Adapting Operator Probabilities In Genetic Algorithms

Andrew Laurence Tuson

MSc Information Technology: Knowledge Based Systems

Department of Artificial Intelligence

University of Edinburgh

1995

Abstract

In the vast majority of genetic algorithm implementations, the operator probabilities are fixed throughout a given run. However, it can be convincingly argued that these probabilities should vary over the course of a genetic algorithm run — so as to account for changes in the ability of the operators to produce children of increased fitness. This dissertation describes an empirical investigation into this question. The effect upon genetic algorithm performance of adaptation methods upon both well-studied theoretical problems, and a hard problem from Operations Research — the flowshop sequencing problem, is examined.

Acknowledgements

I would first of all like to thank my supervisor, Dr Peter Ross, for his valuable guidance of the research reported here. Thanks also to Dave Corne and the members of the DAI EC group for their occasional advice.

Thanks to the other MScs who have provided a *very* enjoyable working atmosphere. Special thanks to Michael L, Emma C, Mike F, Richard W and Larry F (for proving that heartless Canadians are the exception rather than the rule!).

Mention and gratitude must also go to Pete, Daren, Jon, Evonne and Rob for their entertaining and supportive email. I also have to thank Dave and Elaine for providing the same support over more mundane communication methods!

I am grateful to Keith and Frankie Tuson for their encouragement and financial support during my academic career.

Extra special thanks must go to Dr Hugh Cartwright of the Physical and Theoretical Chemistry Laboratory at Oxford University. His supervision of my chemistry Part II research project proved to be a turning point for me, and inspired my interest in AI - no amount of malt whisky can express my gratitude enough.

Finally, I thank the EPSRC who provided financial support during my year of study via studentship no: 94415692.

Dedication

I would like to dedicate this dissertation to my family: Laurence, Margaret and Karen Tuson who have supported and encouraged me throughout my academic career.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Genetic algorithms (GAs) are a stochastic optimisation technique first proposed by John Holland [Holland 75] and loosely based upon the process of evolution by natural selection, first suggested by Darwin in "The Origin Of The Species" [Darwin 59]. Since then, genetic algorithms have found applications[1] in previously difficult or intractable problems such as atmospheric pollution monitoring [Cartwright & Harris 93], and scheduling [Fang 92].

Genetic algorithms usually apply two operators to a population to produce new solutions which then undergo selection: crossover, analogous to sexual reproduction, where information is exchanged between two candidate solutions; and mutation, analogous to asexual reproduction, where a solution is randomly modified. These operations are applied at a fixed probability throughout the run of a genetic algorithm.

There is no reason why this has to be the case. In fact, a moments thought would suggest that it may be beneficial to adaptively change these probabilities *on-line* during a genetic algorithm run — to use more often the operator(s) that are producing, on average, fitter children. This dissertation investigates whether this approach can produce a more effective optimisation technique.

[1] [Ross & Corne 95] provides an overview of genetic algorithm applications

1.2 Genetic Algorithms - A Quick Guide

This section provides a brief introduction to the genetic algorithm. A reader new to the field would be advised to read one of the introductory texts such as [Goldberg 89].

This section begins with an outline of the constituent parts of a genetic algorithm, and is followed by a description of a typical implementation. Then each of the constituent parts will be discussed in turn. Good genetic algorithm design requires the consideration of *all* of these constituents and their interactions.

1.2.1 The Basic Ingredients

To implement a genetic algorithm for a problem we need to design the following:

- A representation of candidate solutions as a string of some sort, typically numbers.
- A population of candidate solutions (also referred to as chromosomes or strings).
- A set of operators to generate new candidate solutions from members of the population, along with information as to how often they should be applied.
- An evaluation function to assess the quality (fitness) of a given candidate solution.
- A selection method which gives good solutions a better chance of survival.

The genetic algorithm is applied iteratively — generating new candidate solutions from the population by the application of operators, evaluating them, and then allowing the fittest of the available solutions to comprise the new population.

The following sections will describe the genetic algorithm and its components in more detail. Also some of the tradeoffs and design decisions that have to be made will be highlighted.

1.2.2 The Algorithm

The exact implementation of a genetic algorithm varies somewhat — the following sequence of steps describes the algorithm for a simple genetic algorithm with *steady-state reproduction*[2]:

1. Generate an initial population of candidate solutions, usually at random.

2. Apply the evaluation function to each member of the population.

3. Select *parent* solutions according to their fitness (i.e. fitter solutions are more likely to become parents). In this case selection is performed by using *rank-based* selection - based upon the ranked fitnesses of the solutions in the population.

4. Apply an operator to generate a new candidate solution.

5. The new candidate solution is then evaluated and if fitter than the worst member of the population, replaces it.

6. Go back to step 3 until a stopping criterion is reached. Examples of stopping criteria are: all members of the population are identical (convergence), a fixed number of evaluations have been computed, or a solution of a given quality has been found.

1.2.3 Representation

How to represent the candidate solutions in a form that the genetic algorithm can manipulate, is the first component to decide upon when attempting any problem, as other decisions (i.e. the choice of operator) are dependent upon this. The genetic algorithm designer has to encode the required information so that correlations exist in the search space, that the GA can exploit.

---

[2] A population model that will be used in the experiments detailed in this dissertation

A 'traditional' implementation of a genetic algorithm uses only binary strings, partly due to a misunderstanding of the *schema theorem* [Holland 75] which describes the GA in terms of *schema* — a formal way of expressing a portion of a candidate solution. For instance, 11****** specifies a portion (schema) of a string that starts with 11, and the rest of the solution taking any allowed value. This theory concludes that low cardinality strings maximise the number of schemata that crossover (see 1.2.4) processes — this ignores the fact that if the resulting representation contains no useful correlations, then the genetic algorithm will not perform better than random search[3].

Unfortunately, some researchers still assume that genetic algorithms equate with encoding candidate solutions in the form of binary strings — this is simply not true. Failure to recognise the effect that representation has on the effectiveness of a optimisation technique has made some comparative studies of genetic algorithms with other techniques, such as simulated annealing, at best worthless, and, at worst, misleading [Ingber & Rosen 92].

Practical applications of genetic algorithms should use whichever encoding is appropriate to the problem (an encoding used by an existing method is usually a good start) — this could be a string of real numbers, a permutation, a LISP S-expression; whatever is found to work for the problem at hand.

1.2.4 Operators

In the basic genetic algorithm, there are two main types of operator: crossover and mutation, drawn from the biological metaphors of sexual and asexual reproduction respectively.

Each operