282	2.2129	2.1939	0.01	0.22
	20	8686.	30.71	2.4622	2.4423	2.4218	0.01	0.22
	25	8630.	35.90	2.6636	2.6367	2.6199	0.01	0.22
	rnd	8921.	34.49	1.7668	1.7484	1.7342	0.01	0.21

Table A.7: Results of AIM (k100_SW)

		Adaptation		MBG				CPU
Per.	Sev.	avg	std	worst	avg	best	std	(sec)
	1	408.	21.44	4.2992	4.1726	4.0818	0.08	0.02
	5	474.	12.20	5.3547	5.3220	5.2555	0.03	0.02
	10	481.	8.12	5.8915	5.8523	5.8206	0.02	0.02
10	15	486.	12.35	6.1579	6.0884	5.9828	0.06	0.02
	20	480.	16.88	6.2926	6.2238	6.1508	0.04	0.02
	25	485.	9.09	6.4055	6.3264	6.2265	0.05	0.02
	rnd	480.	13.22	5.6800	5.5825	5.5110	0.04	0.02
	1	2349.	49.00	2.9043	2.8517	2.8064	0.03	0.10
	5	2632.	21.92	3.8977	3.8398	3.8178	0.03	0.11
	10	2674.	20.50	4.4105	4.3847	4.3541	0.02	0.11
50	15	2691.	23.67	4.7632	4.7081	4.6559	0.04	0.10
	20	2672.	14.27	5.0191	4.9500	4.8684	0.04	0.11
	25	2670.	26.18	5.1663	5.0885	4.9842	0.05	0.11
	rnd	2641.	35.28	4.1381	4.0995	4.0561	0.03	0.11
	1	4802.	60.15	2.4715	2.4250	2.3379	0.05	0.19
	5	5341.	35.45	3.3482	3.2904	3.2519	0.04	0.21
	10	5415.	30.34	3.8214	3.7780	3.7408	0.02	0.22
100	15	5427.	41.46	4.1028	4.0575	4.0008	0.03	0.22
	20	5440.	40.08	4.3082	4.2763	4.2107	0.03	0.22
	25	5426.	35.96	4.4778	4.4356	4.3971	0.03	0.22
	rnd	5369.	25.16	3.5473	3.5075	3.4733	0.02	0.22

Table A.8: Results of AHIM (k100_SW)

		Adaptation		MBG				CPU
Per.	Sev.	avg	std	worst	avg	best	std	(sec)
	1	2684.	14.85	2.1390	2.1185	2.0809	0.02	0.02
	5	2710.	6.55	3.8330	3.7872	3.7361	0.03	0.02
	10	2698.	15.60	4.7669	4.7126	4.6497	0.03	0.02
10	15	2687.	15.03	5.2039	5.1646	5.1190	0.03	0.02
	20	2677.	20.91	5.4836	5.4134	5.3489	0.05	0.03
	25	2693.	24.25	5.6729	5.6164	5.5372	0.04	0.03
	rnd	2705.	11.02	4.2872	4.2485	4.2051	0.03	0.02
	1	14125.	174.61	1.2454	1.2370	1.2320	0.00	0.11
	5	15022.	34.68	1.8466	1.8390	1.8283	0.01	0.12
	10	15076.	18.75	2.3985	2.3856	2.3677	0.01	0.11
50	15	15071.	15.50	2.8249	2.7994	2.7693	0.02	0.12
	20	15057.	10.54	3.1699	3.1255	3.0955	0.03	0.12
	25	15026.	21.26	3.4111	3.3778	3.3336	0.02	0.12
	rnd	15014.	22.89	2.1173	2.1010	2.0805	0.01	0.12
	1	26499.	460.55	1.1218	1.1105	1.0996	0.01	0.20
	5	30083.	63.61	1.4671	1.4528	1.4454	0.01	0.23
	10	30356.	65.87	1.7877	1.7781	1.7674	0.01	0.24
100	15	30433.	42.68	2.0628	2.0377	2.0223	0.01	0.23
	20	30444.	41.34	2.2736	2.2591	2.2436	0.01	0.24
	25	30443.	41.28	2.4855	2.4556	2.4327	0.02	0.24
	rnd	30017.	133.77	1.6091	1.6015	1.5939	0.01	0.23

Table A.9: Results of ADM (p442_SW)

		Adaptation		MBG				CPU
Per.	Sev.	avg	std	worst	avg	best	std	(sec)
	1	377.	18.78	6.4934	6.4200	6.3719	0.06	0.21
	5	458.	22.80	6.8564	6.8273	6.8002	0.03	0.27
	10	481.	13.16	7.1408	7.1338	7.1205	0.01	0.31
10	15	481.	25.32	7.4202	7.3518	7.2762	0.07	0.33
	20	501.	25.09	7.6041	7.5647	7.5411	0.03	0.35
	25	497.	17.91	7.6369	7.6348	7.6329	0.00	0.38
	rnd	489.	9.17	7.0242	7.0063	6.9922	0.02	0.29
	1	2494.	76.94	5.4427	5.4194	5.3884	0.03	1.24
	5	2843.	10.49	6.0095	5.9864	5.9708	0.02	1.47
	10	2936.	29.31	6.2442	6.2425	6.2398	0.00	1.57
50	15	2830.	40.06	6.3941	6.3588	6.3303	0.03	1.57
	20	2850.	52.44	6.5515	6.5114	6.4479	0.06	1.62
	25	2785.	45.24	6.6608	6.5968	6.5448	0.06	1.65
	rnd	2873.	35.17	6.1355	6.1238	6.1105	0.01	1.50
	1	5530.	193.42	5.0637	5.0556	5.0422	0.01	2.72
	5	6119.	69.73	5.7512	5.7317	5.7153	0.02	3.07
	10	6107.	27.00	5.9587	5.9274	5.8928	0.03	3.12
100	15	6040.	83.42	6.0760	6.0569	6.0224	0.03	3.17
	20	5964.	36.77	6.1757	6.1614	6.1383	0.02	3.20
	25	5960.	146.55	6.2517	6.2254	6.2007	0.03	3.26
	rnd	6033.	98.69	5.8308	5.8207	5.8138	0.01	3.07

Table A.10: Results of AHDM (p442_SW)

		Adaptation		MBG				CPU
Per.	Sev.	avg	std	worst	avg	best	std	(sec)
	1	1138.	7.09	3.6456	3.6271	3.6161	0.02	0.36
	5	1127.	2.95	5.2628	5.2345	5.2068	0.03	0.35
	10	1123.	3.31	6.2940	6.2088	6.1637	0.07	0.36
10	15	1110.	3.56	6.8911	6.7896	6.7370	0.09	0.37
	20	1076.	5.45	7.2052	7.1554	7.1070	0.05	0.37
	25	1065.	2.27	7.3662	7.3425	7.3163	0.03	0.38
	rnd	1130.	4.58	5.9248	5.8936	5.8673	0.03	0.36
	1	5886.	8.08	1.7203	1.7148	1.7055	0.01	1.71
	5	5930.	6.35	2.6298	2.6213	2.6048	0.01	1.73
	10	5921.	9.75	3.4151	3.3918	3.3678	0.02	1.73
50	15	5914.	3.90	4.0201	4.0137	4.0048	0.01	1.73
	20	5885.	13.72	4.5239	4.5164	4.5033	0.01	1.75
	25	5880.	25.35	4.9074	4.8994	4.8896	0.01	1.76
	rnd	5927.	14.97	3.0782	3.0634	3.0474	0.02	1.73
	1	11892.	32.06	1.3280	1.3099	1.2987	0.02	3.40
	5	11945.	23.07	1.9930	1.9788	1.9708	0.01	3.44
	10	11931.	6.40	2.5300	2.5212	2.5145	0.01	3.44
100	15	11907.	7.34	2.9773	2.9697	2.9652	0.01	3.45
	20	11888.	6.26	3.3734	3.3590	3.3358	0.02	3.45
	25	11895.	6.29	3.7218	3.6859	3.6514	0.04	3.46
	rnd	11933.	5.62	2.2899	2.2833	2.2710	0.01	3.44

Table A.11: Results of AIM (p442_SW)

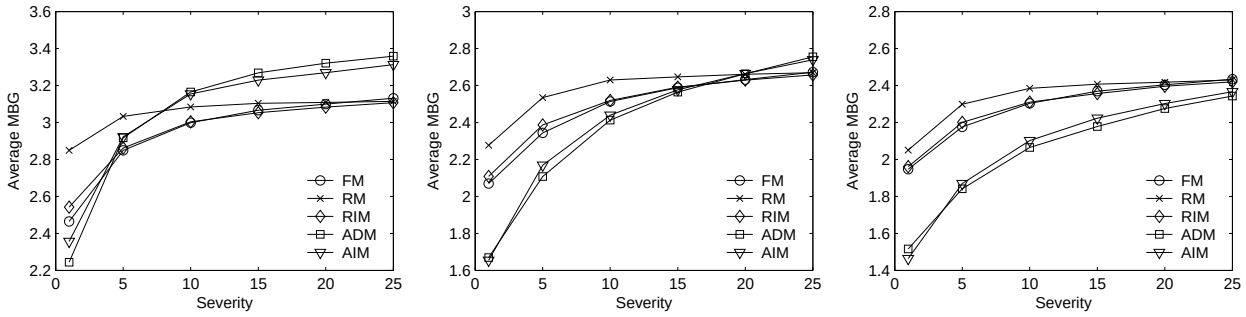
		Adaptation		MBG				CPU
Per.	Sev.	avg	std	worst	avg	best	std	(sec)
	1	443.	20.49	6.2371	6.1881	6.1080	0.07	0.21
	5	600.	5.99	6.9325	6.8646	6.8169	0.06	0.28
	10	654.	6.18	7.2245	7.1996	7.1717	0.03	0.31
10	15	657.	19.16	7.4225	7.3804	7.3159	0.06	0.31
	20	656.	15.07	7.5709	7.5445	7.5110	0.03	0.33
	25	671.	26.49	7.7202	7.6729	7.6411	0.04	0.32
	rnd	625.	32.50	7.0681	7.0539	7.0432	0.01	0.28
	1	2877.	101.86	5.1795	5.1498	5.1308	0.03	1.26
	5	3468.	73.54	5.8826	5.8089	5.7249	0.08	1.53
	10	3634.	27.69	6.2329	6.2045	6.1839	0.03	1.60
50	15	3719.	29.04	6.4098	6.4035	6.3929	0.01	1.65
	20	3731.	47.51	6.6175	6.5901	6.5574	0.03	1.66
	25	3700.	70.92	6.7467	6.7367	6.7308	0.01	1.71
	rnd	3537.	104.93	6.0578	6.0483	6.0385	0.01	1.57
	1	6142.	108.55	4.7536	4.6923	4.6022	0.08	2.66
	5	7117.	172.87	5.5110	5.4340	5.3817	0.07	3.10
	10	7429.	98.67	5.8303	5.8221	5.8107	0.01	3.21
100	15	7555.	78.67	6.0366	6.0341	6.0315	0.00	3.28
	20	7591.	110.99	6.2268	6.1978	6.1687	0.03	3.32
	25	7583.	54.26	6.3416	6.3255	6.3120	0.02	3.37
	rnd	7240.	88.52	5.6955	5.6784	5.6498	0.03	3.17

Table A.12: Results of AHIM (p442_SW)

		Adaptation		MBG				CPU
Per.	Sev.	avg	std	worst	avg	best	std	(sec)
	1	2954.	11.17	3.4193	3.3780	3.3336	0.04	0.36
	5	2940.	10.18	5.0326	4.9843	4.9348	0.05	0.35
	10	2944.	14.88	6.0024	5.9863	5.9719	0.02	0.36
10	15	2928.	8.77	6.6273	6.5774	6.5513	0.04	0.36
	20	2924.	14.00	6.9389	6.9248	6.9137	0.01	0.36
	25	2927.	48.48	7.1377	7.1193	7.0835	0.03	0.35
	rnd	2940.	14.40	5.7369	5.6434	5.5701	0.09	0.35
	1	16260.	15.09	1.5361	1.5322	1.5301	0.00	1.76
	5	16620.	12.00	2.4020	2.3978	2.3944	0.00	1.80
	10	16651.	7.60	3.1468	3.1428	3.1386	0.00	1.80
50	15	16654.	24.80	3.7847	3.7568	3.7187	0.03	1.82
	20	16611.	21.56	4.2403	4.2371	4.2314	0.01	1.81
	25	16601.	9.16	4.6184	4.5903	4.5746	0.02	1.83
	rnd	16646.	21.55	2.8590	2.8446	2.8342	0.01	1.80
	1	32522.	143.25	1.1784	1.1706	1.1661	0.01	3.48
	5	33626.	27.90	1.7657	1.7507	1.7405	0.01	3.59
	10	33767.	10.48	2.2978	2.2886	2.2781	0.01	3.60
100	15	33779.	54.34	2.7288	2.7208	2.7093	0.01	3.61
	20	33797.	25.86	3.1126	3.1027	3.0962	0.01	3.63
	25	33770.	42.46	3.4398	3.4228	3.3947	0.02	3.63
	rnd	33702.	3.65	2.0462	2.0364	2.0231	0.01	3.60

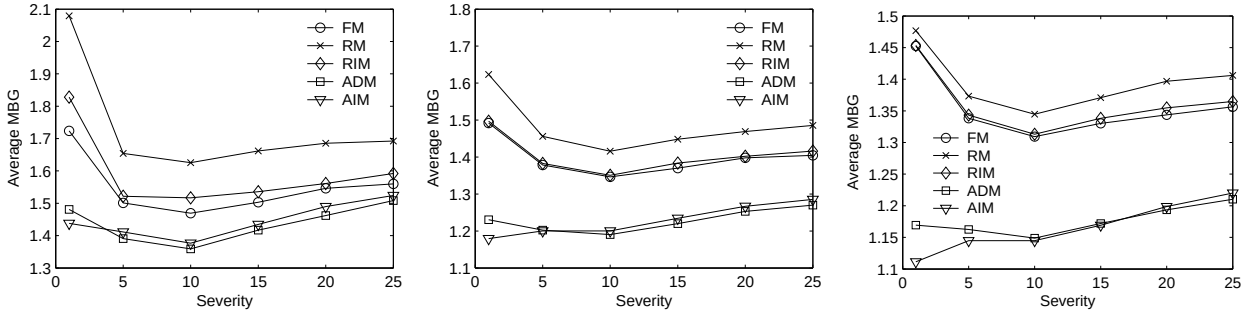
A.2 Comparison of evolutionary models

In this section, comparisons of the performance of evolutionary models are depicted graphically in terms of the mean best of generation (averaged over ten runs).



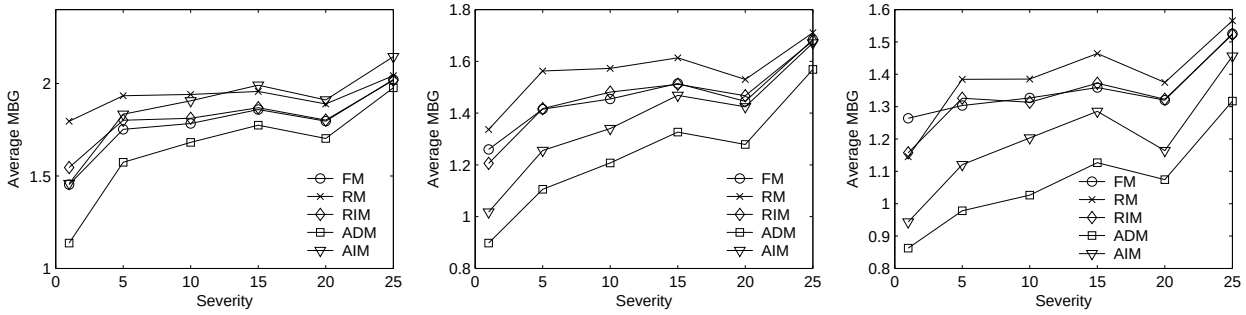
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. A.1: Comparison of evolutionary models (be52_VSM)



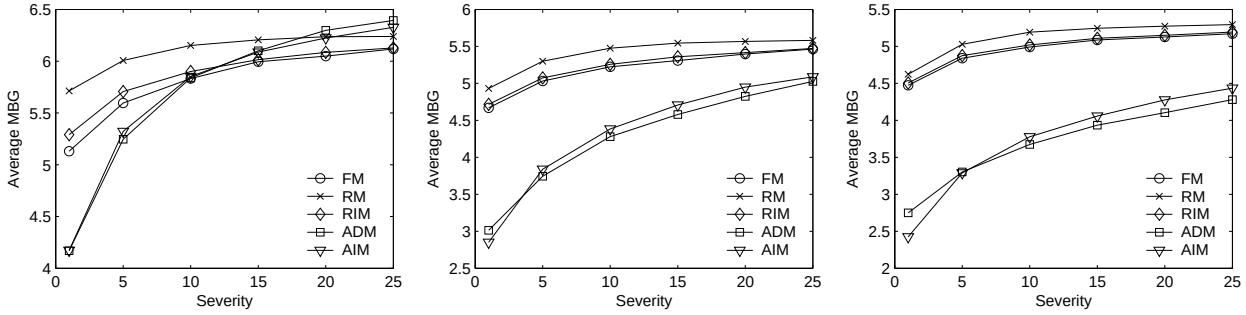
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. A.2: Comparison of evolutionary models (be52_ECM)



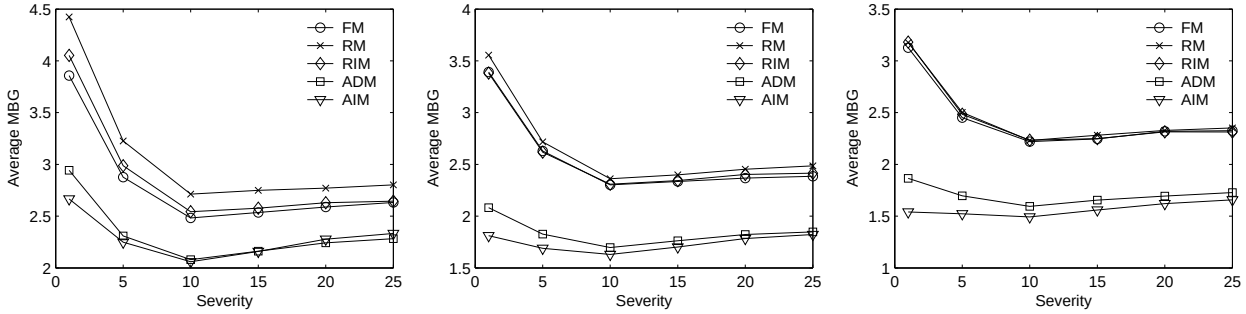
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. A.3: Comparison of evolutionary models (be52_IDM)



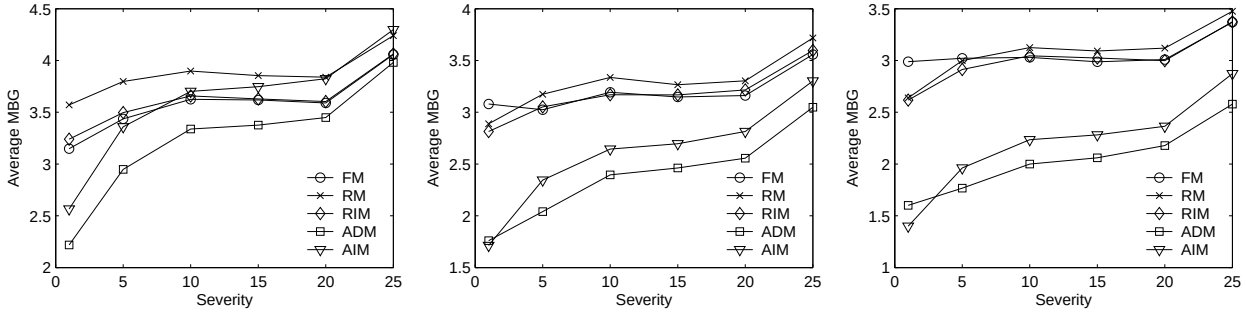
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. A.4: Comparison of evolutionary models (k100_VSM)



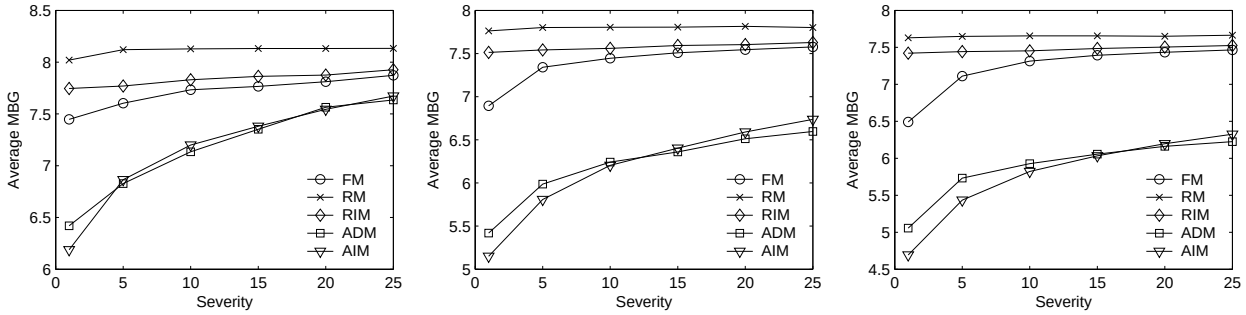
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. A.5: Comparison of evolutionary models (k100_ECM)



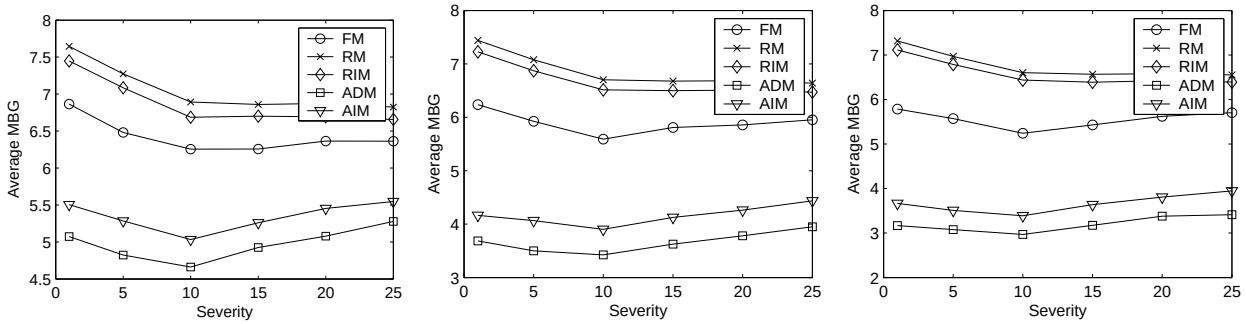
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. A.6: Comparison of evolutionary models (k100_IDM)



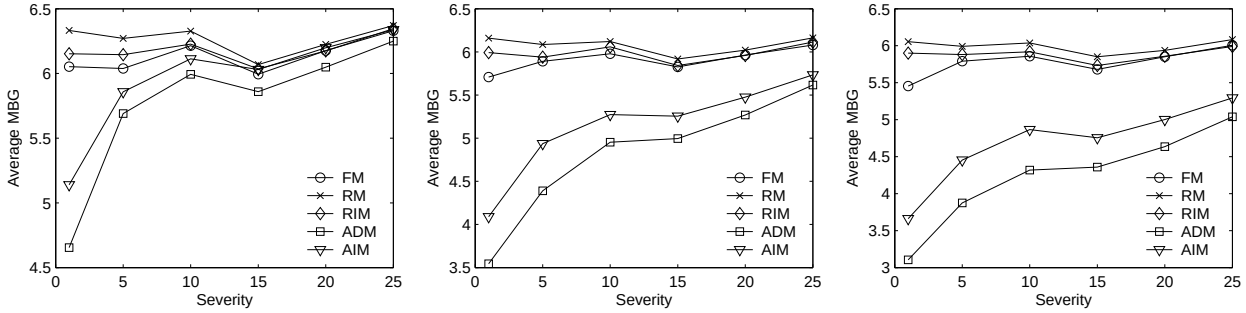
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. A.7: Comparison of evolutionary models (p442_VSM)



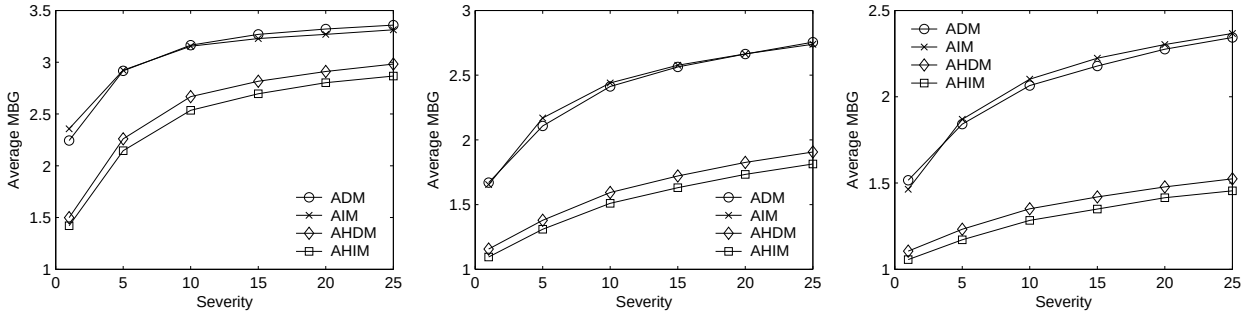
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. A.8: Comparison of evolutionary models (p442_ECM)



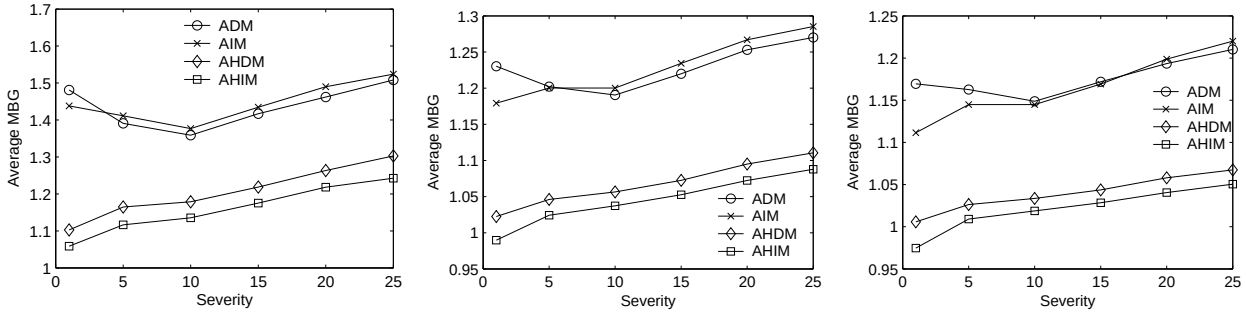
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. A.9: Comparison of evolutionary models (p442_IDM)



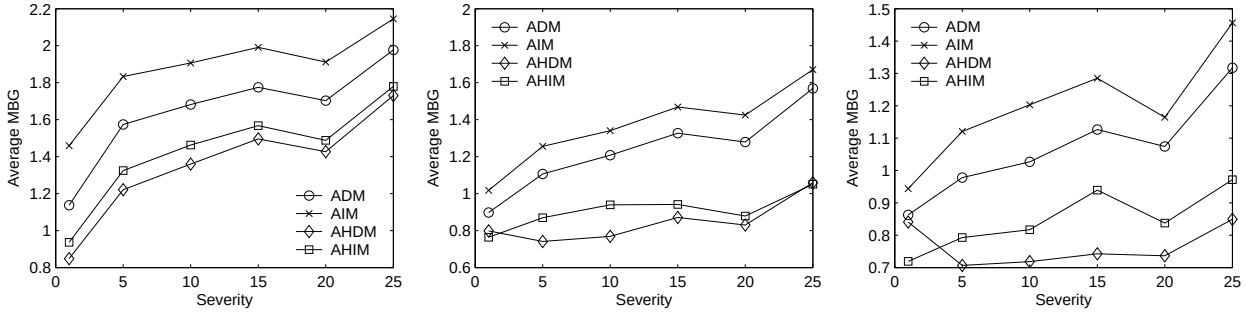
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. A.10: Comparison of hybridized models (be52_VSM)



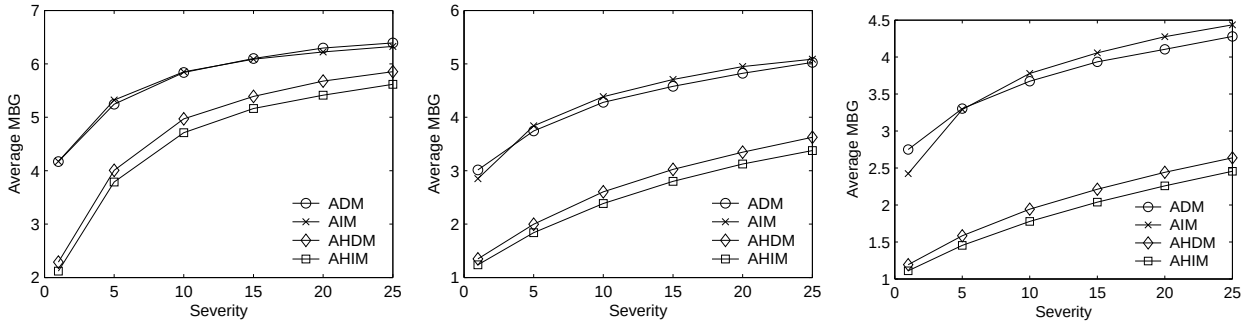
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. A.11: Comparison of hybridized models (be52_ECM)



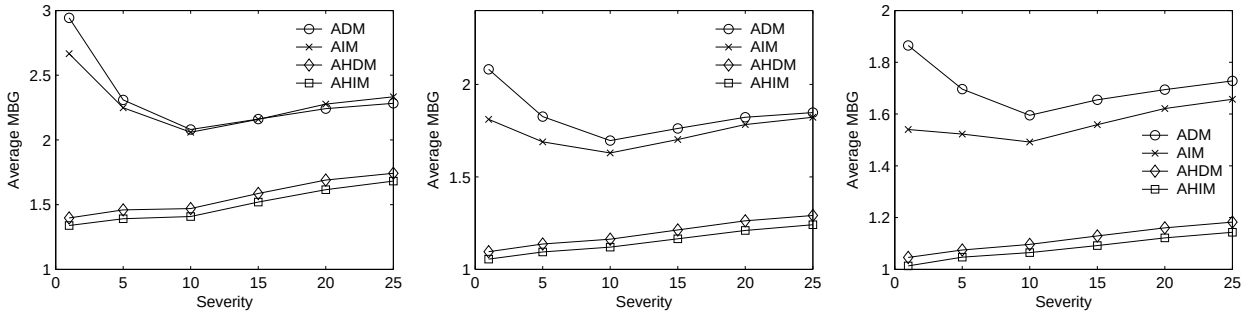
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. A.12: Comparison of hybridized models (be52_IDM)



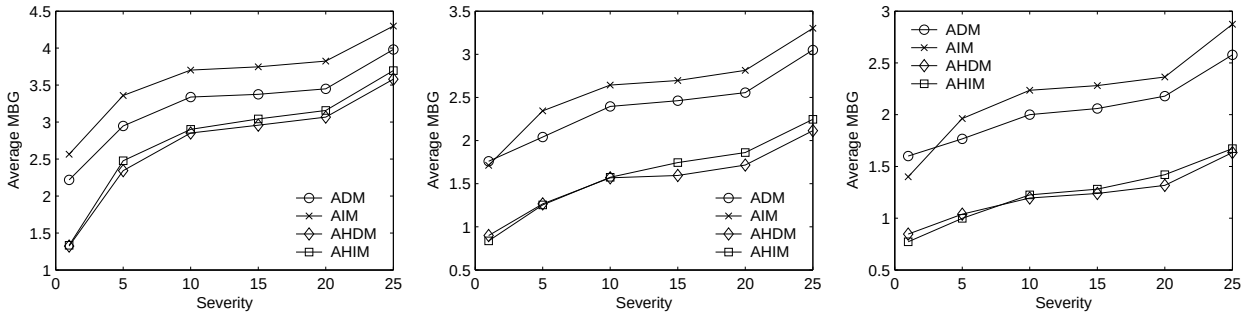
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. A.13: Comparison of hybridized models (k100_VSM)



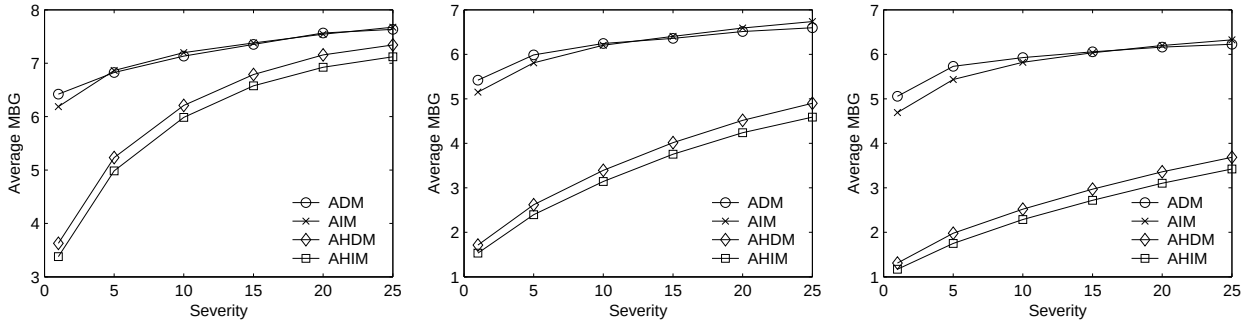
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. A.14: Comparison of hybridized models (k100_ECM)



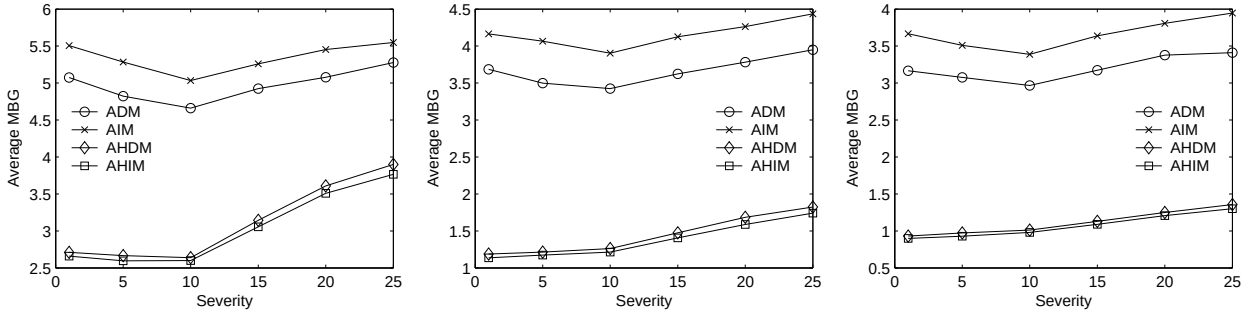
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. A.15: Comparison of hybridized models (k100_IDM)



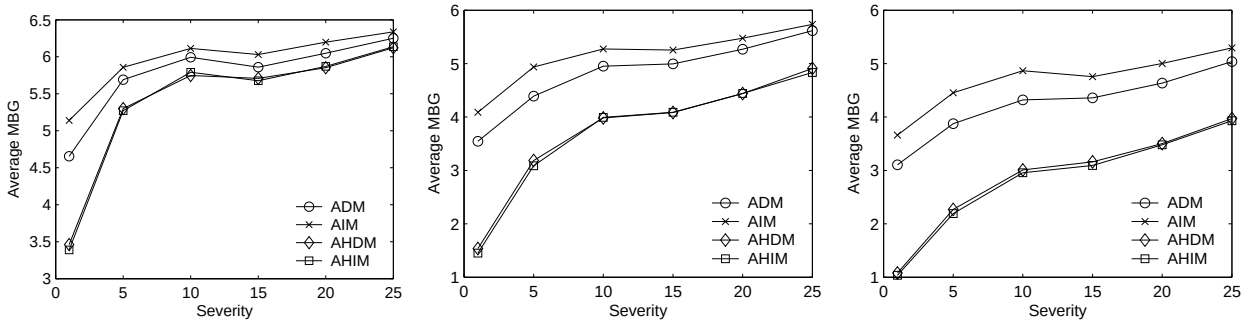
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. A.16: Comparison of hybridized models (p442_VSM)



Period = 10 generations Period = 50 generations Period = 100 generations

Fig. A.17: Comparison of hybridized models (p442_ECM)



Period = 10 generations Period = 50 generations Period = 100 generations

Fig. A.18: Comparison of hybridized models (p442_IDM)

A.3 Multiple post ANOVA tests

Statistical tables contain the results of multiple post ANOVA comparison tests which are obtained using a significance level of 5% as described in Section 4.5.3.

The entries in these tables are interpreted as follows. An entry of 1 signifies that the confidence interval for the difference in performance measures of the corresponding pair consists entirely of positive values, which indicates that the first model is inferior to the second model. Conversely, an entry of -1 signifies that the confidence interval for the corresponding pair consists entirely of negative values, which indicates that the first model is superior to the first one. An entry of 0 indicates that there is no significant difference between the two models.

Table A.13: Multiple comparison test of evolutionary models (be52-VSM)

period	10							100							1000						
severity	1	5	10	15	20	25	r	1	5	10	15	20	25	r	1	5	10	15	20	25	r
FM−RM	-1	-1	-1	0	0	0	-1	-1	-1	-1	0	0	0	-1	-1	-1	-1	0	0	0	-1
FM−RIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−ADM	1	0	-1	-1	-1	-1	-1	1	1	1	0	0	-1	1	1	1	1	1	1	1	1
FM−AIM	0	0	-1	-1	-1	-1	0	1	1	1	0	0	-1	1	1	1	1	1	1	1	1
RM−RIM	1	1	1	0	0	0	1	1	1	1	0	0	0	1	1	1	1	0	0	0	1
RM−ADM	1	0	-1	-1	-1	-1	0	1	1	1	1	0	-1	1	1	1	1	1	1	1	1
RM−AIM	1	0	-1	-1	-1	-1	0	1	1	1	1	0	-1	1	1	1	1	1	1	1	1
RIM−ADM	1	0	-1	-1	-1	-1	-1	1	1	1	0	0	-1	1	1	1	1	1	1	1	1
RIM−AIM	1	0	-1	-1	-1	-1	-1	1	1	1	0	0	-1	1	1	1	1	1	1	0	1
ADM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table A.14: Multiple comparison test of evolutionary models (be52-ECM)

period	10							100							1000						
severity	1	5	10	15	20	25	r	1	5	10	15	20	25	r	1	5	10	15	20	25	r
FM−RM	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	0
FM−RIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−ADM	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
FM−AIM	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−RIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	0	0
RM−ADM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−AIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−ADM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−AIM	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ADM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table A.15: Multiple comparison test of evolutionary models (be52-IDM)

period	10							100							1000						
severity	1	5	10	15	20	25	r	1	5	10	15	20	25	r	1	5	10	15	20	25	r
FM−RM	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−RIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−ADM	1	0	0	0	0	0	0	1	1	1	0	0	0	1	1	1	1	0	1	0	1
FM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
RM−RIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RM−ADM	1	1	1	0	0	0	1	1	1	1	1	1	0	1	1	1	1	1	1	0	1
RM−AIM	1	0	0	0	0	0	0	1	1	1	0	0	0	1	0	0	0	0	1	0	1
RIM−ADM	1	0	0	0	0	0	1	0	1	1	0	0	0	1	1	1	1	0	1	0	1
RIM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ADM−AIM	-1	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table A.16: Multiple comparison test of evolutionary models (k100-VSM)

period	10							100							1000						
severity	1	5	10	15	20	25	r	1	5	10	15	20	25	r	1	5	10	15	20	25	r
FM−RM	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	0	-1	0	-1	-1	-1	-1	-1	-1
FM−RIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−ADM	1	1	0	0	-1	-1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
FM−AIM	1	1	0	0	-1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−RIM	1	1	1	1	0	0	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1
RM−ADM	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−AIM	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−ADM	1	1	0	0	-1	-1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−AIM	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ADM−AIM	0	0	0	0	0	0	0	0	-1	0	-1	0	0	0	1	0	-1	-1	-1	-1	0

Table A.17: Multiple comparison test of evolutionary models (k100-ECM)

period	10							100							1000						
severity	1	5	10	15	20	25	r	1	5	10	15	20	25	r	1	5	10	15	20	25	r
FM−RM	-1	-1	-1	-1	-1	-1	-1	0	0	0	0	0	-1	0	0	0	0	0	0	0	0
FM−RIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−ADM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
FM−AIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−RIM	1	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RM−ADM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−AIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−ADM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−AIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ADM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	1

Table A.18: Multiple comparison test of evolutionary models (k100-IDM)

period	10							100							1000						
severity	1	5	10	15	20	25	r	1	5	10	15	20	25	r	1	5	10	15	20	25	r
FM−RM	-1	-1	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	0
FM−RIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
FM−ADM	1	1	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
FM−AIM	1	0	0	0	0	0	0	1	1	1	1	1	0	1	1	1	1	1	1	0	1
RM−RIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RM−ADM	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−AIM	1	1	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−ADM	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−AIM	1	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ADM−AIM	0	-1	0	-1	-1	0	-1	0	-1	0	0	-1	0	0	0	0	0	0	0	0	0

Table A.19: Multiple comparison test of evolutionary models (p442-VSM)

period	10							100							1000						
severity	1	5	10	15	20	25	r	1	5	10	15	20	25	r	1	5	10	15	20	25	r
FM−RM	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
FM−RIM	-1	-1	-1	0	-1	0	-1	-1	-1	-1	-1	0	0	-1	-1	-1	-1	-1	-1	-1	-1
FM−ADM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
FM−AIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−RIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−ADM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−AIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−ADM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−AIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ADM−AIM	1	0	-1	0	0	0	-1	1	1	0	0	-1	-1	1	1	1	1	0	0	-1	1

Table A.20: Multiple comparison test of evolutionary models (p442-ECM)

period	10							100							1000						
severity	1	5	10	15	20	25	r	1	5	10	15	20	25	r	1	5	10	15	20	25	r
FM−RM	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
FM−RIM	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
FM−ADM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
FM−AIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−RIM	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	0	1	1
RM−ADM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−AIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−ADM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−AIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ADM−AIM	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Table A.21: Multiple comparison test of evolutionary models (p442-IDM)

period	10							100							1000						
severity	1	5	10	15	20	25	r	1	5	10	15	20	25	r	1	5	10	15	20	25	r
FM−RM	-1	-1	0	0	0	0	-1	-1	-1	-1	0	0	0	-1	-1	-1	-1	-1	0	0	-1
FM−RIM	0	-1	0	0	0	0	0	-1	0	0	0	0	0	0	-1	0	0	0	0	0	0
FM−ADM	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
FM−AIM	1	1	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−RIM	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
RM−ADM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−AIM	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−ADM	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−AIM	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ADM−AIM	-1	-1	0	-1	-1	-1	0	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1

Table A.22: Multiple comparison test of hybridized models (be52-VSM)

period	10							100							1000						
severity	1	5	10	15	20	25	r	1	5	10	15	20	25	r	1	5	10	15	20	25	r
ADM−AIM	-1	0	0	0	0	0	0	0	-1	0	0	0	0	0	1	0	-1	0	0	0	0
ADM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ADM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AHDM−AHIM	0	0	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1

Table A.23: Multiple comparison test of hybridized models (be52-ECM)

period	10							100							1000						
severity	1	5	10	15	20	25	r	1	5	10	15	20	25	r	1	5	10	15	20	25	r
ADM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
ADM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ADM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AHDM−AHIM	0	0	1	0	0	1	1	0	0	1	0	0	0	1	0	0	0	0	0	0	0

Table A.24: Multiple comparison test of hybridized models (be52-IDM)

period	10							100							1000						
severity	1	5	10	15	20	25	r	1	5	10	15	20	25	r	1	5	10	15	20	25	r
ADM−AIM	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ADM−AHDM	1	1	0	0	0	0	1	0	1	1	1	1	1	1	0	0	0	1	0	1	0
ADM−AHIM	1	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0
AIM−AHDM	1	1	1	1	1	0	1	0	1	1	1	1	1	1	0	1	1	1	1	1	1
AIM−AHIM	1	1	1	1	1	0	1	0	1	1	1	1	1	0	1	1	1	1	0	1	0
AHDM−AHIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

where period in generations/shift, severity in steps/shift
r denotes a random severity between 1 and 25.

Table A.25: Multiple comparison test of hybridized models (k100-VSM)

period	10							100							1000						
severity	1	5	10	15	20	25	r	1	5	10	15	20	25	r	1	5	10	15	20	25	r
ADM−AIM	0	0	0	0	0	0	0	1	-1	-1	-1	0	0	-1	1	0	-1	-1	-1	-1	0
ADM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ADM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AHDM−AHIM	0	1	1	1	1	0	1	0	1	1	1	1	1	1	0	1	1	1	1	1	1

Table A.26: Multiple comparison test of hybridized models (k100-ECM)

period	10							100							1000						
severity	1	5	10	15	20	25	r	1	5	10	15	20	25	r	1	5	10	15	20	25	r
ADM−AIM	1	0	0	0	0	0	0	1	1	1	1	0	0	1	1	1	1	1	1	1	1
ADM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ADM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AHDM−AHIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table A.27: Multiple comparison test of hybridized models (k100-IDM)

period	10							100							1000						
severity	1	5	10	15	20	25	r	1	5	10	15	20	25	r	1	5	10	15	20	25	r
ADM−AIM	-1	0	0	-1	-1	0	-1	0	-1	0	0	0	0	0	0	0	0	0	0	0	0
ADM−AHDM	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ADM−AHIM	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AHDM−AHIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table A.28: Multiple comparison test of hybridized models (p442-VSM)

period	10							100							1000						
severity	1	5	10	15	20	25	r	1	5	10	15	20	25	r	1	5	10	15	20	25	r
ADM−AIM	1	0	0	0	0	0	0	1	1	0	0	-1	-1	1	1	1	1	0	0	-1	1
ADM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ADM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AHDM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table A.29: Multiple comparison test of hybridized models (p442-ECM)

period	10							100							1000						
severity	1	5	10	15	20	25	r	1	5	10	15	20	25	r	1	5	10	15	20	25	r
ADM−AIM	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
ADM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ADM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AHDM−AHIM	0	0	0	0	0	1	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0

Table A.30: Multiple comparison test of hybridized models (p442-IDM)

period	10							100							1000						
severity	1	5	10	15	20	25	r	1	5	10	15	20	25	r	1	5	10	15	20	25	r
ADM−AIM	-1	-1	0	-1	-1	-1	0	-1	-1	-1	-1	0	0	-1	-1	-1	-1	-1	-1	-1	-1
ADM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ADM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AHDM−AHIM	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

where period in generations/shift, severity in steps/shift
r denotes a random severity between 1 and 25.

Appendix B

FMS Results

This appendix lists results of experiments conducted on the dynamic FMS problem. The appendix contains sufficient detail for readers who want to replicate the work.

B.1 Individual strategy results

Details on the conducted experiments on individual models are given in this section. The number of evaluations needed to reach the best solution after an environmental change is averaged for all instances of the problem. The adaptation columns give the average and the standard deviation (over the conducted runs) of the number of evaluations per instance, including those of the local search when applicable. The MBG columns give the worst, the average, the best, and the standard deviation of the mean best of generation over the conducted runs. The last column gives the mean CPU time per instance (from an environmental change till the discovery of the best solution); $mean\ CPU = \frac{T}{R} \times \frac{E}{E_t}$, where T is the total CPU time, R is the number of conducted runs, E is the average number of evaluations to best per instance, and E_t is the average total number of evaluations per run.

Table B.1: Results of ADM (rnd1_MSM)

		Adaptation			MBG			CPU
Per.	Sev.	avg	std	worst	avg	best	std	(sec)
	1	727.	28.82	1.3665	1.3398	1.3180	0.02	0.01
	2	775.	66.95	1.4192	1.4008	1.3820	0.01	0.01
	3	852.	44.25	1.4259	1.4110	1.3941	0.01	0.01
10	5	808.	51.98	1.4845	1.4495	1.4272	0.02	0.01
	10	858.	95.17	1.4983	1.4725	1.4305	0.03	0.01
	rnd	780.	49.97	1.4141	1.3923	1.3749	0.02	0.01
	1	3927.	173.30	1.2206	1.2047	1.1894	0.01	0.06
	2	4250.	320.77	1.2627	1.2509	1.2365	0.01	0.06
	3	4154.	204.39	1.2684	1.2585	1.2463	0.01	0.06
50	5	4303.	330.08	1.3119	1.2889	1.2724	0.01	0.06
	10	4183.	464.11	1.3248	1.3056	1.2886	0.02	0.06
	rnd	4134.	206.80	1.2604	1.2407	1.2219	0.02	0.06
	1	8321.	430.17	1.1619	1.1565	1.1431	0.01	0.12
	2	8829.	204.67	1.2030	1.1938	1.1807	0.01	0.13
	3	8778.	348.42	1.2200	1.2034	1.1760	0.02	0.13
100	5	8586.	1336.73	1.2291	1.2133	1.2028	0.01	0.13
	10	8507.	654.40	1.2848	1.2385	1.1951	0.04	0.13
	rnd	8471.	371.40	1.2006	1.1764	1.1559	0.02	0.13

Table B.2: Results of AHDM (rnd1_MSM)

		Adaptation			MBG			CPU
Per.	Sev.	avg	std	worst	avg	best	std	(sec)
	1	897.	52.24	1.1283	1.1071	1.0874	0.02	0.01
	2	1195.	69.56	1.1789	1.1466	1.1225	0.02	0.01
	3	1125.	123.29	1.1662	1.1484	1.1352	0.01	0.01
10	5	1219.	99.89	1.2243	1.1916	1.1774	0.02	0.01
	10	1174.	212.53	1.2331	1.2086	1.1696	0.03	0.01
	rnd	1049.	73.53	1.1314	1.1177	1.0876	0.02	0.01
	1	1802.	284.03	1.0533	1.0309	1.0111	0.02	0.02
	2	2405.	498.92	1.0746	1.0585	1.0365	0.01	0.02
	3	2575.	563.81	1.0813	1.0653	1.0459	0.01	0.02
50	5	2896.	1003.30	1.1105	1.0806	1.0475	0.03	0.02
	10	2401.	681.06	1.1250	1.1008	1.0678	0.03	0.02
	rnd	1858.	629.89	1.0608	1.0513	1.0417	0.01	0.02
	1	3289.	504.33	1.0494	1.0194	.9980	0.02	0.03
	2	4263.	1317.73	1.0483	1.0315	.9976	0.02	0.04
	3	4155.	1590.94	1.0653	1.0547	1.0449	0.01	0.04
100	5	5084.	1572.73	1.1039	1.0702	1.0238	0.03	0.04
	10	6762.	2424.68	1.1027	1.0698	.9887	0.05	0.06
	rnd	3895.	1446.56	1.0433	1.0367	1.0161	0.01	0.03

Table B.3: Results of AIM (rnd1_MSM)

		Adaptation			MBG			CPU
Per.	Sev.	avg	std	worst	avg	best	std	(sec)
	1	815.	49.83	1.3158	1.2938	1.2759	0.01	0.01
	2	859.	16.85	1.3917	1.3819	1.3731	0.01	0.01
	3	891.	75.09	1.4144	1.4034	1.3915	0.01	0.01
10	5	850.	80.71	1.4601	1.4427	1.4218	0.01	0.01
	10	912.	76.59	1.4816	1.4712	1.4628	0.01	0.01
	rnd	851.	61.08	1.3735	1.3549	1.3406	0.01	0.01
	1	4340.	108.22	1.1360	1.1149	1.0924	0.02	0.06
	2	4627.	251.58	1.1829	1.1734	1.1603	0.01	0.07
	3	4618.	271.32	1.2050	1.1870	1.1708	0.02	0.07
50	5	4634.	177.44	1.2383	1.2170	1.1996	0.02	0.07
	10	4866.	345.55	1.2694	1.2502	1.2296	0.01	0.07
	rnd	4643.	162.45	1.1770	1.1525	1.1204	0.02	0.07
	1	9150.	324.43	1.0719	1.0617	1.0526	0.01	0.14
	2	9339.	105.36	1.1055	1.0928	1.0646	0.02	0.14
	3	9235.	547.56	1.1495	1.1174	1.1057	0.02	0.14
100	5	9804.	380.03	1.1622	1.1360	1.1254	0.02	0.15
	10	9692.	492.43	1.1841	1.1623	1.1444	0.01	0.14
	rnd	9030.	293.17	1.0941	1.0873	1.0724	0.01	0.13

Table B.4: Results of AHIM (rnd1_MSM)

		Adaptation			MBG			CPU
Per.	Sev.	avg	std	worst	avg	best	std	(sec)
	1	2350.	118.09	1.0365	1.0264	1.0121	0.01	0.02
	2	2607.	105.82	1.0781	1.0675	1.0503	0.01	0.02
	3	2803.	217.30	1.1284	1.1094	1.0947	0.01	0.02
10	5	2727.	212.68	1.1488	1.1376	1.1240	0.01	0.02
	10	3031.	112.11	1.1838	1.1712	1.1468	0.02	0.02
	rnd	2483.	184.13	1.0899	1.0717	1.0570	0.01	0.02
	1	13262.	711.11	.9534	.9409	.9264	0.01	0.11
	2	14193.	1019.11	.9788	.9675	.9629	0.01	0.11
	3	14819.	942.17	.9765	.9669	.9594	0.01	0.12
50	5	13970.	1743.19	1.0082	.9842	.9728	0.01	0.11
	10	14494.	2628.00	1.0097	.9937	.9751	0.01	0.12
	rnd	14224.	1273.42	.9668	.9570	.9480	0.01	0.11
	1	29397.	2165.13	.9164	.9090	.8957	0.01	0.23
	2	29047.	2000.52	.9255	.9136	.9047	0.01	0.23
	3	28580.	1927.21	.9320	.9232	.9102	0.01	0.23
100	5	29976.	2473.37	.9463	.9296	.9100	0.01	0.24
	10	27164.	4428.04	.9548	.9325	.9162	0.02	0.22
	rnd	27563.	2240.76	.9253	.9171	.9072	0.01	0.22

Table B.5: Results of ADM (gap1_MSM)

		Adaptation			MBG			CPU
Per.	Sev.	avg	std	worst	avg	best	std	(sec)
	1	753.	13.13	1.4053	1.3798	1.3546	0.02	0.09
	2	754.	24.10	1.5971	1.5816	1.5653	0.01	0.09
	3	741.	20.20	1.6788	1.6590	1.6338	0.02	0.09
10	5	758.	26.00	1.7786	1.7461	1.6975	0.03	0.10
	10	753.	31.58	1.8484	1.8253	1.7780	0.03	0.10
	rnd	755.	5.12	1.6401	1.5954	1.5516	0.04	0.09
	1	3774.	68.96	.8528	.8109	.7740	0.04	0.41
	2	3896.	48.27	1.0380	.9858	.9079	0.05	0.42
	3	3968.	19.18	1.1349	1.1030	1.0588	0.04	0.43
50	5	3988.	44.99	1.2932	1.2780	1.2454	0.02	0.44
	10	4015.	24.65	1.5033	1.4469	1.4055	0.04	0.45
	rnd	3884.	90.99	1.0319	1.0089	.9635	0.03	0.42
	1	7252.	189.67	.6728	.6524	.6360	0.01	0.77
	2	7593.	113.62	.8226	.7914	.7463	0.03	0.81
	3	7651.	302.99	.9214	.8610	.8038	0.04	0.83
100	5	7977.	99.09	1.0482	1.0237	1.0047	0.02	0.86
	10	7979.	98.57	1.2247	1.1954	1.1743	0.02	0.87
	rnd	7622.	371.17	.8287	.7897	.7278	0.04	0.82

Table B.6: Results of AHDM (gap1_MSM)

		Adaptation			MBG			CPU
Per.	Sev.	avg	std	worst	avg	best	std	(sec)
	1	1163.	25.56	.8336	.8075	.7810	0.02	0.06
	2	1177.	27.67	.9944	.9637	.9186	0.03	0.07
	3	1188.	49.02	1.0964	1.0483	1.0077	0.04	0.07
10	5	1201.	40.15	1.2431	1.1992	1.1670	0.03	0.07
	10	1220.	31.61	1.3980	1.3577	1.3122	0.04	0.08
	rnd	1156.	47.81	1.0151	.9606	.9178	0.04	0.06
	1	4985.	448.03	.5518	.5408	.5248	0.01	0.26
	2	5703.	203.83	.6273	.6077	.5810	0.02	0.30
	3	5830.	318.43	.6542	.6457	.6351	0.01	0.31
50	5	6126.	151.12	.7691	.7301	.7004	0.03	0.33
	10	5840.	379.96	.9042	.8495	.7961	0.04	0.32
	rnd	5754.	98.76	.6408	.6129	.5691	0.03	0.30
	1	8309.	942.71	.4767	.4510	.4063	0.03	0.43
	2	9859.	461.59	.5477	.5213	.4560	0.04	0.51
	3	10630.	732.22	.5584	.5428	.5325	0.01	0.56
100	5	11446.	632.88	.6339	.6072	.5570	0.03	0.60
	10	12121.	1305.84	.7482	.7009	.6846	0.03	0.64
	rnd	9797.	587.27	.5609	.5142	.4786	0.03	0.51

Table B.7: Results of AIM (gap1_MSM)

		Adaptation			MBG			CPU
Per.	Sev.	avg	std	worst	avg	best	std	(sec)
	1	789.	11.28	1.3878	1.3623	1.3214	0.02	0.09
	2	787.	21.99	1.6032	1.5758	1.5244	0.03	0.09
	3	781.	18.08	1.7413	1.6733	1.6223	0.05	0.10
10	5	786.	33.80	1.8008	1.7883	1.7728	0.01	0.10
	10	803.	18.81	1.8746	1.8483	1.8262	0.02	0.11
	rnd	793.	15.57	1.6148	1.5779	1.5347	0.03	0.09
	1	4099.	87.16	.8608	.8418	.8234	0.02	0.45
	2	4184.	169.52	1.0168	.9958	.9677	0.02	0.46
	3	4377.	75.73	1.1274	1.1070	1.0760	0.02	0.48
50	5	4370.	34.12	1.2700	1.2400	1.2166	0.02	0.49
	10	4342.	57.53	1.5032	1.4476	1.4122	0.04	0.50
	rnd	4291.	95.92	1.1069	1.0368	.9806	0.05	0.47
	1	8286.	146.05	.7279	.6928	.6566	0.03	0.90
	2	8400.	379.46	.8550	.7926	.7147	0.05	0.91
	3	8595.	333.84	.9128	.8906	.8713	0.01	0.94
100	5	8941.	66.47	1.0680	1.0321	.9924	0.03	0.98
	10	8870.	90.69	1.2499	1.1727	1.1364	0.05	0.98
	rnd	8637.	211.30	.8982	.8406	.8230	0.03	0.94

Table B.8: Results of AHIM (gap1_MSM)

		Adaptation			MBG			CPU
Per.	Sev.	avg	std	worst	avg	best	std	(sec)
	1	2796.	34.82	.7167	.6752	.6472	0.03	0.15
	2	2868.	38.66	.8837	.8249	.7947	0.04	0.15
	3	2906.	56.13	.9699	.9484	.9290	0.02	0.16
10	5	2935.	42.34	1.1282	1.1036	1.0569	0.03	0.16
	10	2868.	83.99	1.3117	1.2786	1.2177	0.04	0.16
	rnd	2897.	49.77	.9241	.8510	.8236	0.04	0.15
	1	11930.	947.74	.4957	.4523	.4189	0.03	0.62
	2	12577.	1686.69	.5385	.4996	.4684	0.03	0.65
	3	14463.	822.84	.5490	.5243	.5029	0.02	0.75
50	5	14250.	1155.78	.6413	.6082	.5763	0.03	0.74
	10	15311.	1002.54	.7652	.7025	.6791	0.04	0.80
	rnd	12429.	1721.15	.5193	.5060	.4844	0.01	0.64
	1	18911.	1014.74	.4117	.3943	.3834	0.01	0.98
	2	20889.	3539.95	.4588	.4413	.4109	0.02	1.08
	3	21604.	3869.70	.5032	.4758	.4429	0.03	1.12
100	5	23718.	3479.17	.5420	.5131	.4784	0.03	1.23
	10	27449.	3629.76	.6332	.5862	.5467	0.04	1.43
	rnd	18285.	1000.20	.4768	.4481	.4313	0.02	0.95

Table B.9: Results of ADM (gap2_MSM)

		Adaptation			MBG			CPU
Per.	Sev.	avg	std	worst	avg	best	std	(sec)
	1	731.	31.05	1.9834	1.9229	1.8608	0.05	0.04
	2	745.	18.42	2.2697	2.2202	2.1335	0.06	0.04
	3	749.	11.98	2.4828	2.3822	2.3085	0.08	0.04
10	5	758.	18.13	2.6732	2.6140	2.5609	0.05	0.05
	10	747.	31.66	2.8340	2.7466	2.6682	0.07	0.05
	rnd	752.	13.83	2.3162	2.2632	2.1897	0.05	0.04
	1	3408.	226.57	1.2636	1.1438	1.0367	0.10	0.18
	2	3670.	76.47	1.4217	1.3308	1.2215	0.09	0.20
	3	3804.	95.10	1.5761	1.4985	1.4030	0.08	0.20
50	5	3789.	49.06	1.8533	1.7187	1.6206	0.11	0.21
	10	3956.	59.17	2.0701	1.9861	1.9128	0.07	0.22
	rnd	3634.	179.88	1.5719	1.3712	1.2768	0.12	0.19
	1	6358.	482.71	1.0122	.9499	.8619	0.05	0.33
	2	6767.	298.13	1.1822	1.0803	.8839	0.12	0.36
	3	7233.	165.78	1.2671	1.1934	1.0206	0.10	0.38
100	5	7596.	228.02	1.5126	1.3890	1.1861	0.13	0.41
	10	7782.	303.00	1.7477	1.6168	1.4126	0.13	0.42
	rnd	7046.	325.96	1.1672	1.0797	.9662	0.10	0.37

Table B.10: Results of AHDM (gap2_MSM)

		Adaptation			MBG			CPU
Per.	Sev.	avg	std	worst	avg	best	std	(sec)
	1	1021.	63.25	1.1755	1.1121	1.0320	0.06	0.03
	2	1088.	71.79	1.4172	1.3126	1.2299	0.07	0.03
	3	1101.	70.07	1.5056	1.4250	1.3370	0.07	0.03
10	5	1181.	50.05	1.8081	1.6971	1.6311	0.07	0.03
	10	1222.	31.05	2.0370	1.9239	1.8553	0.09	0.04
	rnd	1118.	35.05	1.4195	1.3672	1.2922	0.06	0.03
	1	3574.	318.25	.9655	.8506	.7182	0.12	0.09
	2	4126.	729.18	1.0296	.9086	.6893	0.14	0.11
	3	4491.	366.95	1.0256	.9474	.8135	0.09	0.12
50	5	5669.	413.41	1.2042	1.0488	.8605	0.12	0.15
	10	5659.	238.47	1.2773	1.1915	1.0267	0.10	0.15
	rnd	4490.	386.03	1.0260	.9178	.8102	0.08	0.12
	1	4789.	1086.27	.8480	.7795	.6511	0.08	0.12
	2	6078.	1463.02	.9685	.8326	.7274	0.10	0.16
	3	7037.	729.10	.9069	.8269	.7193	0.07	0.18
100	5	9055.	438.71	1.0951	.9138	.7604	0.13	0.24
	10	7262.	1490.42	1.1130	.9632	.8140	0.12	0.19
	rnd	5784.	826.02	1.0059	.8285	.6956	0.12	0.15

Table B.11: Results of AIM (gap2_MSM)

		Adaptation			MBG			CPU
Per.	Sev.	avg	std	worst	avg	best	std	(sec)
	1	740.	39.21	2.0810	1.9088	1.7791	0.12	0.04
	2	780.	13.77	2.3078	2.2827	2.2393	0.03	0.05
	3	767.	37.53	2.5508	2.4108	2.3211	0.10	0.05
10	5	795.	9.76	2.7275	2.6806	2.6416	0.04	0.05
	10	728.	22.81	2.9153	2.8655	2.8097	0.04	0.05
	rnd	770.	38.38	2.3872	2.3103	2.1908	0.08	0.04
	1	3697.	110.55	1.2664	1.2034	1.1529	0.05	0.20
	2	3898.	169.47	1.4409	1.3821	1.2487	0.08	0.21
	3	4015.	68.60	1.6290	1.5760	1.4586	0.07	0.22
50	5	4225.	72.70	1.8759	1.7763	1.7247	0.06	0.23
	10	4242.	95.34	2.0946	2.0663	2.0435	0.02	0.24
	rnd	4094.	55.00	1.5054	1.4505	1.3867	0.04	0.22
	1	7036.	254.63	1.0749	1.0267	.9772	0.04	0.38
	2	7747.	380.14	1.2580	1.1634	1.0689	0.08	0.42
	3	8169.	257.73	1.3481	1.2638	1.2072	0.07	0.44
100	5	8423.	217.11	1.5486	1.4888	1.3863	0.07	0.46
	10	8386.	609.17	1.7487	1.6649	1.5615	0.07	0.46
	rnd	7581.	184.97	1.2570	1.2007	1.1329	0.06	0.41

Table B.12: Results of AHIM (gap2_MSM)

		Adaptation			MBG			CPU
Per.	Sev.	avg	std	worst	avg	best	std	(sec)
	1	2463.	137.22	1.0859	1.0329	.9957	0.04	0.06
	2	2672.	64.69	1.1941	1.1558	1.0998	0.05	0.07
	3	2712.	69.42	1.3606	1.2971	1.1754	0.07	0.07
10	5	2763.	162.23	1.6472	1.5270	1.4156	0.08	0.07
	10	2868.	98.22	1.8742	1.7964	1.7071	0.06	0.08
	rnd	2763.	118.75	1.2781	1.2003	1.0495	0.09	0.07
	1	7836.	1935.93	.7899	.7009	.5246	0.10	0.20
	2	9080.	1167.88	.9084	.7706	.5976	0.11	0.23
	3	10802.	1459.10	.8764	.8034	.6991	0.07	0.28
50	5	11774.	1846.57	1.0423	.9068	.8137	0.08	0.30
	10	11031.	2227.74	1.1007	1.0352	.9722	0.05	0.29
	rnd	10787.	1119.13	.9816	.8365	.7548	0.09	0.28
	1	11486.	2666.02	.7319	.6532	.5398	0.08	0.30
	2	15429.	2848.33	.7332	.6758	.6422	0.04	0.40
	3	16853.	2893.62	.7811	.7092	.6653	0.05	0.43
100	5	19024.	6491.95	.7968	.7743	.7243	0.03	0.49
	10	23701.	5688.94	.9212	.8635	.7392	0.07	0.61
	rnd	14917.	4627.47	.8116	.6992	.5197	0.11	0.38

Table B.13: Results of ADM (gap3_MSM)

		Adaptation			MBG			CPU
Per.	Sev.	avg	std	worst	avg	best	std	(sec)
	1	740.	17.34	2.1353	2.0573	2.0225	0.05	0.03
	2	763.	14.33	2.4173	2.3473	2.2460	0.06	0.03
	3	765.	11.61	2.5545	2.5150	2.4469	0.04	0.03
10	5	755.	32.38	2.6502	2.5682	2.4721	0.09	0.03
	10	767.	39.65	2.7463	2.6742	2.5847	0.06	0.03
	rnd	748.	21.98	2.4984	2.3973	2.2977	0.07	0.03
	1	3722.	111.33	1.3980	1.3550	1.3155	0.04	0.12
	2	3875.	39.23	1.6000	1.5562	1.4908	0.04	0.12
	3	3800.	103.85	1.7216	1.6916	1.6166	0.04	0.12
50	5	3916.	56.41	1.8666	1.8155	1.7625	0.04	0.13
	10	3893.	82.33	2.0446	1.9649	1.9017	0.06	0.13
	rnd	3859.	60.99	1.5960	1.5592	1.5414	0.02	0.12
	1	6619.	208.30	1.1590	1.1252	1.0808	0.04	0.21
	2	7215.	219.60	1.3322	1.2832	1.2445	0.04	0.23
	3	7213.	290.17	1.4039	1.3478	1.2775	0.05	0.23
100	5	7634.	216.58	1.5648	1.4985	1.4411	0.05	0.24
	10	7836.	111.56	1.7052	1.6578	1.5431	0.07	0.25
	rnd	7178.	396.01	1.3338	1.2918	1.2546	0.03	0.23

Table B.14: Results of AHDM (gap3_MSM)

		Adaptation			MBG			CPU
Per.	Sev.	avg	std	worst	avg	best	std	(sec)
	1	1081.	48.88	1.3408	1.2834	1.2283	0.05	0.02
	2	1105.	60.09	1.4637	1.4319	1.3887	0.04	0.02
	3	1050.	50.94	1.5865	1.5535	1.5262	0.03	0.02
10	5	1016.	119.42	1.7539	1.6768	1.6087	0.05	0.02
	10	1088.	125.72	1.9480	1.8434	1.7190	0.09	0.02
	rnd	1135.	48.64	1.5198	1.4833	1.4379	0.04	0.02
	1	3242.	253.45	1.0847	1.0052	.9429	0.05	0.05
	2	4051.	344.21	1.1239	1.1039	1.0768	0.02	0.06
	3	3797.	677.37	1.1749	1.1127	1.0185	0.06	0.06
50	5	4199.	788.84	1.2937	1.2222	1.1696	0.05	0.06
	10	4732.	793.37	1.5047	1.3216	1.1962	0.11	0.07
	rnd	3675.	242.33	1.1493	1.0701	1.0298	0.05	0.06
	1	5612.	809.40	1.0358	.9798	.9387	0.04	0.08
	2	5852.	762.15	1.0368	1.0164	.9876	0.02	0.09
	3	5420.	1024.55	1.0644	1.0320	.9719	0.04	0.08
100	5	6903.	2038.51	1.2033	1.1192	1.0312	0.07	0.11
	10	7734.	1599.45	1.2275	1.1689	1.0834	0.06	0.12
	rnd	6240.	941.88	1.0918	1.0357	.9670	0.05	0.09

Table B.15: Results of AIM (gap3_MSM)

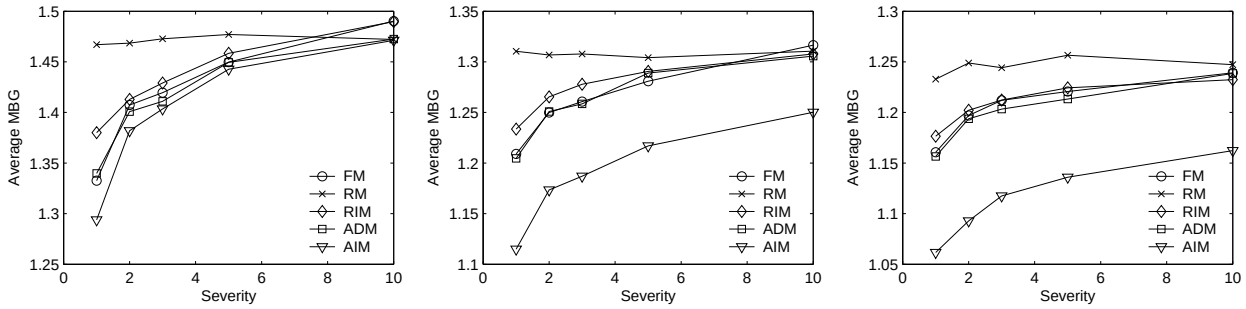
		Adaptation			MBG			CPU
Per.	Sev.	avg	std	worst	avg	best	std	(sec)
	1	776.	17.89	2.1079	2.0689	2.0298	0.03	0.03
	2	793.	8.34	2.3945	2.3586	2.3181	0.03	0.03
	3	785.	18.64	2.5083	2.4725	2.4129	0.04	0.03
10	5	786.	26.33	2.6275	2.5497	2.4999	0.05	0.03
	10	796.	8.97	2.7231	2.6689	2.6192	0.05	0.03
	rnd	791.	15.82	2.3845	2.3112	2.1979	0.07	0.03
	1	4018.	160.62	1.4591	1.3740	1.3193	0.06	0.13
	2	4049.	83.95	1.5614	1.4962	1.4493	0.04	0.13
	3	4191.	119.19	1.6558	1.6100	1.5419	0.05	0.14
50	5	4163.	160.46	1.8482	1.7477	1.6191	0.09	0.14
	10	4179.	115.57	2.0272	1.9351	1.8435	0.07	0.14
	rnd	4069.	135.19	1.6347	1.5613	1.4764	0.07	0.13
	1	7842.	279.90	1.2202	1.1800	1.1429	0.03	0.25
	2	8002.	412.63	1.3629	1.3175	1.2826	0.03	0.26
	3	7933.	352.63	1.4421	1.3924	1.3326	0.05	0.26
100	5	8184.	493.36	1.5362	1.4926	1.4121	0.05	0.27
	10	8393.	264.83	1.6828	1.6125	1.5126	0.07	0.27
	rnd	8077.	336.30	1.3489	1.2979	1.2514	0.04	0.26

Table B.16: Results of AHIM (gap3_MSM)

		Adaptation			MBG			CPU
Per.	Sev.	avg	std	worst	avg	best	std	(sec)
	1	2648.	61.09	1.1250	1.1092	1.0920	0.01	0.04
	2	2770.	91.76	1.3326	1.3186	1.2980	0.01	0.04
	3	2828.	140.10	1.4686	1.3889	1.3400	0.05	0.04
10	5	2878.	58.35	1.5584	1.5290	1.4973	0.02	0.04
	10	2954.	56.27	1.6957	1.6759	1.6359	0.03	0.05
	rnd	2812.	122.92	1.3525	1.3176	1.2850	0.03	0.04
	1	8759.	1042.89	.8602	.8354	.7849	0.03	0.13
	2	9607.	2784.30	.9764	.9093	.8477	0.05	0.14
	3	10316.	715.38	1.0411	.9611	.9012	0.06	0.16
50	5	9995.	906.21	1.0374	.9995	.9515	0.03	0.15
	10	8648.	1640.83	1.1455	1.1079	1.0732	0.03	0.13
	rnd	8769.	1124.89	.9749	.9235	.8774	0.04	0.13
	1	15229.	2489.42	.8348	.7794	.7307	0.04	0.23
	2	17372.	2368.61	.8722	.8250	.7381	0.05	0.26
	3	15888.	6618.43	.9381	.8760	.7974	0.05	0.24
100	5	21418.	5246.58	1.0057	.9427	.8447	0.06	0.32
	10	24104.	6905.96	1.0221	.9816	.9293	0.03	0.36
	rnd	15263.	3732.81	.8729	.8296	.7646	0.04	0.23

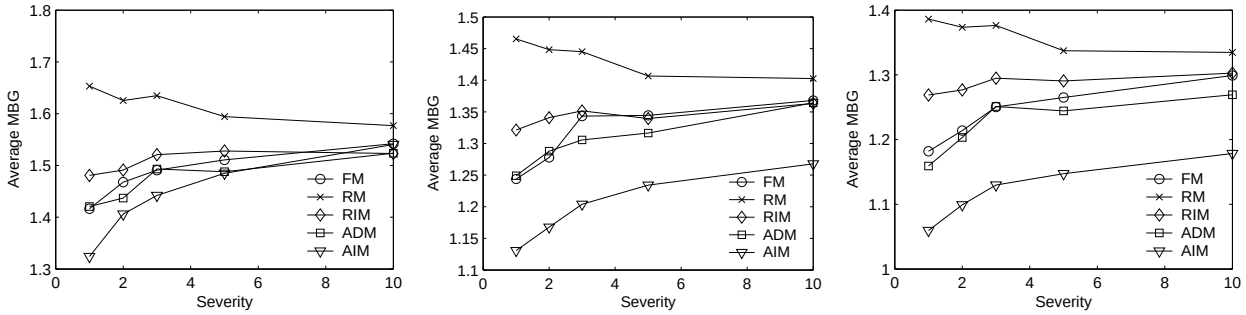
B.2 Comparison of evolutionary models

In this section, comparisons of the performance of evolutionary models are depicted graphically in terms of the mean best of generation (averaged over ten runs).



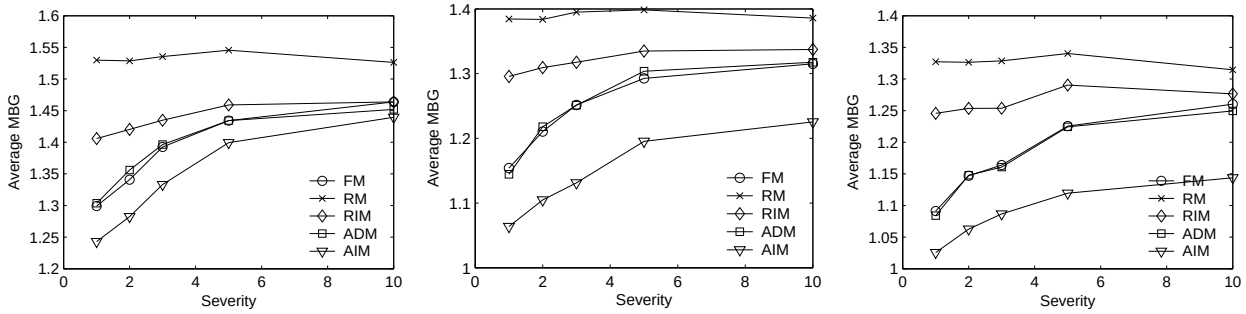
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. B.1: Comparison of evolutionary models (rnd1_MSM)



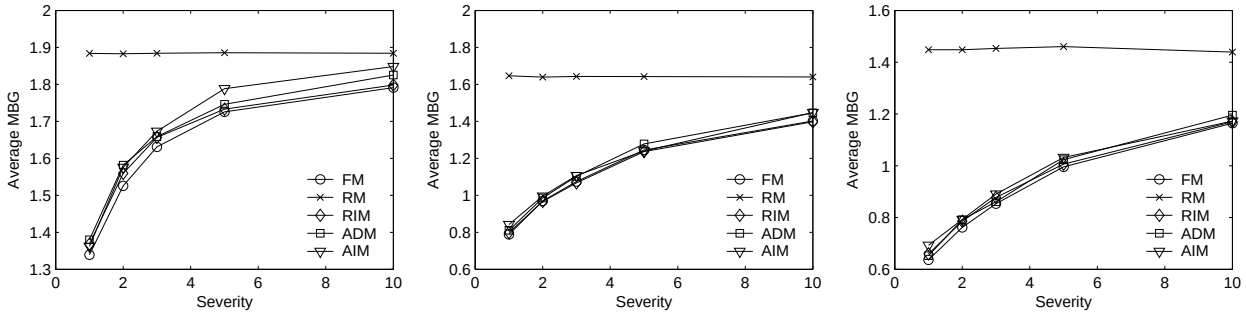
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. B.2: Comparison of evolutionary models (rnd1_MDM)



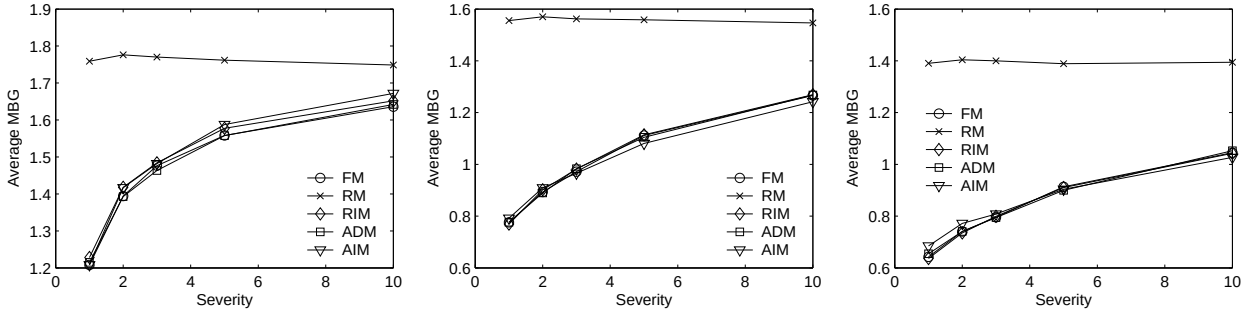
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. B.3: Comparison of evolutionary models (rnd1_PAM)



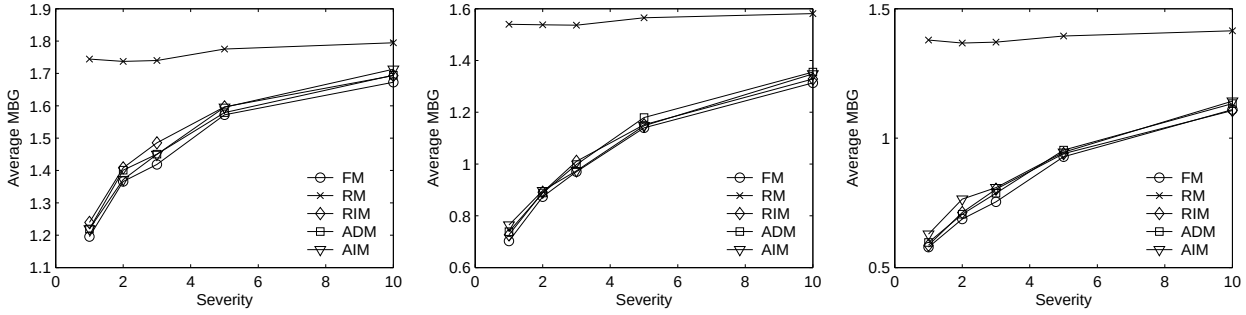
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. B.4: Comparison of evolutionary models (gap1_MSM)



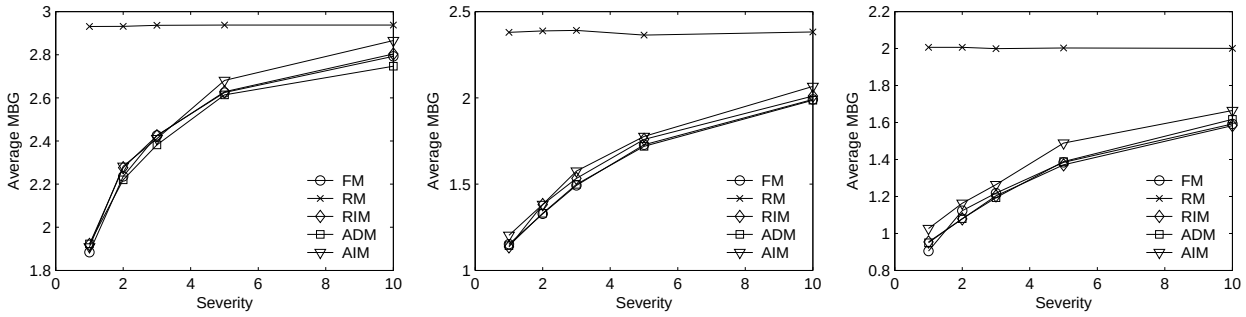
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. B.5: Comparison of evolutionary models (gap1_MDM)



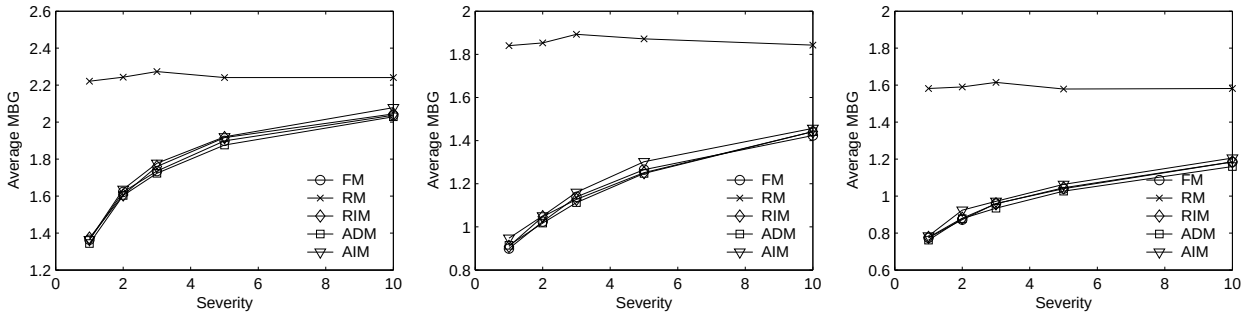
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. B.6: Comparison of evolutionary models (gap1_PAM)



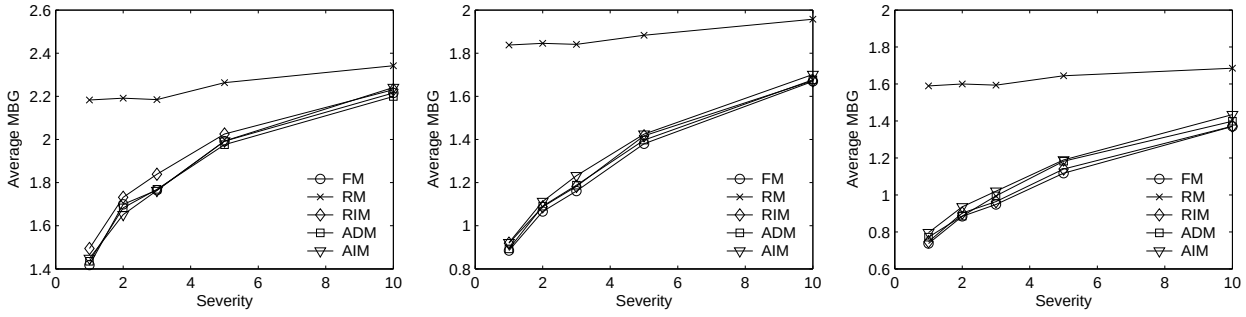
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. B.7: Comparison of evolutionary models (gap2_MSM)



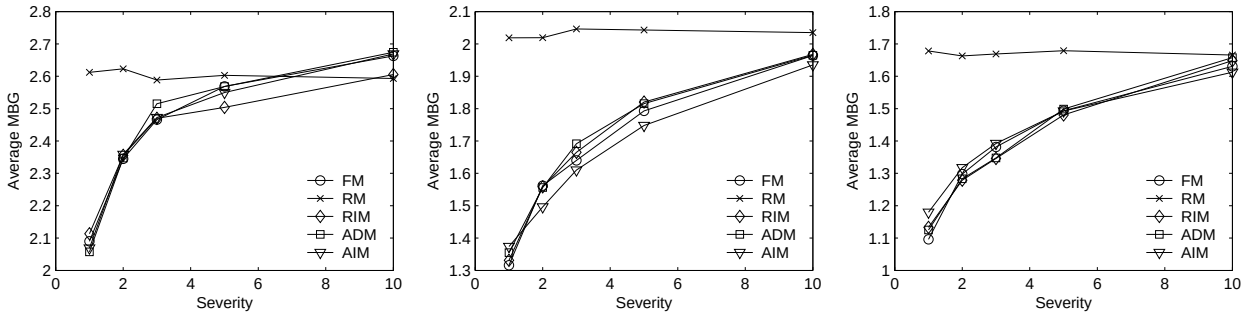
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. B.8: Comparison of evolutionary models (gap2_MDM)



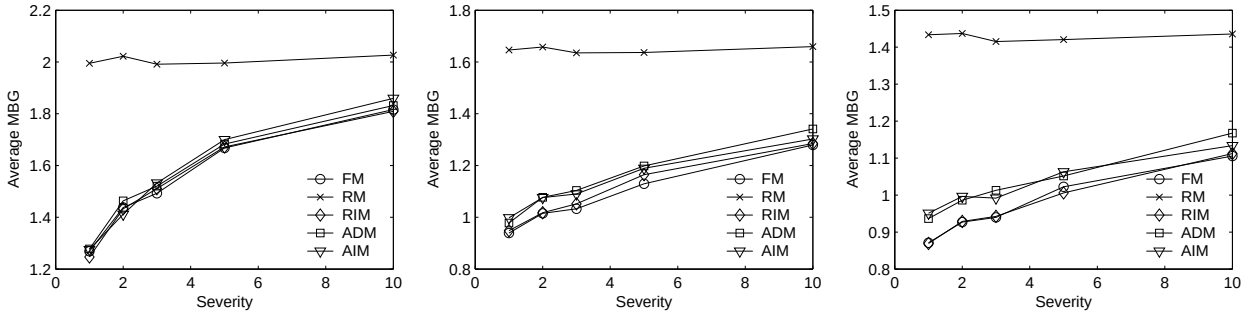
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. B.9: Comparison of evolutionary models (gap2_PAM)



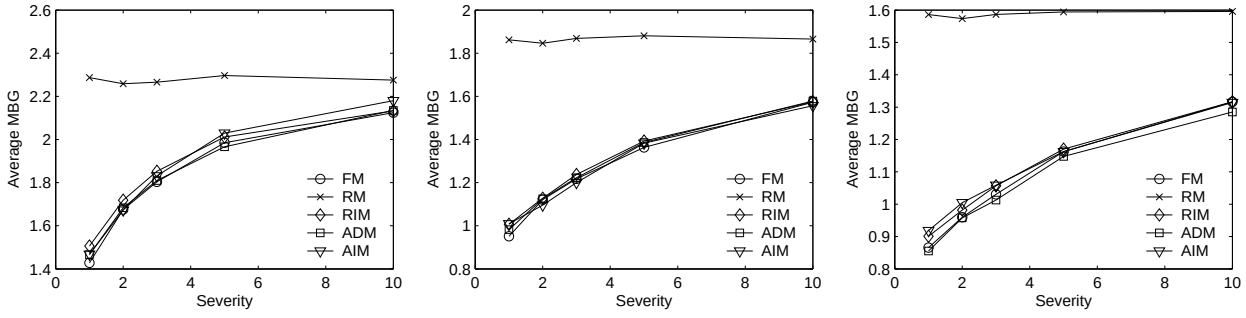
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. B.10: Comparison of evolutionary models (gap3_MSM)



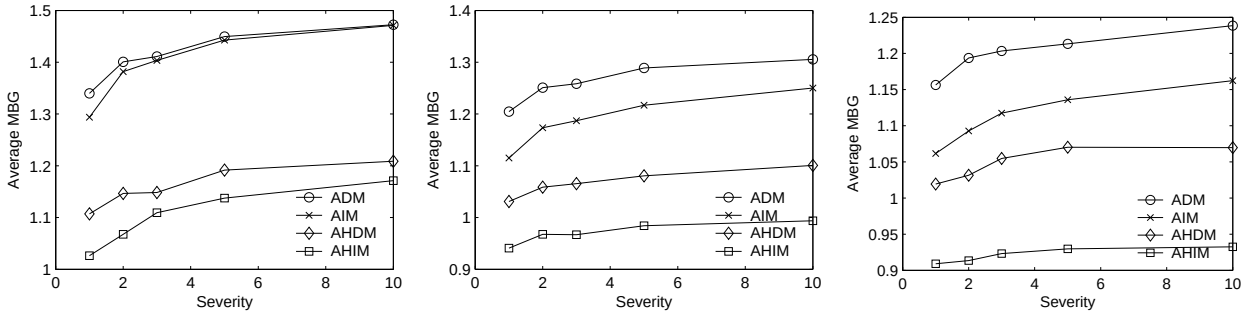
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. B.11: Comparison of evolutionary models (gap3_MDM)



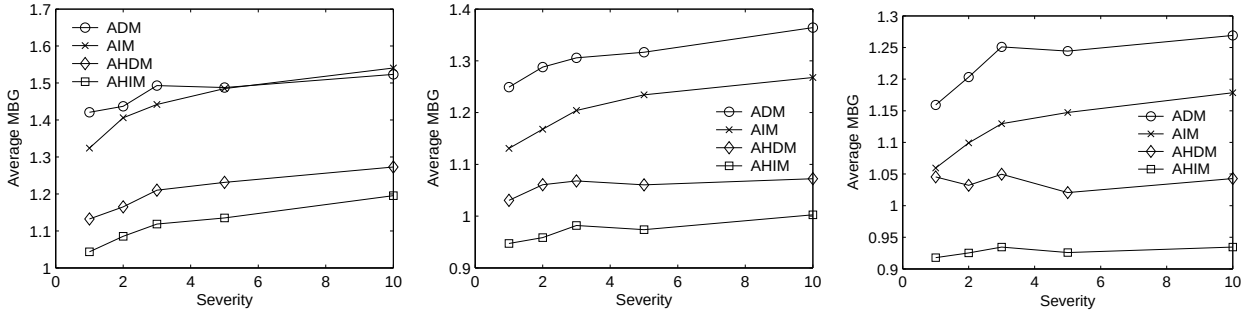
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. B.12: Comparison of evolutionary models (gap3_PAM)



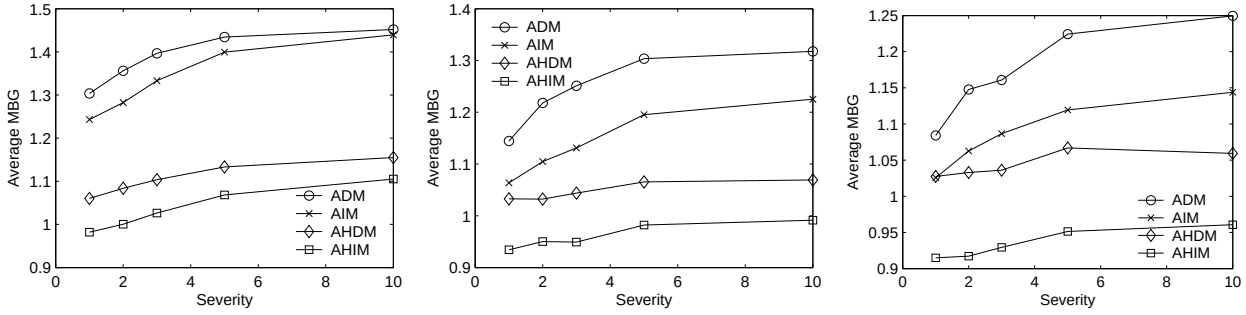
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. B.13: Comparison of hybridized models (rnd1_MSM)



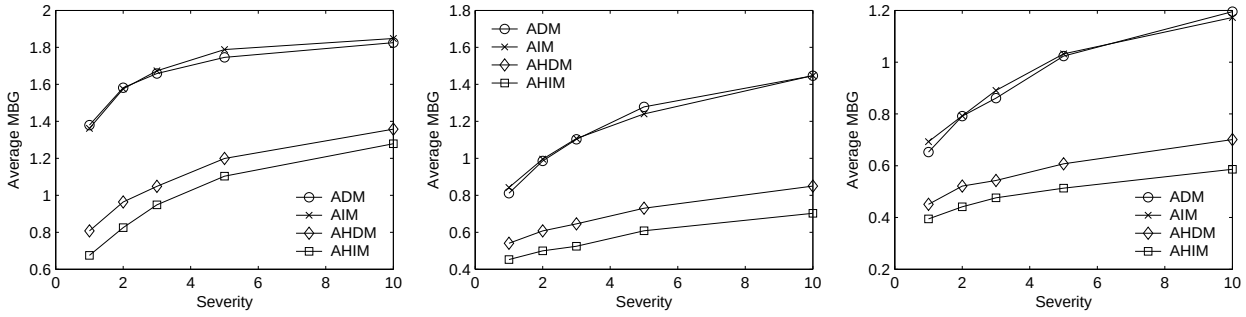
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. B.14: Comparison of hybridized models (rnd1_MDM)



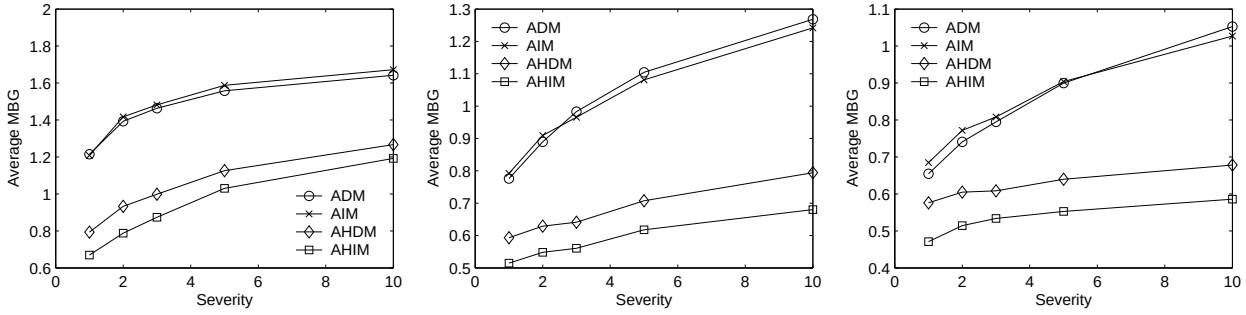
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. B.15: Comparison of hybridized models (rnd1_PAM)



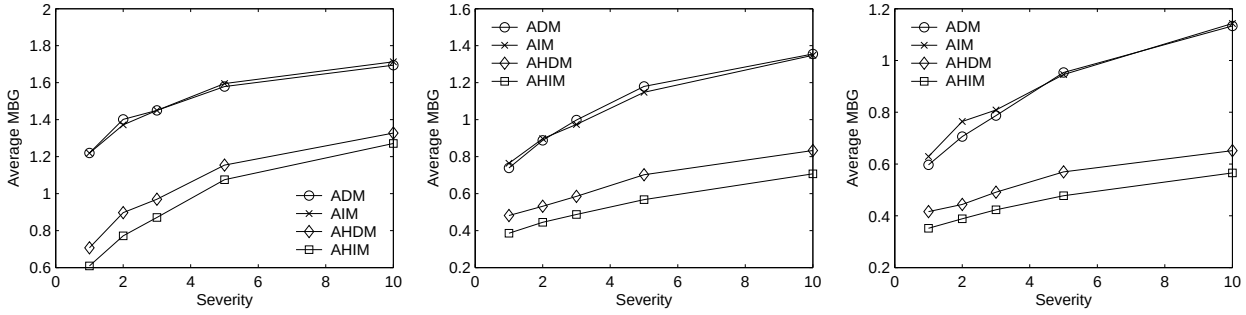
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. B.16: Comparison of hybridized models (gap1_MSM)



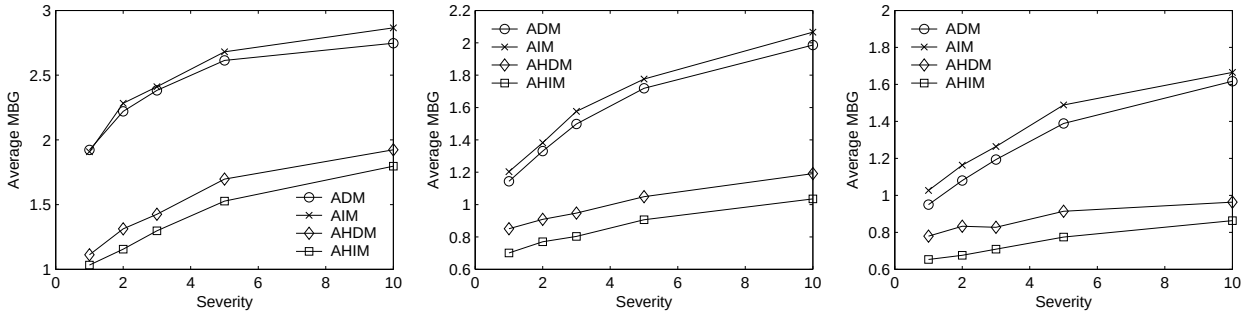
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. B.17: Comparison of hybridized models (gap1_MDM)



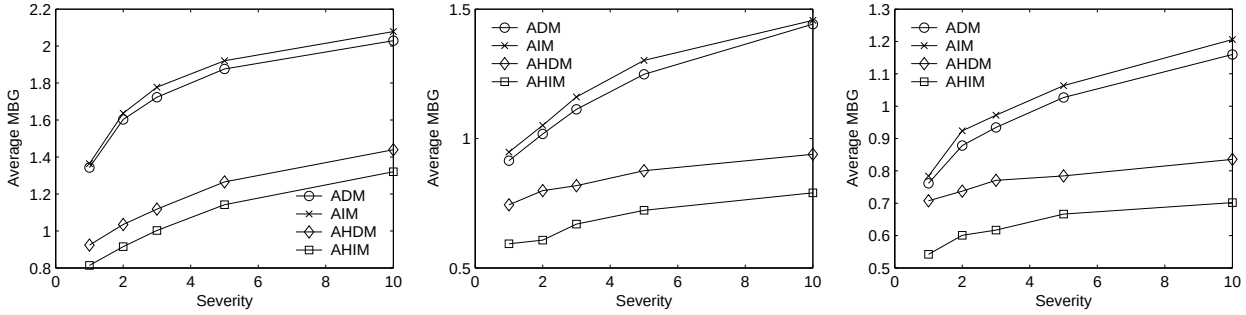
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. B.18: Comparison of hybridized models (gap1_PAM)



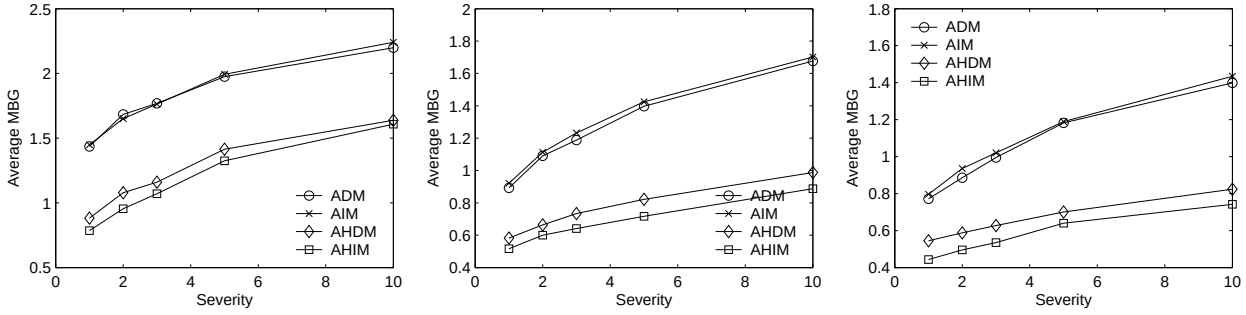
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. B.19: Comparison of hybridized models (gap2_MSM)



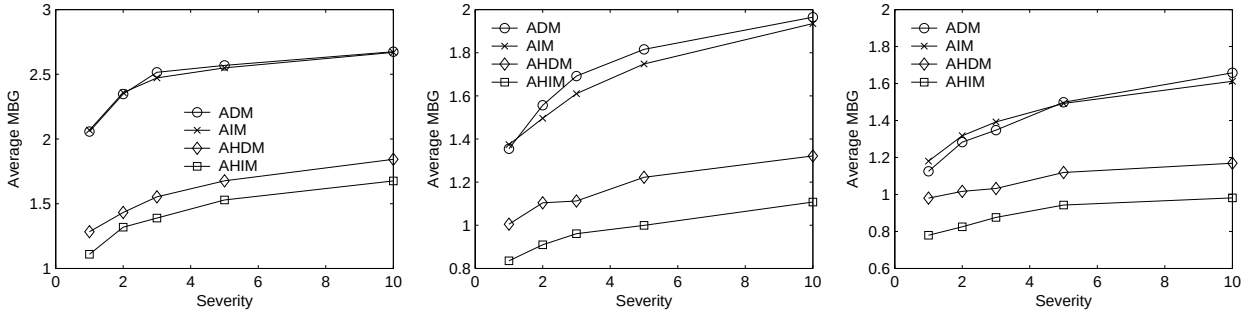
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. B.20: Comparison of hybridized models (gap2_MDM)



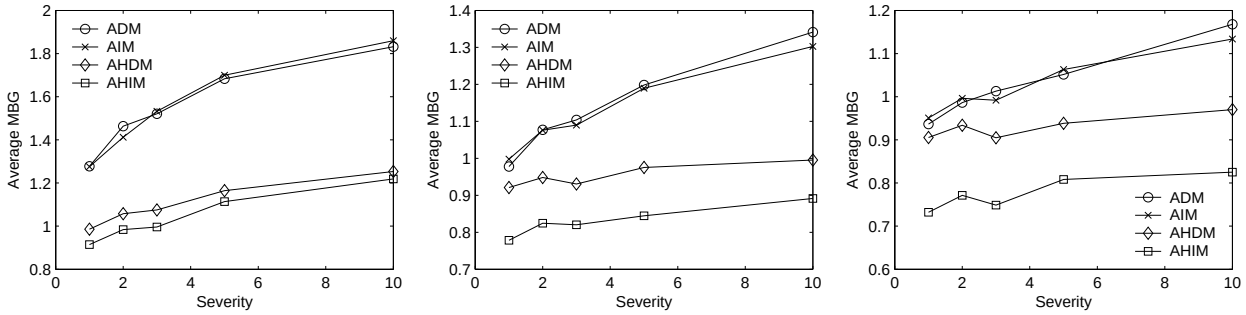
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. B.21: Comparison of hybridized models (gap2_PAM)



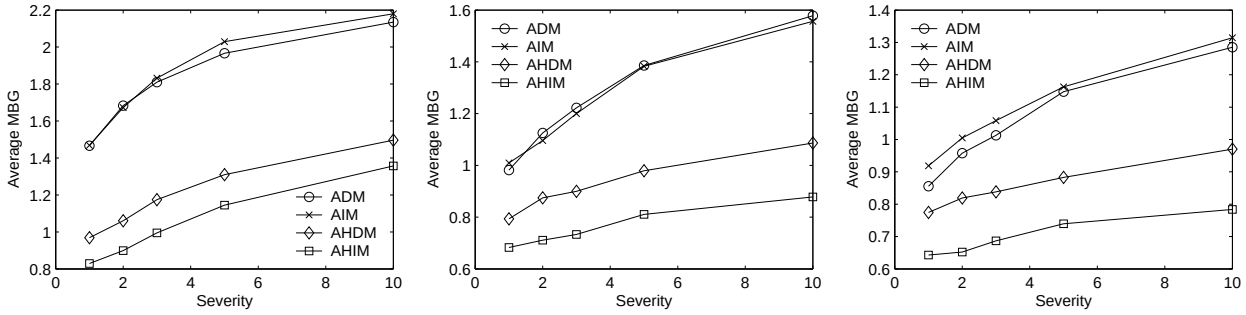
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. B.22: Comparison of hybridized models (gap3_MSM)



Period = 10 generations Period = 50 generations Period = 100 generations

Fig. B.23: Comparison of hybridized models (gap3_MDM)



Period = 10 generations Period = 50 generations Period = 100 generations

Fig. B.24: Comparison of hybridized models (gap3_PAM)

B.3 Multiple post ANOVA tests

Statistical tables contain the results of multiple post ANOVA comparison tests which are obtained using a significance level of 5% as described in Section 4.5.3.

The entries in these tables are interpreted as follows. An entry of 1 signifies that the confidence interval for the difference in performance measures of the corresponding pair consists entirely of positive values, which indicates that the first model is inferior to the second model. Conversely, an entry of -1 signifies that the confidence interval for the corresponding pair consists entirely of negative values, which indicates that the first model is superior to the first one. An entry of 0 indicates that there is no significant difference between the two models.

Table B.17: Multiple comparison test of evolutionary models (rnd1-MSM)

period	10						100						1000					
severity	1	2	3	5	10	r	1	2	3	5	10	r	1	2	3	5	10	r
FM−RM	-1	-1	-1	0	0	-1	-1	-1	-1	0	0	-1	-1	-1	0	0	0	-1
FM−RIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−ADM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−AIM	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
RM−RIM	1	1	1	0	0	1	1	1	0	0	0	1	1	1	0	0	0	1
RM−ADM	1	1	1	0	0	1	1	1	1	0	0	1	1	1	0	0	0	1
RM−AIM	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−ADM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RIM−AIM	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	1
ADM−AIM	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1

Table B.18: Multiple comparison test of evolutionary models (rnd1-MDM)

period	10						100						1000					
severity	1	2	3	5	10	r	1	2	3	5	10	r	1	2	3	5	10	r
FM−RM	-1	-1	-1	-1	0	-1	-1	-1	-1	0	0	-1	-1	-1	-1	0	0	-1
FM−RIM	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0
FM−ADM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−AIM	1	0	0	0	0	0	1	1	1	0	1	1	1	1	1	1	1	1
RM−RIM	1	1	1	1	0	1	1	1	1	0	0	1	1	1	1	0	0	0
RM−ADM	1	1	1	1	0	1	1	1	1	0	0	1	1	1	1	1	0	1
RM−AIM	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−ADM	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1
RIM−AIM	1	1	1	0	0	1	1	1	1	0	1	1	1	1	1	1	1	1
ADM−AIM	1	0	0	0	0	0	1	1	1	0	1	1	1	1	1	0	0	0

Table B.19: Multiple comparison test of evolutionary models (rnd1-PAM)

period	10						100						1000					
severity	1	2	3	5	10	r	1	2	3	5	10	r	1	2	3	5	10	r
FM−RM	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	0	-1
FM−RIM	-1	0	0	0	0	-1	-1	-1	0	0	0	-1	-1	-1	-1	-1	0	-1
FM−ADM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−AIM	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
RM−RIM	1	1	1	1	1	1	1	1	0	0	0	1	1	1	1	0	0	1
RM−ADM	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	0	1
RM−AIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−ADM	1	0	0	0	0	0	1	1	0	0	0	1	1	1	1	1	0	1
RIM−AIM	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
ADM−AIM	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1

Table B.20: Multiple comparison test of evolutionary models (gap1-MSM)

period	10						100						1000					
severity	1	2	3	5	10	r	1	2	3	5	10	r	1	2	3	5	10	r
FM−RM	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
FM−RIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−ADM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RM−RIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−ADM	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−AIM	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−ADM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RIM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ADM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table B.21: Multiple comparison test of evolutionary models (gap1-MDM)

period	10						100						1000					
severity	1	2	3	5	10	r	1	2	3	5	10	r	1	2	3	5	10	r
FM−RM	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
FM−RIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−ADM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RM−RIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−ADM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−AIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−ADM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RIM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ADM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table B.22: Multiple comparison test of evolutionary models (gap1-PAM)

period	10						100						1000					
severity	1	2	3	5	10	r	1	2	3	5	10	r	1	2	3	5	10	r
FM−RM	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
FM−RIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−ADM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	-1	0	0	0
RM−RIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−ADM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−AIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−ADM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RIM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0
ADM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0

Table B.23: Multiple comparison test of evolutionary models (gap2-MSM)

period	10						100						1000					
severity	1	2	3	5	10	r	1	2	3	5	10	r	1	2	3	5	10	r
FM−RM	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
FM−RIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−ADM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RM−RIM	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−ADM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−AIM	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−ADM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RIM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ADM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table B.24: Multiple comparison test of evolutionary models (gap2-MDM)

period	10						100						1000					
severity	1	2	3	5	10	r	1	2	3	5	10	r	1	2	3	5	10	r
FM−RM	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
FM−RIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−ADM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RM−RIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−ADM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−AIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−ADM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RIM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ADM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table B.25: Multiple comparison test of evolutionary models (gap2-PAM)

period	10						100						1000					
severity	1	2	3	5	10	r	1	2	3	5	10	r	1	2	3	5	10	r
FM−RM	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
FM−RIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−ADM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RM−RIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−ADM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−AIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−ADM	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
RIM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ADM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table B.26: Multiple comparison test of evolutionary models (gap3-MSM)

period	10						100						1000					
severity	1	2	3	5	10	r	1	2	3	5	10	r	1	2	3	5	10	r
FM−RM	-1	-1	0	0	0	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	0	-1
FM−RIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−ADM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RM−RIM	1	1	0	0	0	1	1	1	1	1	0	1	1	1	1	1	0	1
RM−ADM	1	1	0	0	0	0	1	1	1	1	0	1	1	1	1	1	0	1
RM−AIM	1	1	0	0	0	1	1	1	1	1	0	1	1	1	1	1	0	1
RIM−ADM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RIM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ADM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table B.27: Multiple comparison test of evolutionary models (gap3-MDM)

period	10						100						1000					
severity	1	2	3	5	10	r	1	2	3	5	10	r	1	2	3	5	10	r
FM−RM	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
FM−RIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−ADM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RM−RIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−ADM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−AIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−ADM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RIM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ADM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table B.28: Multiple comparison test of evolutionary models (gap3-PAM)

period	10						100						1000					
severity	1	2	3	5	10	r	1	2	3	5	10	r	1	2	3	5	10	r
FM−RM	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
FM−RIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−ADM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RM−RIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−ADM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−AIM	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−ADM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RIM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ADM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table B.29: Multiple comparison test of hybridized models (rnd1-MSM)

period	10						100						1000					
severity	1	2	3	5	10	r	1	2	3	5	10	r	1	2	3	5	10	r
ADM−AIM	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	1
ADM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ADM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
AIM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AHDM−AHIM	1	1	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1

Table B.30: Multiple comparison test of hybridized models (rnd1-MDM)

period	10						100						1000					
severity	1	2	3	5	10	r	1	2	3	5	10	r	1	2	3	5	10	r
ADM−AIM	1	0	0	0	0	0	1	1	1	0	1	1	1	1	1	0	1	0
ADM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ADM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHDM	1	1	1	1	1	1	0	1	1	1	1	1	0	0	0	1	1	0
AIM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AHDM−AHIM	1	1	0	0	0	1	0	0	1	0	1	1	1	1	1	1	1	1

Table B.31: Multiple comparison test of hybridized models (rnd1-PAM)

period	10						100						1000					
severity	1	2	3	5	10	r	1	2	3	5	10	r	1	2	3	5	10	r
ADM−AIM	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0
ADM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
ADM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHDM	1	1	1	1	1	1	0	0	0	1	1	0	0	0	0	0	0	0
AIM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AHDM−AHIM	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0

Table B.32: Multiple comparison test of hybridized models (gap1-MSM)

period	10						100						1000					
severity	1	2	3	5	10	r	1	2	3	5	10	r	1	2	3	5	10	r
ADM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ADM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ADM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AHDM−AHIM	1	1	0	1	0	0	1	0	1	1	1	0	0	0	0	1	1	0

Table B.33: Multiple comparison test of hybridized models (gap1-MDM)

period	10						100						1000					
severity	1	2	3	5	10	r	1	2	3	5	10	r	1	2	3	5	10	r
ADM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ADM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ADM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AHDM−AHIM	1	1	1	1	0	1	0	0	0	1	1	0	1	0	0	0	0	0

Table B.34: Multiple comparison test of hybridized models (gap1-PAM)

period	10						100						1000					
severity	1	2	3	5	10	r	1	2	3	5	10	r	1	2	3	5	10	r
ADM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ADM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ADM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AHDM−AHIM	1	1	1	0	0	1	1	1	1	1	1	1	0	0	1	1	0	1

Table B.35: Multiple comparison test of hybridized models (gap2-MSM)

period	10						100						1000					
severity	1	2	3	5	10	r	1	2	3	5	10	r	1	2	3	5	10	r
ADM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ADM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	0
ADM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AHDM−AHIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table B.36: Multiple comparison test of hybridized models (gap2-MDM)

period	10						100						1000					
severity	1	2	3	5	10	r	1	2	3	5	10	r	1	2	3	5	10	r
ADM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ADM−AHDM	1	1	1	1	1	1	0	1	1	1	1	1	0	0	0	1	1	0
ADM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
AIM−AHDM	1	1	1	1	1	1	0	1	1	1	1	1	0	0	0	1	1	1
AIM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
AHDM−AHIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table B.37: Multiple comparison test of hybridized models (gap2-PAM)

period	10						100						1000					
severity	1	2	3	5	10	r	1	2	3	5	10	r	1	2	3	5	10	r
ADM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ADM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ADM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AHDM−AHIM	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0

Table B.38: Multiple comparison test of hybridized models (gap3-MSM)

period	10						100						1000					
severity	1	2	3	5	10	r	1	2	3	5	10	r	1	2	3	5	10	r
ADM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ADM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ADM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AHDM−AHIM	1	0	1	0	0	0	1	1	0	1	0	1	1	1	0	0	0	1

Table B.39: Multiple comparison test of hybridized models (gap3-MDM)

period	10						100						1000					
severity	1	2	3	5	10	r	1	2	3	5	10	r	1	2	3	5	10	r
ADM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ADM−AHDM	1	1	1	1	1	1	0	0	1	1	1	0	0	0	0	0	1	0
ADM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHDM	1	1	1	1	1	1	0	0	0	1	1	0	0	0	0	0	1	0
AIM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AHDM−AHIM	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0	1	1

Table B.40: Multiple comparison test of hybridized models (gap3-PAM)

period	10						100						1000					
severity	1	2	3	5	10	r	1	2	3	5	10	r	1	2	3	5	10	r
ADM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ADM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	0
ADM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AIM−AHDM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
AIM−AHIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AHDM−AHIM	1	1	0	0	1	1	0	1	0	0	1	1	0	1	0	1	1	0

Appendix C

A genetic algorithm for FMS

Most material in this appendix was published in Younes et al. [2002] as a part of a genetic algorithm solution to a multiple objective problem in flexible manufacturing. In this implementation a Pareto-based approach was combined with an adaptive weighted sum technique for tackling the multi-objective flexible manufacturing systems problem. Experimental results demonstrated the effectiveness of this approach for handling such complex systems.

This section outlines the genetic operators and the algorithmic approach that are also used in tackling the dynamic version of the problem. The interested reader is referred to Younes et al. [2002] for the test problems used in the static implementation, the experimental results, and more details of the employed algorithm.

C.1 Genetic operators

The adopted scheme of solution encoding enables the use of standard mutation and crossover operators without producing infeasible solutions.

In applying the mutation operator to a bit string, it sweeps down the list of bits and

replaces each, if a probability test is passed, by a randomly selected bit.

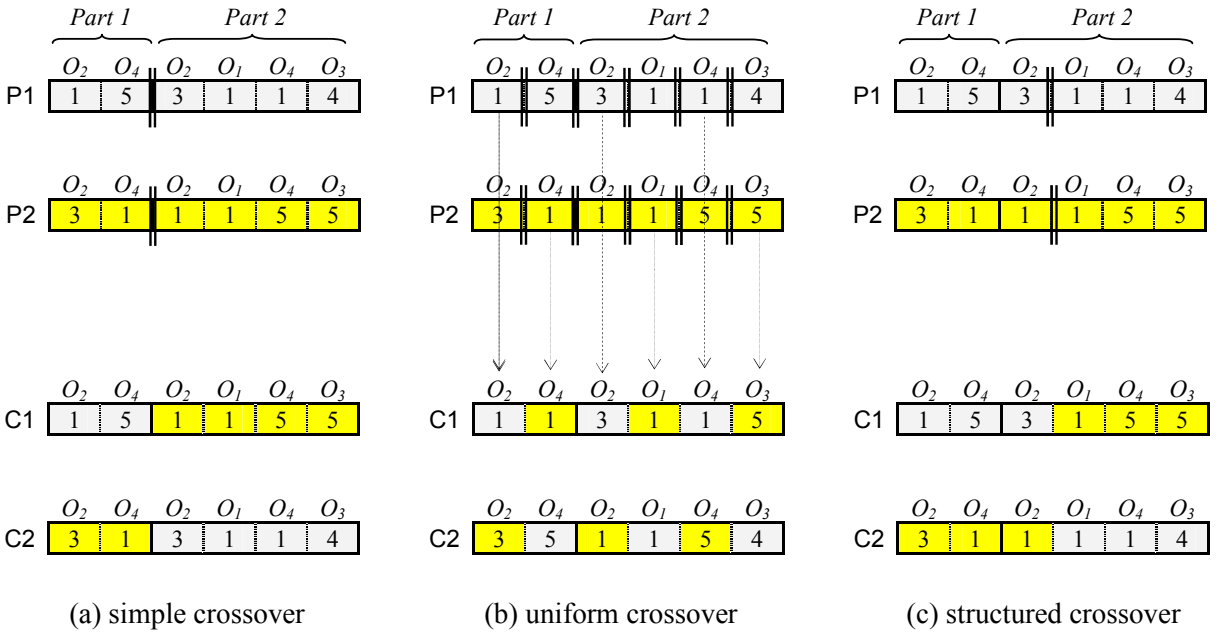


Fig. C.1: Crossover operator

The three types of crossover operators implemented in this thesis are illustrated in Figure C.1. In the simple crossover strategy the cut point is set at the part delimiter so that whole parts are transferred from a parent to a child. In uniform crossover every other gene is received from a different parent as seen in Figure C.1b. The structured crossover operator begins by randomly choosing a cut point in the chromosome as described above (see Figure C.1c).

If all initial solutions are feasible then these crossover strategies lead to complete feasible solutions. Due to the mutation operator, some chromosomes may become infeasible (when an operation is assigned to a machine that cannot handle it). In this situation a simple heuristic technique is used to repair the chromosome by assigning the correct machine to the designated operation.

C.2 Algorithm structure

A weighted-sum approach is used to assign weights to each objective function and combine the weighted objectives into a single objective function. To fully utilize the power of the GA we use several approaches: (i) fixed-weight approach, (ii) random-weight approach, and (iii) adaptive weight approach.

In the fixed-weight approach, weights are not changed during an entire evolutionary process. Weights are determined a priori to give selective pressure towards the part transfer objective function. In the random based implementation weights are randomly reset at each step in the selection procedure to give an even chance to all possible combinations. Finally, in the adaptive weight approach, weights are adjusted adaptively based on the current generation to increase search pressure toward part transfer while balancing the work-load. Figure C.2 shows a genetic algorithm implementation for flexible manufacturing systems. The algorithm begins with an encoding and initialization phase during which each string in the population is assigned a uniformly distributed random point in the solution space. In the first phase the system assigns a large weight to the first objective function (i.e w_1) and the values of the objective functions f_1 and f_2 are calculated and set to f_1^{phase1} and f_2^{phase1} respectively. In the next few phases the weights of the objective functions are adjusted adaptively such that the value of f_1 is within a $\delta\%$ tolerance of f_1^{phase1}. Each iteration of the genetic algorithm begins by evaluating the fitness of the current generation of strings. A new generation of offspring is created by applying crossover and mutation to pairs of parents which have been selected based on their fitness. The algorithm terminates after some fixed number of iterations.

```
Procedure FMS_GeneticAlgorithm
Encode Solution Space
set pop_size, max_gen, gen=0;
  set cross_rate, mutate_rate;
Initialize Population.
For phase ≤ max_phases
    Adapt Weights of Obj Functions
    While  max_gen ≥ gen
       Evaluate Fitness
       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.
       Save Best δ Solutions
       gen = gen + 1
   End While
Return best chromosomes.
```

Fig. C.2: A genetic algorithm for FMS

Appendix D

FMS Benchmarks

This appendix lists the instances used to construct dynamic FMS benchmarks. Instances gap1,gap2 and gap3 are modifications of instances in gapd file in the OR-library. Instance rnd1 is randomly generated.

D.1 gap1

Dimensions : machines = 20 parts = 40 operations = 200 part operations = 200 ;
Machines_Info
1 : 1 2 3 6 7 8 9 10 11 12 13 14 15 17 19 20 21 22 25 26 27 28 29 30 31 33 34 35 37 38 39 40 42 43 44 45 46 47 48 49 50 51 53 54 57 58 59 61 62 63 64 65 66 67 68 69 71 73 74 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 95 96 97 98 99 100 101 103 104 105 106 108 109 110 112 113 114 115 116 117 118 119 120 121 122 123 124 125 127 128 129 130 131 132 133 134 135 136 137 138 139 141 142 143 144 145 146 148 149 150 151 153 154 155 157 159 160 163 164 165 166 167 168 169 171 174 175 176 177 178 179 181 182 183 184 185 186 188 189 190 193 194 195 196 197 198 199 200
2 : 1 2 3 4 5 6 7 8 10 11 14 15 17 18 19 20 25 26 27 28 29 30 31 32 34 35 36 37 40 41 42 44 46 47 49 50 51 52 53 54 55 56 57 58 59 60 61 62 66 67 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 99 101 102 103 105 106 107 108 109 110 111 112 114 116 117 118 119 120 121 122 123 124 125 128 129 130 131 132 133 134 135 136 137 138 139 141 142 143 144 145 147 148 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 187 188 190 191 192 193 194 195 196 197 198 199 200
3 : 1 2 3 4 5 6 7 8 9 11 12 13 16 17 18 19 20 23 24 25 26 27 28 29 30 31 32 34 36 37 38 39 40 41 42 45 47

48 49 50 51 52 53 55 56 57 58 59 62 63 64 65 66 67 69 70 71 72 74 75 76 77 79 80 82 84 85 86 87 92 93 94
95 97 98 100 101 102 103 104 105 106 107 108 109 111 113 114 115 116 117 118 119 120 121 122 123 124
125 126 127 128 129 130 131 132 133 135 136 137 138 139 140 141 142 143 144 146 147 148 149 150 151
154 155 156 158 159 160 161 162 163 164 165 166 167 169 170 171 172 173 174 175 176 177 178 179 180
182 183 185 186 187 188 190 191 193 195 197 198 199 200
4 : 1 2 5 6 7 8 9 10 11 13 14 15 16 17 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41
42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 68 69 70 71 72 73 74 75 76 77
78 81 82 83 85 86 87 88 89 90 91 92 93 94 95 96 97 98 101 102 103 104 105 107 108 109 110 111 112 113
114 116 117 120 121 122 123 124 125 126 127 128 129 130 131 133 134 135 136 137 138 139 142 143 144
145 146 147 148 149 150 151 152 153 154 155 156 158 159 160 161 162 163 164 165 166 167 168 169 170
172 173 174 175 176 178 179 180 181 182 183 184 185 186 187 188 189 191 192 194 196 199 200
5 : 1 2 4 5 6 8 9 10 11 12 13 14 15 16 17 18 19 20 22 23 24 25 26 27 28 29 31 32 33 34 35 37 38 39 40 41
42 43 44 47 48 49 51 52 53 54 55 56 57 59 60 61 62 63 64 65 66 67 68 69 71 72 73 74 76 77 79 80 82 83 84
85 86 87 88 89 90 91 92 93 95 96 97 98 99 101 102 103 104 105 107 108 109 110 111 113 114 116 117 118
119 120 121 122 123 124 125 126 127 128 129 130 131 133 134 135 136 138 139 140 141 143 144 145 146
147 148 150 151 153 154 155 156 157 158 159 160 161 163 165 166 167 168 169 170 172 173 174 175 176
177 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 199 200
6 : 2 3 4 5 6 7 8 9 10 12 14 15 16 18 19 20 21 22 24 25 26 27 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43
44 45 46 48 49 50 51 52 53 54 55 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 74 75 76 78 79 80 81 82
83 84 85 87 88 89 90 91 92 94 95 96 97 98 99 100 101 102 104 105 106 107 108 109 110 111 112 113 114
115 118 119 120 121 122 123 124 126 127 128 130 131 132 133 134 135 136 137 138 139 140 141 142 143
144 145 146 148 149 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 169 170 172 173 174
176 178 180 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200
7 : 1 2 3 4 5 6 7 8 9 10 11 12 13 15 16 17 18 19 20 21 23 24 25 26 28 29 30 31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 48 49 51 52 53 56 57 58 59 60 61 62 63 64 65 66 67 70 71 72 74 75 77 78 79 84 85 87 89
90 92 93 94 95 96 97 98 99 100 101 102 104 105 107 108 109 110 113 114 115 116 117 118 119 120 121 122
123 124 125 126 127 128 129 132 133 134 135 136 137 138 139 140 141 142 143 144 146 147 148 149 150
151 152 153 155 156 157 158 159 161 162 163 164 165 166 167 168 169 170 171 173 174 176 177 179 180
181 182 183 184 185 190 191 193 194 195 196 198 199 200
8 : 1 3 4 5 7 8 9 10 11 12 13 14 15 16 17 18 19 20 24 25 26 27 28 29 32 33 34 35 36 37 38 39 40 41 42 44 45
47 48 50 51 52 54 56 60 61 62 63 65 66 67 68 69 71 72 74 76 77 78 79 80 82 83 84 85 86 87 90 91 92 94 95
96 97 99 100 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 122 123 124
126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 144 145 146 147 148 151 152 153 154
155 156 157 158 159 160 161 162 164 165 166 169 170 171 172 173 174 175 176 177 178 179 180 183 184
185 187 188 190 191 192 193 194 196 197 198 199 200
9 : 1 2 3 4 5 6 8 9 10 11 13 14 15 16 18 19 20 21 23 25 27 28 29 30 31 33 36 37 38 39 41 42 43 44 45 46 48
49 50 51 52 54 55 56 57 59 60 62 63 64 65 66 67 69 70 71 72 73 74 75 76 78 79 81 82 83 84 85 88 89 90 91
92 93 94 95 96 97 99 100 101 102 103 105 106 107 109 110 111 112 114 115 116 118 120 121 122 123 125

126 127 130 131 132 133 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153
154 155 156 159 160 161 162 163 164 165 166 167 168 171 172 173 174 175 176 177 178 179 180 182 183
186 187 188 189 190 191 192 193 194 196 197 198 199 200
10 : 1 2 3 4 6 8 9 10 11 12 13 14 15 17 18 19 20 21 22 24 25 26 27 28 29 31 32 33 34 36 37 38 39 41 42 43
45 46 47 51 52 55 56 57 58 59 60 61 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 86
87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 105 106 108 111 112 113 114 116 117 118 120 122
123 124 126 128 129 130 131 132 133 134 135 137 138 139 140 141 142 143 144 145 146 149 150 151 152
153 154 155 156 157 158 159 160 161 162 163 164 166 167 168 169 172 173 174 175 176 178 179 180 181
183 184 185 186 188 189 190 191 192 193 194 195 196 197 198 199 200
11 : 1 2 3 4 5 6 7 9 10 11 12 13 14 15 16 17 18 19 20 22 23 24 25 26 28 29 30 31 33 34 35 36 37 38 39 40
41 42 43 44 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 64 66 67 68 69 70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 88 89 90 91 92 93 94 96 98 99 100 101 102 103 104 105 106 107 108 109 110 111 113
114 117 118 119 120 122 123 125 126 127 128 129 130 131 132 133 134 137 138 139 140 141 142 143 145
146 147 148 149 150 151 152 153 154 156 157 158 159 160 161 162 163 165 167 168 169 170 171 172 173
174 175 177 178 179 180 182 183 185 186 188 189 190 191 193 194 195 196 197 199 200
12 : 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 19 20 21 22 23 24 25 26 27 28 30 31 32 33 34 35 36 37 38
39 40 42 43 44 45 46 47 48 49 50 51 52 53 54 56 57 58 59 60 61 63 64 65 66 68 70 71 72 73 74 77 78 79 80
81 82 84 86 89 90 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 109 110 111 112 113 114 115 116
117 118 119 120 121 122 123 126 127 128 130 131 134 135 136 137 138 139 140 141 142 143 144 145 147
148 149 150 154 157 158 159 160 161 162 163 164 166 167 168 169 170 171 172 173 174 175 176 178 179
180 182 183 184 186 187 188 189 190 192 193 194 195 196 197 198 199 200
13 : 1 2 3 4 5 6 7 9 10 11 12 13 14 15 16 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 36 37 38 40
41 44 45 46 47 48 49 50 51 52 53 54 55 56 58 59 60 61 62 64 65 66 67 68 69 70 71 72 74 75 76 77 78 79 80
81 82 84 85 86 87 88 89 91 92 94 96 97 98 99 100 101 102 104 105 106 107 108 109 110 111 113 115 116
117 118 119 120 121 122 123 124 126 127 128 129 132 133 134 135 137 138 139 140 141 143 145 146 147
148 149 150 151 152 153 155 156 157 159 160 162 164 165 166 167 168 169 170 171 172 173 174 175 176
177 178 179 182 183 184 185 186 187 189 190 191 192 193 194 195 196 197 198 200
14 : 1 3 4 5 6 8 9 12 13 14 16 17 18 19 21 22 23 24 26 27 28 29 31 32 33 34 35 36 37 38 39 40 41 42 43 44
46 47 48 49 50 51 52 53 55 56 57 58 59 60 61 62 64 65 66 67 68 69 70 71 73 74 75 76 77 79 80 81 82 83 84
85 86 87 88 91 92 93 94 95 96 97 98 99 100 101 102 103 104 106 107 108 109 110 111 112 113 114 115 117
118 119 120 121 122 123 124 125 126 127 128 129 130 132 133 134 136 137 138 140 141 142 143 144 145
146 147 148 150 151 152 153 155 156 158 159 161 164 165 166 167 168 169 170 171 174 175 176 177 178
179 180 182 183 184 185 186 187 188 190 191 192 194 195 196 197 198 199 200
15 : 1 2 3 4 5 6 7 8 9 10 12 13 14 15 17 19 20 21 24 25 26 27 28 29 30 31 32 34 35 37 38 39 41 44 45 47 49
51 54 55 56 57 58 59 60 61 62 63 64 65 67 68 69 70 71 72 73 74 75 76 77 78 80 81 82 83 84 85 86 87 91 92
93 97 98 99 100 101 102 103 104 105 106 107 110 111 112 113 114 115 116 117 118 119 120 121 122 123
124 125 127 128 129 131 132 133 134 135 137 138 140 141 142 143 144 145 148 149 150 151 152 153 154
155 156 157 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 181

182 183 186 187 188 189 191 192 193 195 198 199 200
16 : 1 2 3 4 5 7 8 10 12 13 14 15 16 18 19 20 21 22 23 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41
42 43 44 45 46 51 52 53 54 55 56 57 58 61 62 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83
84 85 86 87 88 89 90 91 93 94 95 98 99 100 101 102 103 104 105 107 108 109 110 111 112 113 114 115 116
117 119 121 123 125 128 131 132 133 134 135 137 138 139 140 141 142 144 145 146 147 148 149 150 151
152 153 154 155 156 158 159 160 161 162 163 164 165 166 167 168 169 173 174 175 176 178 179 180 181
182 183 185 186 187 188 189 190 191 192 193 196 198 199 200
17 : 1 3 4 5 6 7 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 27 28 29 30 31 33 34 35 36 37 38 39 40
41 42 44 45 46 47 49 50 52 56 57 58 59 60 61 62 63 64 65 66 67 70 71 72 73 74 75 77 78 79 81 82 83 85 86
87 88 89 90 91 92 93 95 96 97 98 99 100 101 102 103 104 105 106 107 108 110 111 112 113 115 116 117 118
119 121 122 123 124 125 126 127 128 129 130 131 132 133 134 136 137 138 139 140 141 143 145 146 147
148 150 151 152 153 154 155 156 157 158 159 161 162 163 164 166 167 168 169 170 171 173 175 176 177
178 179 180 181 182 183 184 185 186 187 189 190 191 192 193 194 195 196 197 198 199 200
18 : 1 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 19 21 23 24 25 26 27 28 30 31 32 33 35 36 37 38 40 41 42 43 44
45 46 47 48 49 50 51 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 69 70 71 72 73 75 76 77 78 79 80 81 82
83 84 85 86 87 89 90 91 92 93 94 96 97 98 99 100 101 102 103 104 105 106 107 108 109 112 113 114 115
116 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 142 143
144 145 147 148 149 150 151 154 155 156 157 158 160 161 162 163 164 165 167 168 170 171 172 173 174
175 176 178 179 180 181 183 184 185 186 187 188 189 190 192 193 194 195 196 197 198 199 200
19 : 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 21 23 24 27 28 30 31 32 33 34 37 38 39 40 41 42 43 44
45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
82 83 84 85 86 87 88 89 90 93 94 95 96 97 99 100 101 102 103 104 105 106 108 109 111 112 113 114 115
117 118 119 120 121 122 123 124 127 128 130 131 132 134 135 136 139 140 141 144 145 146 147 148 150
151 152 153 154 155 157 158 159 160 161 162 163 164 165 166 167 169 170 171 172 173 174 175 176 177
179 180 181 182 183 184 185 186 188 189 190 191 192 193 194 195 196 197 198 199 200
20 : 1 3 4 5 6 7 10 11 12 13 15 16 17 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 37 38 39 40 41 42
43 44 45 46 47 48 49 50 51 52 53 54 55 57 59 60 61 62 63 64 65 66 67 68 70 71 72 73 75 76 77 78 79 80 82
83 84 85 86 87 89 90 91 93 95 96 97 98 99 100 101 102 103 104 105 107 108 110 111 112 113 114 115 116
117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141
142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 165 166 167
168 169 172 173 175 176 177 178 179 181 182 183 184 185 186 189 190 191 192 193 194 195 196 197 198
199 200
;
Parts_Info
1 : 141 160 19 83 25
2 : 65 93 9 125 96
3 : 146 4 24 115 105
4 : 98 59 26 10 60

5 : 37 77 54 68 92
6 : 40 89 134 112 15
7 : 31 181 99 97 81
8 : 192 12 21 88 116
9 : 161 128 108 133 142
10 : 57 113 78 144 126
11 : 132 62 163 71 148
12 : 94 14 170 178 75
13 : 136 91 152 180 69
14 : 49 117 129 28 43
15 : 11 39 190 6 8
16 : 110 198 51 199 200
17 : 44 154 103 119 104
18 : 164 38 79 158 174
19 : 176 147 20 102 187
20 : 67 80 171 64 33
21 : 30 120 46 76 58
22 : 182 121 177 73 173
23 : 5 100 7 186 16
24 : 139 124 52 41 169
25 : 127 188 172 3 191
26 : 95 55 162 138 74
27 : 153 86 123 195 168
28 : 183 156 157 87 135
29 : 111 42 114 106 53
30 : 23 179 131 122 56
31 : 118 196 167 34 90
32 : 130 22 151 165 184
33 : 149 70 197 140 175
34 : 18 82 13 32 155
35 : 194 137 29 159 150
36 : 27 72 66 101 185
37 : 35 84 145 107 48
38 : 47 17 166 50 109
39 : 63 193 1 143 189
40 : 85 2 45 36 61
; END

D.2 gap2

Dimensions : machines = 20 parts = 20 operations = 100 part operations = 100 ;
Machines_Info
1 : 1 4 5 6 7 8 9 10 11 12 13 15 19 21 22 26 27 28 29 30 31 33 34 35 36 37 38 39 40 42 43 44 45 46 47 48 49 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 74 75 77 78 79 80 81 82 84 85 86 89 90 91 92 94 95 96 97 99 100
2 : 1 2 3 4 5 7 9 10 13 14 15 16 17 20 21 22 23 24 25 26 27 28 30 31 32 33 34 35 36 37 38 40 41 42 43 45 46 47 48 49 50 51 52 53 54 56 57 58 59 61 62 63 64 65 66 67 68 69 70 72 73 74 75 76 78 79 80 81 82 83 84 85 86 88 90 91 92 93 94 95 96 98 99 100
3 : 1 2 3 5 7 9 10 11 12 13 15 17 18 19 20 21 22 23 24 25 26 27 29 30 31 32 33 34 36 37 38 39 40 42 44 45 46 48 49 50 51 52 53 54 55 57 60 61 62 63 64 66 67 68 69 70 71 73 75 76 77 78 79 80 81 82 83 84 86 87 89 90 91 92 93 94 95 96 97 98 99 100
4 : 1 2 3 4 5 6 7 10 12 13 14 16 17 18 19 20 21 23 25 26 27 28 31 32 33 34 35 36 37 38 40 41 42 43 44 45 46 48 49 51 52 53 54 55 56 57 58 60 61 62 63 64 65 66 68 70 71 72 76 77 79 80 81 82 83 84 85 86 87 88 89 91 92 93 94 95 96 97
5 : 1 2 3 4 5 6 7 9 10 11 12 13 14 15 16 18 19 20 21 23 24 25 26 27 28 30 32 34 37 38 39 42 43 47 48 49 50 52 53 54 55 56 57 58 59 60 62 63 64 65 66 67 68 69 71 72 73 74 76 77 79 80 81 83 84 85 88 89 90 91 94 96 97 98 99
6 : 1 2 3 4 5 6 7 8 10 11 13 14 15 16 17 18 20 21 22 23 24 25 26 29 30 31 32 33 34 36 38 40 41 42 43 44 45 47 49 51 52 53 55 57 58 59 60 61 62 63 64 65 67 69 70 71 72 73 74 75 76 77 78 80 81 82 83 84 86 87 88 90 93 95 96 97 98 99 100
7 : 1 2 3 4 5 7 9 10 11 12 13 14 15 16 17 18 19 20 22 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 41 42 43 44 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 65 66 67 68 69 70 71 73 74 75 76 77 79 80 81 82 83 84 85 86 89 90 91 92 93 94 95 96 97 98 99 100
8 : 1 2 3 4 5 6 9 10 11 12 13 14 15 16 17 18 19 20 21 23 24 25 26 27 29 31 32 33 34 35 36 37 38 39 40 41 42 43 45 46 47 49 50 51 52 53 54 55 56 57 58 59 60 65 68 69 70 72 73 74 77 79 81 82 83 84 85 86 87 88 90 91 92 93 94 95 97 98 99 100
9 : 1 2 4 6 7 8 9 10 11 12 13 14 15 17 18 19 20 21 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 42 43 46 47 49 50 51 53 54 55 56 57 59 60 61 62 63 65 66 68 72 73 74 75 76 77 78 79 80 81 82 84 85 86 87 88 89 90 91 92 93 94 95 97 99 100
10 : 1 2 4 5 7 8 9 10 11 12 13 15 16 17 18 19 20 21 22 23 25 26 27 28 29 30 31 32 33 34 35 36 37 38 40 41 42 44 46 47 48 49 50 51 53 54 55 57 58 62 63 64 65 66 68 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 96 97 98 99 100
11 : 1 2 4 6 7 8 10 11 13 14 15 17 18 19 21 22 23 24 25 26 27 28 29 30 31 34 35 36 37 38 39 40 41 42 43 44 45 46 48 49 50 51 52 53 54 55 56 57 58 59 61 62 64 65 67 68 69 70 72 76 77 78 79 80 81 82 83 84 85 86 87 89 90 91 93 94 95 96 97
12 : 1 3 4 5 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 27 29 32 34 35 36 37 38 39 40 42 43 44

45 46 47 48 50 52 53 54 55 57 59 60 61 62 63 64 65 66 68 70 71 73 74 76 77 78 79 80 81 82 83 84 87 88 90 91 92 93 94 95 96 97 98 99
13 : 1 2 3 4 5 6 7 8 9 10 11 12 13 15 17 18 19 20 23 24 25 26 27 29 30 31 33 34 36 37 38 39 40 41 42 43 44 45 46 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 77 78 79 81 82 83 84 86 87 88 89 90 91 93 94 95 96 97 99
14 : 1 2 3 5 6 7 8 9 11 12 14 15 16 17 18 19 21 23 25 26 27 28 29 30 31 32 34 35 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 53 54 56 57 58 59 60 61 62 63 64 65 66 68 70 71 77 78 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
15 : 1 2 3 6 7 8 9 10 11 13 14 15 16 18 19 20 21 22 23 24 25 26 27 28 29 31 32 34 35 36 37 39 40 41 42 44 45 47 48 49 50 51 52 53 55 56 59 60 62 63 64 65 66 67 69 70 71 72 74 75 76 77 78 79 80 81 82 83 85 86 87 88 89 90 91 92 93 94 95 97 99 100
16 : 1 2 4 6 7 9 10 11 12 13 14 15 16 18 20 21 22 23 24 25 26 27 28 29 31 32 33 34 35 36 37 38 39 40 41 43 44 45 46 47 48 49 50 51 53 54 55 56 57 58 60 61 62 63 64 66 67 68 69 70 71 72 74 76 77 78 80 81 82 83 84 86 87 89 90 91 92 94 95 96 97 99 100
17 : 1 3 5 6 7 8 9 11 12 14 15 16 17 19 20 21 22 24 26 27 28 29 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 48 49 51 53 54 56 57 58 59 60 61 62 63 64 66 67 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 94 95 96 98 99 100
18 : 1 2 3 4 5 6 7 9 10 11 12 13 15 16 18 20 21 22 23 24 25 27 28 29 31 35 36 37 39 41 42 43 44 45 46 47 48 49 50 52 53 54 56 57 58 59 60 61 63 64 65 66 67 68 69 72 73 75 77 79 83 84 87 88 89 90 91 92 93 94 95 96 97 98
19 : 1 2 3 4 5 6 7 8 10 11 12 14 16 17 18 19 20 21 22 23 24 25 30 31 34 35 36 37 38 41 42 43 44 45 46 48 49 50 52 53 55 56 57 58 60 61 62 63 64 65 66 67 69 70 71 72 73 74 75 76 77 78 79 81 83 84 85 86 88 89 90 91 92 93 95 96 97 99 100
20 : 1 3 4 5 6 8 9 10 11 12 13 14 15 16 17 18 19 22 23 24 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 56 57 58 59 60 61 62 63 64 65 66 67 68 71 72 73 74 75 76 77 79 80 81 82 84 85 86 87 88 89 90 92 93 94 95 96 97 98 99
;
Parts_Info
1 : 4 50 98 69 38
2 : 99 22 53 26 77
3 : 96 10 8 44 49
4 : 25 19 63 36 17
5 : 3 65 12 41 83
6 : 92 45 64 85 27
7 : 78 56 7 72 11
8 : 91 81 80 79 62
9 : 39 84 54 28 60
10 : 51 48 67 75 90

11 : 95 97 74 14 94
12 : 5 55 57 18 100
13 : 61 73 46 16 71
14 : 31 24 89 33 40
15 : 15 37 21 42 87
16 : 30 32 35 59 47
17 : 86 82 6 58 34
18 : 88 76 29 1 70
19 : 93 2 52 43 68
20 : 13 23 66 9 20
; END

D.3 gap3

Dimensions : machines = 10 parts = 20 operations = 100 part operations = 100 ;
Machines_Info
1 : 1 2 3 4 5 9 10 12 13 14 15 16 17 18 19 21 22 24 25 26 27 28 29 31 33 35 36 37 40 41 43 45 47 48 49 50 51 52 53 57 58 59 60 62 64 65 66 67 68 69 71 72 74 76 77 78 79 80 82 83 85 86 87 88 89 91 94 96 97 98 100
2 : 1 2 3 4 5 7 8 11 12 19 21 22 23 24 27 29 31 32 33 35 36 37 39 43 44 45 46 48 51 54 55 60 65 67 68 71 72 73 74 75 77 78 80 81 82 83 84 85 86 87 88 89 91 93 94 95 96 97
3 : 1 2 4 5 7 9 11 12 13 15 16 17 18 20 21 24 25 27 30 32 33 34 35 37 38 39 43 44 45 47 48 49 50 51 53 54 56 59 60 61 67 68 70 71 72 73 74 75 77 79 80 82 83 84 86 87 89 90 91 93 95 96 97 98 100
4 : 1 4 5 6 9 10 11 13 14 15 16 18 19 20 21 23 25 26 27 29 30 31 32 33 34 35 38 40 41 45 46 48 49 52 53 57 58 59 61 64 67 70 71 73 74 77 78 79 80 81 82 83 84 86 90 91 92 93 94 96 98 99 100
5 : 1 2 4 7 8 10 11 12 13 16 17 18 20 22 23 24 25 28 29 30 31 33 35 36 37 40 41 42 43 44 45 46 48 49 50 52 54 55 59 60 61 62 67 69 71 75 79 80 81 82 84 85 86 87 88 89 91 92 93 94 95 98 100
6 : 1 2 3 4 5 6 7 8 13 14 15 17 18 19 21 25 26 28 29 34 37 39 40 41 42 43 45 47 51 53 54 55 56 57 58 59 62 63 64 65 67 68 70 71 77 78 79 81 82 83 85 86 87 88 89 90 91 92 93 94 95 96 97 98 100
7 : 2 3 4 6 7 8 9 11 12 13 14 15 16 17 19 22 23 24 25 26 29 30 33 34 35 39 40 41 42 43 44 45 50 51 52 54 55 56 57 58 61 62 65 66 70 72 73 74 79 83 84 85 86 89 90 92 94 95 96 97 100
8 : 1 3 5 6 7 8 9 12 13 14 15 16 17 19 20 21 22 23 24 25 26 27 28 29 30 32 33 34 35 37 39 41 44 45 46 47 48 50 51 52 54 55 57 59 60 61 62 63 64 65 68 69 70 71 72 73 74 77 78 79 80 82 84 85 86 89 90 91 94 95 96 97 98 99 100
9 : 3 4 5 8 9 10 11 13 16 17 19 20 24 25 26 27 28 30 32 33 38 39 41 42 45 46 47 49 50 51 52 54 56 57 58 59 60 62 63 64 65 66 67 68 70 73 74 77 78 82 84 86 88 92 93 95 97 98 99 100
10 : 2 3 4 5 6 8 10 11 12 13 15 16 18 20 22 23 24 26 30 33 34 35 36 37 38 41 42 44 46 49 50 53 55 56 57 58 59 60 61 62 63 64 65 67 69 75 76 77 79 81 82 83 84 86 87 89 90 91 92 95 96 97 99
;
Parts_Info
1 : 80 14 52 83 78
2 : 82 16 79 96 30
3 : 44 41 24 97 55
4 : 70 86 74 47 11
5 : 36 94 65 25 57
6 : 35 54 9 90 23
7 : 48 49 98 64 85
8 : 8 7 27 92 10
9 : 37 71 46 17 62
10 : 2 67 5 58 39
11 : 69 63 32 91 59

12 : 88 33 20 87 28
13 : 4 77 43 21 61
14 : 84 29 38 68 53
15 : 34 72 89 100 76
16 : 6 19 95 22 40
17 : 42 99 45 15 18
18 : 93 3 75 31 12
19 : 66 1 50 73 81
20 : 56 26 60 51 13
; END

D.4 rnd1

Dimensions : machines = 11 parts = 20 operations = 9 part operations = 62 ;
Machines_Info
1 : 1 2
2 : 3 4
3 : 5 6
4 : 7 8
5 : 1 2 3
6 : 3 4 9
7 : 5 6 7
8 : 7 8 9
9 : 1 2 3 4
10 : 5 6 7 8
11 : 1 2 3 4 9
;
Parts_Info
1 : 1 2 3
2 : 1 2 4
3 : 1 2 5
4 : 3 6 7
5 : 4 7 8
6 : 5 8 9
7 : 6 7 9
8 : 4 5 9
9 : 6 8 9
10 : 5 7 9
11 : 7 8 9
12 : 3 4 5
13 : 4 5 6
14 : 6 7 8
15 : 2 4 6
16 : 1 3 5
17 : 1 3 7
18 : 1 3 9
19 : 2 5 7 9
20 : 1 3 5 7
; END

Appendix E

Environmental Changes in COPs

Table E.1: Environmental changes in COPs

Problem	Affected Element	Elementary Step	Type of Change
TSP	Inter-city distance	Change on leg	Non-dimensionality
	Number of cities	Add/delete one city	Dimensionality
	Cities	Swap one pair	Mapping
KP	Object value	Change one value	Non-dimensionality
	Object weight	Change one weight	Non-dimensionality
	Knapsack capacity	Change capacity	Non-dimensionality
	Number of objects	Add/delete one object	Dimensionality
	Objects	Swap one pair	Mapping
FMS	Number of machines	Add/del one machine	Non-dimensionality
	Number of parts	Add/del one part	Dimensionality
	Machines	Swap one pair	Mapping

Bibliography

Alba, E. and Saucedo, J. F. 2005. Panmictic versus decentralized genetic algorithms for non-stationary problems. (MIC2005) The 6th Metaheuristics International Conference, Vienna, Austria.

Algorithms and Theory of Computation Handbook, C. P. L. 1999. Levenshtein distance. from Dictionary of Algorithms and Data Structures, Paul E. Black, ed., NIST.

Angeline, P. J. 1997. Tracking extrema in dynamic environments. In *Sixth International Conference on Evolutionary Programming*, P. J. Angeline, R. G. Reynolds, J. R. McDonnell, and R. Eberhart, Eds. LNCS, vol. 1213. Springer, 335–345.

Aporntewan, C. and Chongstitvatana, P. 2001. A hardware implementation of the compact genetic algorithm. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*. IEEE Press, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, 624–629.

Bäck, T. 1992. The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm. In *Parallel Problem Solving from Nature, PPSN*. 87–96.

Bäck, T. 1997. Self-adaptation. In *Handbook of evolutionary computation*, D. B. F. . Z. M. T. Back, Ed. Oxford University Press and Institute of Physics.

Bäck, T. and Schwefel, H.-P. 1993. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation 1,* 1, 1–23.

Barker, A. L. and Martin, W. N. 2000. Dynamics of a distance-based population diversity measure. In *Proc. of the 2000 Congress on Evolutionary Computation*. IEEE Service Center, Piscataway, NJ, 1002–1009.

Barr, R., Golden, B., Kelly, J., Resende, M., and Stewart, W. 1995. Designing and reporting on computational experiments with heuristic methods. *J. of Heuristics 1,* 1, 9–32.

Beasley, D., Bull, D. R., and Martin, R. R. 1993a. An overview of genetic algorithms: Part 1, fundamentals. *University Computing 15,* 2, 58–69.

Beasley, D., Bull, D. R., and Martin, R. R. 1993b. An overview of genetic algorithms: Part 2, research topics. *University Computing 15,* 4, 170–181.

Beasley, J. E. 1990. Or-library: distributing test problems by electronic mail. *Journal of the Operational Research Society 41(11)*, 1069–1072.

Bianchi, L. 1990. Notes on dynamic vehicle routing - the state of the art. Tech. Rep. idsia 05-01, Italy.

Bierwirth, C. and Kopfer, H. 1994a. Dynamic task scheduling with genetic algorithms in manufacturing systems. Tech. rep., Department of Economics, University of Bremen, Germany.

Bierwirth, C. and Kopfer, H. 1994b. Dynamic task scheduling with genetic algorithms in manufacturing systems. Tech. rep., Department of Economics, University of Bremen, Germany.

Bierwirth, C., Kopfer, H., Mattfeld, D. C., and Rixen, I. 1995a. Genetic algorithm based scheduling in a dynamic manufacturing environment. In *Proc. of 1995 IEEE Conf. on Evolutionary Computation*. IEEE Press, Piscataway, NJ.

Bierwirth, C., Kopfer, H., Mattfeld, D. C., and Rixen, I. 1995b. Genetic algorithm based scheduling in a dynamic manufacturing environment. In *Proc. of IEEE Conference on Evolutionary Computation*. IEEE Press.

Bierwirth, C. and Mattfeld, D. C. 1999. Production scheduling and rescheduling with genetic algorithms. *Evolutionary Computation 7,* 1, 1–18.

Blum, C. and Roli, A. 2003. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv. 35,* 3, 268–308.

Blythe, P. 1998. Evolving robust strategies for autonomous flight: A challenge to optimal control theory. In *Adaptive Computation in Design and Manufacture*. Springer-Verlag.

Branke, J. 1999a. Evolutionary approaches to dynamic optimization problems: A survey. pp. 134–137, GECCO Workshops, A. Wu (ed.).

Branke, J. 1999b. Memory enhanced evolutionary algorithms for changing optimization problems. In *1999 Congress on Evolutionary Computation*. IEEE Service Center, Piscataway, NJ, 1875–1882.

Branke, J. 2001. *Evolutionary Optimization in Dynamic Environments*. Environments. Kluwer Academic Publishers.

Branke, J., Kaussler, T., Schmidt, C., and Schmeck, H. 2000. A multi-population approach to dynamic optimization problaqw2 nh ems. In *Adaptive Computing in Design and Manufacturing 2000*. Springer.

Burke, E. K., Gustafson, S., and Kendall, G. 2004. Diversity in genetic programming: An analysis of measures and correlation with fitness. *IEEE Transactions on Evolutionary Computation 8,* 1, 47–62.

Chen, J.-H. and Ho, S.-Y. 2002. Multi-objective evolutionary optimization of flexible manufacturing systems. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO2002)*. Morgan Koffman, New York, New York, 1260–1267.

Chu, P. C. and Beasley, J. E. 1997. A genetic algorithm for the generalised assignment problem. *Computers and Operations Research 24*, 17–23.

Cobb, H. G. 1990. An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments. Tech. Rep. 6760 (NLR Memorandum), Navy Center for Applied Research in Artificial Intelligence, Washington, D.C.

Cobb, H. G. and Grefenstette, J. J. 1993. Genetic algorithms for tracking changing environments. In *Proc. of the Fifth Int. Conf. on Genetic Algorithms*, S. Forrest, Ed. Morgan Kaufmann, San Mateo, CA, 523–530.

Cook, S. A. 1971. The complexity of theorem-proving procedures. In *STOC*. ACM, 151–158.

Dasgupta, D. and McGregor, D. R. 1992. Nonstationary function optimization using the structured genetic algorithm. In *Parallel Problem Solving from Nature 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels 1992)*, R. Männer and B. Manderick, Eds. Elsevier, Amsterdam, 145–154.

Davis, L. 1985. Applying adaptive algorithms to epistatic domains. In *Proc. of the 9th Int. Joint Conf. on Artificial Intelligence*, A. Joshi, Ed. Morgan Kaufmann, Los Angeles, CA, 162–164.

Davis, L. 1989. Adapting operator probabilities in genetic algorithms. In *ICGA*. 61–69.

De Jong, K. A. 1975. An analysis of the behavior of a class of genetic adaptive systems. Ph.D. thesis, University of Michigan.

Dimopoulos, C. and Zalzala, A. 2000. Recent developments in evolutionary computation for manufacturing optimization: Problems, solutions, and comparisons. *IEEE Transactions on Evolutionary Computation 4,* 2, 93–113.

Dorigo, M. and Di Caro, G. 1999. The ant colony optimization meta-heuristic. In *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, Eds. McGraw-Hill, London, 11–32.

Dorigo, M., Maniezzo, V., and Colorni, A. 1996. Ant system: Optimization by a colony of cooperating agents. *IEEE Trans. on Systems, Man, and Cybernetics–Part B 26,* 1, 29–41.

Eiben, A. E., Hinterding, R., and Michalewicz, Z. 1999. Parameter control in evolutionary algorithms. *IEEE Trans. on Evolutionary Computation 3,* 2, 124–141.

Eliot, T. 1963. *Collected Poems 1909–1962*. Faber and Faber, London.

Eyckelhof, C. J. and Snoek, M. 2002. Ant systems for a dynamic TSP. In *Ant Algorithms*. 88–99.

Feo, T. and Resende, M. 1995. Greedy randomized adaptive search procedures. *J. of Global Optimization 6*, 109–133.

Fogel, L. J., Owens, A. J., and Walsh, M. J. 1966. *Artificial Intelligence Through simulated Evolution*. John Wiley and Sons, New York.

Gelfand, S. B. and Mitter, S. K. 1985. Analysis of simulated annealing for optimization. Laboratory for Information and Decision Systems, MIT LIDS-P ; 1494 p. 32.

Gen, M. and Cheng, R. 1999. *Genetic Algorithms*. John Wiley & Sons, Inc., New York, NY, USA.

Glover, F. and Kochenberger, G. 2003. New gains for applying metaheuristics in mixed integer, combinatorial and nonlinear optimization.

Glover, F. and Laguna, F. 1997. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA.

Goetschalckx, M., Sharp, G., TjenderaSantoso, and Bhargavi, L. 1999. Vehicle routing. This is an electronic document. Date of publication: May 9, 1999. Dateretrieved: April 26, 2002. Date last modified: May 9, 1999.

Goldberg, D. E. 1989. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, Reading, MA.

Goldberg, D. E. 2002. *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA.

Goldberg, D. E. and Lingle, Jr., R. 1985. Alleles, loci, and the traveling salesman problem. In *Proc. of an Int. Conf. on Genetic Algorithms and Their Applications*, J. J. Grefenstette, Ed. Lawrence Erlbaum, Hillsdale, NJ, 154–159.

Goldberg, D. E. and Smith, R. E. 1987. Nonstationary function optimization using genetic algorithms with dominance and diploidy. In *Second International Conference on Genetic Algorithms*, J. J. Grefenstette, Ed. Lawrence Erlbaum Associates, 59–68.

Gomory, R. 1958. Outline of an algorithm for integer solution to linear programs. *Bulletin American Mathematical Society 64*, 275–278.

Graham, P. and Nelson, B. 1996. Genetic algorithms in software and in hardware - A performance analysis of workstations and custom computing machine implementations. In *IEEE Symposium on FPGAs for Custom Computing Machines*, K. L. Pocek and J. Arnold, Eds. IEEE Computer Society Press, Los Alamitos, CA, 216–225.

Grefenstette, J. 1986. Optimization of control parameters for genetic algorithms. *IEEE Trans. on Systems, Man, and Cybernetics SMC-16,* 1, 122–128.

Grefenstette, J. J. 1992. Genetic algorithms for changing environments. In *Parallel Problem Solving from Nature 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels 1992)*, R. Männer and B. Manderick, Eds. Elsevier, Amsterdam, 137–144.

Grefenstette, J. J. 1999. Evolvability in dynamic fitness landscapes: a genetic algorithm approach. In *1999 Congress on Evolutionary Computation*. IEEE Service Center, Piscataway, NJ, 2031–2038.

Grötschel, M. and Holland, O. 1991. Solution of large-scale symmetric travelling salesman problems. *Math. Program. 51*, 141–202.

Guntsch, M., Middendorf, M., and Schmeck, H. 2001. An ant colony optimization approach to dynamic tsp. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, L. Spector, E. D. Goodman, A. Wu, W. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, Eds. Morgan Kaufmann, San Francisco, California, USA, 860–867.

Hadad, B. S. and Eick, C. F. 1997. Supporting polyploidy in genetic algorithms using dominance vectors. In *Evolutionary Programming VI*, P. J. Angeline, R. G. Reynolds, J. R. McDonnell, and R. Eberhart, Eds. Springer, Berlin, 223–234. Lecture Notes in Computer Science 1213.

Hajek, B. 1988. Cooling schedules for optimal annealing. *Mathematics of Operations Research 13,* 2, 311–329.

Hansen, P. and Mladenović, N. 1999. An introduction to variable neighborhood search. In *Meta-heuristics, Advances and trends in local search paradigms for optimization*, S. Voss, S. Martello, I. H. Osman, and C. Roucairol, Eds. Kluwer Academic Publishers, 433–458.

Hansen, P. and Mladenovic, N. 2001. Variable neighborhood search: Principles and applications. *European Journal of Operations Research 130*, 449–467. Invited Review.

Hart, E. and Ross, P. 1999a. The evolution and analysis of a potential antibody library for use in job-shop scheduling. In *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, Eds. McGraw-Hill, London, 185–202.

Hart, E. and Ross, P. 1999b. An immune system approach to scheduling in changing environments. In *Proc. of the Genetic and Evolutionary Computation Conf. GECCO-99*, W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, Eds. Morgan Kaufmann, San Francisco, CA, 1559–1566.

Hart, E., Ross, P., and Nelson, J. 1998. Producing robust schedules via an artificial immune system. In *Proceedings of International Conference on Evolutionary Computing*. IEEE Press, Anchorage, Alaska, 464–469.

He, . L. and Mort, N. 2000. Hybrid genetic algorithms for telecommunications network back-up routing. *BT Technology Journal 18,* 4, 42–56.

Hertz, A. and Widmer, M. 2003. Guidelines for the use of meta-heuristics in combinatorial optimization. *European Journal of Operations Research 151(2)*, 247–252.

Hochberg, Y. and Tamhane, A. C. 1987. *Multiple Comparison Procedures*. Wiley.

Hoffman, K. and Padberg, M. 1985. Lp-based combinatorial problem solving. *Annals Operations Research 4*, 145–194.

Hoffman, K. and Padberg, M. 1991. Improving the lp-representation of zero-one linear programs for branch-and-cut. *ORSA Journal Computing 3*, 121–134.

Hoffman, K. and Padberg, M. 1993. Solving airline crew scheduling problems by branch-and-cut. *Management Science 39*, 657–682.

Holland, J. H. 1975. *Adaptation in Natural and Artifical Systems*. University of Michigan Press, Ann Arbor, MI.

Holland, J. H. 2000. Building blocks, cohort genetic algorithms, and hyperplane-defined functions. *Evolutionary Computation 8,* 4, 373–391.

Hooker, J. 1996. Testing heuristics: we have it all wrong. *Journal of Heuristics 1,* 1, 33–42.

Hromkovic, J. 2001. *Algorithmics for hard problems : introduction to combinatorial optimization, randomization, approximation, and heuristics.* Springer-Verlag.

Jakobi, N. 1997. Evolutionary robotics and the radical envelope-of-noise hypothesis. *Adapt. Behav. 6,* 2, 325–368.

Jin, Y. and Branke, J. 2005a. Evolutionary optimization in uncertain environments—a survey. *IEEE Transactions on Evolutionary Computation 9,* 3, 303–317.

Jin, Y. and Branke, J. 2005b. Evolutionary optimization in uncertain environments-a survey. *IEEE Trans. Evolutionary Computation 9,* 3, 303–317.

Jin, Y. and Sendhoff, B. 2004. Constructing dynamic optimization test problems using the multi-objective optimization concept. In *Applications of evolutionary computing*, G. R. Raidl, Ed. LNCS, vol. 3005. Springer, 525–536.

Johnson, D., Papadimitriou, C., and Yannakakis, M. 1988. How Easy Is Local Search? *Journal of Computer and System Sciences 37,* 1, 79–100.

Johnson, D. S. 2002. A theoretician's guide to the experimental analysis of algorithms. In *the 5th and 6th DIMACS Implementation Challenges*, D. S. J. M. Goldwasser and C. C. McGeoch, Eds. American Mathematical Society, 215–250.

Jones, T. 1995. Evolutionary algorithms, fitness landscapes and search. Ph.D. thesis, The University of New Mexico, Albuquerque, NM.

Kapur, J. N. and Kesavan, H. K. 1992. *Entropy Optimization Principles with Applications*. Academic Press, San Diego.

Karp, R. M. 1972. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher, Eds. Plenum Press, 85–103.

Koonar, G., Areibi, S., and Moussa, M. 2002. Hardware Implementation of Genetic Algorithms for VLSI CAD Design. In *15th International Conference on Computer Applications in Industry and Engineering*. ISCA, San Diego, California, 197–200.

Koza, J. R., Forest H. Bennett, I., Hutchings, J. L., Bade, S. L., Keane, M. A., and Andre, D. 1998. Evolving computer programs using rapidly reconfigurable field-programmable gate arrays and genetic programming. In *FPGA '98: Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays*. ACM Press, New York, NY, USA, 209–219.

Krasnogor, N. and Smith, J. 2005. A tutorial for competent memetic algorithms: model, taxonomy, and design issues. *IEEE Transactions on Evolutionary Computation 9,* 5, 474– 488.

Laguna, M. and Marti, R. 2003. *Scatter Search: Methodology and Implementation in C*. Kluwer Academic Publishers, Boston,.

Lewis, J., Hart, E., and Ritchie, G. 1998. A comparison of dominance mechanisms and simple mutation on non-stationary problems. In *Parallel Problem Solving from Nature – PPSN V*, A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, Eds. Springer, Berlin, 139–148. Lecture Notes in Computer Science 1498.

Lin, S.-C., Goodman, E. D., and Punch, W. F. 1997. A genetic algorithm approach to dynamic job shop scheduling problems. In *Seventh International Conference on Genetic Algorithms*, T. Bäck, Ed. Morgan Kaufmann, 481–488.

Louis, S. J. and Johnson, J. 1997. Solving similar problems using genetic algorithms and case-based memory. In *Proc. of The Seventh Int. Conf. on Genetic Algorithms*. Morgan Kaufmann, San Mateo, CA, 283–290.

Louis, S. J. and Xu, Z. 1996. Genetic algorithms for open shop scheduling and rescheduling. In *ISCA 11th Int. Conf. on Computers and their Applications*, M. E. Cohen and D. L. Hudson, Eds. 99–102.

Lourenço, H., Martin, O., and Stützle, T. 2001. A beginner's introduction to iterated local search. In *Proceedings of the Fourth Metaheuristics International Conference*. Vol. 1. 1–6.

Lourenco, H. R., Martin, O., and Stützle, T. 2002. Iterated local search. In *Handbook of Metaheuristics*, F. Glover and G. A. Kochenberger, Eds. Kluwer.

Maniezzo V., A. C. 2002. Ant colony optimization: An overview. In *In Essays and Surveys in Metaheuristics*, C. Ribeiro and P. Hansen, Eds. Kluwer, 469–492.

McIlhagga, M., Husbands, P., and Ives, R. 1996. A comparison of search techniques on a wing-box optimisation problem. In *Parallel Problem Solving from Nature – PPSN IV*, H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, Eds. Springer, Berlin, 614–623.

Merz, P. 2004. Advanced fitness landscape analysis and the performance of memetic algorithms. *Evol. Comput. 12,* 3, 303–325.

Merz, P. and Freisleben, B. 1997. Genetic local search for the TSP: New results. In *IEEECEP: Proceedings of The IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*.

Michalewicz, Z. 1992. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlog, Berlin, Heidelberg.

Mitchell, M. 1996. *An Introduction to Genetic Algorithms*. The MIT Press, Cambridge, Massachusetts.

Mori, N., Imanishi, S., Kita, H., and Nishikawa, Y. 1997. Adaptation to changing environments by means of the memory based thermodynamical genetic algorithm. In *International Conference on Genetic Algorithms*, T. Bäck, Ed. Morgan Kaufmann, 299–306.

Mori, N., Kita, H., and Nishikawa, Y. 1996. Adaptation to a changing environment by means of the thermodynamical genetic algorithm. In *Parallel Problem Solving from Nature – PPSN IV (Berlin, 1996) (Lecture Notes in Computer Science 1141)*, H. Voigt, W. Ebeling, and I. Rechenberg, Eds. Springer, Berlin, 513–522.

Mori, N., Kita, H., and Nishikawa, Y. 1998. Adaptation to a changing environment by means of the feedback thormodynamical geentic algorithm. In *Parallel Problem Solving from Nature – PPSN V*, A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, Eds. Springer, Berlin, 149–158. Lecture Notes in Computer Science 1498.

Morrison, R. 2003. Performance measurement in dynamic environments. In *GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems*, J. Branke, Ed. 5–8.

Morrison, R. W. and De Jong, K. A. 1999. A test problem generator for non-stationary environments. In *1999 Congress on Evolutionary Computation*. IEEE Service Center, Piscataway, NJ.

Morrison, R. W. and Jong, K. A. D. 2000. Triggered hypermutation revisited. In *Congress on Evolutionary Computation*. 1025–1032.

Moscato, P. 1999. Memetic algorithms: A short introduction. In *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, Eds. McGraw-Hill, London, 219–234.

Ng, K. P. and Wong, K. C. 1995. A new diploid scheme and dominance change mechanism for non-stationary function optimization. In *Sixth International Conference on Genetic Algorithms*. Morgan Kaufmann, 159–166.

Nordin, P. and Banzhaf, W. 1996. An on-line method to evolve behavior and to control a miniature robot in real time with genetic programming. *Adaptive Behavior 5,* 2 (Fall), 107–140.

Oliver, I. M., Smith, D. J., and Holland, J. R. C. 1987. A study of permutation crossover operators on the travelling salesman problem. In *Genetic algorithms and their applications : Proc. of the second Int. Conf. on Genetic Algorithms*, J. J. Grefenstette, Ed. Lawrence Erlbaum Assoc., Hillsdale, NJ, 224–230.

Oppacher, F. and Wineberg, M. 1999. The shifting balance genetic algorithm: Improving the GA in a dynamic environment. In *Proc. of the Genetic and Evolutionary Computation Conf. GECCO-99*, W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, Eds. Morgan Kaufmann, San Francisco, CA, 504–510.

Osman, I. H. and Laporte, G. 1996. Metaheuristics: A bibliography. *Annals Operations Research 63*, 513–623.

Padberg, M. and Rinaldi, G. 1991. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Rev. 33,* 1, 60–100.

Parker, R. and Rardin, R. 1988. *Discrete Optimization*. Academic Press, Inc.

Parkess, R. 2005. *Collins internet linked dictionary of statistics*. HarperCollins Publishers, Glasgow.

Parmee, I. 1996a. Cluster-oriented genetic algorithms (cogas) for the identification of high-performance regions of design spaces. In *In Proceedings of EvCA96*.

Parmee, I. 1996b. The maintenance of search diversity for effective design space decomposition using cluster- orientated genetic algorithms (cogas) and multi-agent strategies (gaant). In *Proceedings of 2nd International Conference on Adaptive Computing in Engineering Design and Control*, I. Parmee, Ed.

Pham, D. T. and Karaboga, D. 2000. *Intelligent Optimization Techniques*. Springer-Verlag, USA.

Psaraftis, H. 1995. Dynamic vehicle routing: Status and prospects. *Annals Operations Research 61*, 143–164.

Ramsey, C. L. and Grefenstette, J. J. 1993. Case-based initialization of genetic algorithms. In *Proc. of the Fifth Int. Conf. on Genetic Algorithms*, S. Forrest, Ed. Morgan Kaufmann, San Mateo, CA, 84–91.

Rand, W. and Riolo, R. 2005. Measurements for understanding the behavior of the genetic algorithm in dynamic environments: A case study using the shaky ladder hyperplane-defined functions. In *GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization*, S. Yang and J. Branke, Eds.

Rardin, R. 1998. *Optimization In Operation Research*. Prentice-Hall, Inc.

Rechenberg, I. 1973. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. frommann-holzbog, Stuttgart. German.

Reeves, C. and Karatza, H. 1993. Dynamic sequencing of a multi-processor system: a genetic algorithm approach. In *Artificial Neural Nets and Genetic Algorithms*, R. F. Albrecht, C. R. Reeves, and N. C. Steele, Eds. Springer, 491–495.

Reeves, C. R. 1992. A hardware implementation of the compact genetic algorithm. In *Proc. IEE Colloquium on Genetic Algorithms for Control and Systems Engineering*. Number 1992/106. IEE, London.

Reeves, C. R. and Rowe, J. E. 2002. *Genetic Algorithms: Principles and Perspectives: A Guide to GA Theory*. Kluwer Academic Publishers, Norwell, MA, USA.

Reinelt, G. 1991. TSPLIB — a traveling salesman problem library. *ORSA Journal on Computing 3*, 376 – 384.

Riget, J. and Vesterstroem, J. 2002. A diversity-guided particle swarm optimizer - the arpso.

Salomon, R. and Eggenberger, P. 1997. Adaptation on the evolutionary time scale: A working hypothesis and basic experiments. In *Artificial Evolution*. 251–262.

Siegel, S. 1956. *Nonparametric Statistics for the Behavioral Sciences*. McGraw-Hill Book Company, Inc., New York.

Sörensen, K. and Sevaux, M. 2004. MA—PM: Memetic algorithms with population management. *Computers and Operations Research*. In Press, available online 26 November 2004.

Stanhope, S. A. and Daida, J. M. 1999. Genetic algorithm fitness dynamics in a changing environment. In *Congress on Evolutionary Computation*. Vol. 3. IEEE, 1851–1858.

Stützle, T. 1999. Local search algorithms for combinatorial problems—analysis, algorithms and new applications. *DISKI—Dissertationen zur Knstliken Intelligenz*.

Tanese, R. 1989. Distributed genetic algorithm. In *Proc. of the Third Int. Conf. on Genetic Algorithms*, J. D. Schaffer, Ed. Morgan Kaufmann, San Mateo, CA, 434–439.

Tjørnfelt-Jensen, M. and Hansen, T. K. 1999. Robust solutions to job shop problems. In *1999 Congress on Evolutionary Computation*. IEEE Service Center, Piscataway, NJ, 1138–1144.

Trojanowski, K. and Michalewicz, Z. 1999. Searching for optima in non-stationary environments. In *1999 Congress on Evolutionary Computation*. IEEE Service Center, Piscataway, NJ, 1843–1850.

Trojanowski, K. and Michalewicz, Z. 2000. Evolutionary optimization in non-stationary environments. *Journal of Computer Science and Technology 1,* 2, 93–124.

Ursem, R. K. 1999. Multinational evolutionary algorithms. In *1999 Congress on Evolutionary Computation*. IEEE Service Center, Piscataway, NJ, 1633–1640.

URSEM, R. K. 2000. Multinational GAs: Multimodal optimization techniques in dynamic environments. In *Proc. of the Genetic and Evolutionary Computation Conf. (GECCO-00)*, D. Whitley, D. Goldberg, E. Cantu-Paz, L. Spector, I. Parmee, and H.-G. Beyer, Eds. Morgan Kaufmann, San Francisco, CA, 19–26.

URSEM, R. K. 2002. Diversity-guided evolutionary algorithms. In *Proceedings of Parallel Problem Solving from Nature VII (PPSN-2002)*. Springer Verlag, 462–471.

URSEM, R. K., KRINK, T., JENSEN, M. T., AND MICHALEWICZ, Z. 2002. Analysis and modeling of control tasks in dynamic systems. *IEEE Trans. Evolutionary Computation 6,* 4, 378–389.

VAN LAARHOVEN, P. J. M. AND AARTS, E. H. L. 1985. Statistical cooling: A general approach to combinatorial optimization problems. *Philips J. Res.* 40, 193–226.

VAVAK, F., FOGARTY, T. C., AND JUKES, K. 1996. A genetic algorithm with variable range of local search for tracking changing environments. In *Parallel Problem Solving from Nature*, H.-M. Voigt, Ed. Number 1141 in LNCS. Springer Verlag Berlin.

VENKATASUBRAMANIAN, V. AND ANDROULAKIS, I. 1991. A Genetic Algorithm Framework for Process Design and Optimization. *Computers Chemical Engineering 15,* 4, 217–228.

VOSS, S., MARTELLO, S., OSMAN, I. H., AND ROUCAIROL, C. 1999. On evolution strategy optimization in dynamic environments. In *Meta-Heuristics—Advances and Trends in Local Search Paradigms for Optimization*.

VOSE, M. D. 1999. *The simple genetic algorithm: foundations and theory*. MIT Press, Cambridge, MA.

VOUDOURIS, C. AND TSANG, E. 1999. Guided local search. *European Journal of Operations Research 113,2*, 468–499.

WANG, C., GHENNIWA, H., AND SHEN, W. 2005. Heuristic scheduling algorithm for flexible manufacturing systems with partially overlapping machine capabilities. In *Proc. of 2005 IEEE International Conference on Mechatronics and Automation,*. 1139 1144.

WEICKER, K. 2000. An analysis of dynamic severity and population size. In *PPSN VI: Proceedings of the 6th International Conference on Parallel Problem Solving from Nature*. Springer-Verlag, London, UK, 159–168.

Weicker, K. 2002. Performance measures for dynamic environments. In *Parallel Problem Solving from Nature*, J. Merelo, P. Adamidis, H.-G. Beyer, J. Fernández-Villacañas, and H.-P. Schwefel, Eds. LNCS, vol. 2439. Springer, 64–73.

Weicker, K. and Weicker, N. 1999. On evolution strategy optimization in dynamic environments. In *Congress on Evolutionary Computation*. Vol. 3. 2039–2046.

Whitley, D. 2001. An overview of evolutionary algorithms: practical issues and common pitfalls. *Information and Software Technology 43,* 14, 817–831.

Whitley, D., Rana, S., Dzubera, J., and Mathias, K. E. 1996. Evaluating evolutionary algorithms. *Artif. Intell. 85,* 1-2, 245–276.

Whitley, D., Rana, S., and Heckendorn, R. 1999. The Island Model Genetic Algorithm: On Separability, Population Size and Convergence. *Journal of Computing and Information Technology 7,* 1, 33–47.

Whitley, D. and Starkweather, T. 1990. Genitor ii.: a distributed genetic algorithm. *J. Exp. Theor. Artif. Intell. 2,* 3, 189–214.

Whitley, D., Starkweather, T., and Shaner, D. 1991. The traveling salesman and sequence scheduling: Quality solutions using genetic edge recombination. In *Handbook of Genetic Algorithms*, L. Davis, Ed. Van Nostrand Reinhold, New York, 350–372.

Wiesmann, D., Hammel, U., and Bäck, T. 1998. Robust design of multilayer optical coatings by means of evolutionary algorithms. *IEEE Trans. on Evolutionary Computation 2,* 4, 162–167.

Wineberg, M. and Oppacher, F. 2000. Enhancing the ga's ability to cope with dynamic environments. In *GECCO*. 3–10.

Wolpert, D. H. and Macready, W. G. 1997. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation 1,* 1 (April), 67–82.

Yang, S. 2003. Non-stationary problem optimization using the primal-dual genetic algorithm. In *Proceedings of the 2003 Congress on Evolutionary Computation CEC2003*, R. Sarker, R. Reynolds, H. Abbass, K. C. Tan, B. McKay, D. Essam, and T. Gedeon, Eds. IEEE Press, Canberra, 2246–2253.

Yang, S. 2004. Constructing dynamic test environments for genetic algorithms based on problem difficulty. In *Congress on Evolutionary Computation*. Vol. 2. 1262–1269.

Yang, S. and Yao, X. 2003. Dual population-based incremental learning for problem optimization in dynamic environments. In *Proceedings of the 7th Asia Pacific Symposium on Intelligent and Evolutionary Systems*, I. M. G. et. al., Ed. 49–56.

Younes, A. 2002. Evolutionary computing and agent-orientation for multidisciplinary optimization. presented as part of a poster at Materials and Manufacturing Ontario (MMO) , Partnerships 2002, Toronto.

Younes, A. and Basir, O. 2002. A hybridized genetic algorithm for solving the dynamic vehicle routing problem. presented at the 2nd Annual McMaster Optimization Conference: Theory and Applications (MOPTA 02), Hamilton, Canada.

Younes, A., Basir, O., and Calamai, P. 2003. A benchmark generator for dynamic optimization. In *the 3rd WSEAS International Conference on Soft Computing, Optimization, Simulation & Manufacturing Systems*. WSEAS, Malta.

Younes, A., Basir, O., and Calamai, P. 2004. A benchmark generator for dynamic optimization. *WSEAS Transactions on SYSTEMS 3,* 1, 273–278.

Younes, A., Basir, O., and Calamai, P. 2005. Approaching dynamic combinatorial problems by controlling population diversity. (MIC2005) The 6th Metaheuristics International Conference, Vienna, Austria.

Younes, A., Basir, O., and Calamai, P. 2006a. Adaptive control of genetic parameters for dynamic combinatorial problems. accepted as a book chapter of Metaheuristics.

Younes, A., Basir, O., and Calamai, P. 2006b. A hybrid evolutionary approach for combinatorial problems in dynamic environments. In *IEEE CCECE 2006*. IEEE, OTTAWA, CANADA.

Younes, A., Calamai, P., and Basir, O. 2005. Generalized benchmark generation for dynamic combinatorial problems. In *Genetic and Evolutionary Computation Conference (GECCO2005) workshop program*, F. Rothlauf, M. Blowers, J. Branke, S. Cagnoni, I. I. Garibay, O. Garibay, J. Grahl, G. Hornby, E. D. de Jong, T. Kovacs, S. Kumar, C. F. Lima, X. Llorà, F. Lobo, L. D. Merkle, J. Miller, J. H. Moore, M. O'Neill, M. Pelikan, T. P. Riopka, M. D. Ritchie, K. Sastry, S. L. Smith, H. Stringer, K. Takadama, M. Toussaint, S. C. Upton, and A. H. Wright, Eds. ACM Press, Washington, D.C., USA, 25–31.

Younes, A., Calamai, P., and Basir, O. 2006. Issues in constructing dynamic combinatorial test problems. submitted to the journal: transactions on evolutionary computation. Submitted to the journal: transactions on Evolutionary computation (August 2006).

Younes, A., Ghenniwa, H., and Areibi, S. 2002. An adaptive genetic algorithm for multi objective flexible manufacturing systems. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO2002)*. Morgan Koffman, New York, New York, 1241–1249.

Zhu, K. Q. 2003. A diversity-controlling adaptive genetic algorithm for the vehicle routing problem with time windows. In *ICTAI*. 176–183.

Zhu, K. Q. and Liu, Z. 2004. Population diversity in permutation-based genetic algorithm. In *ECML*. 537–547.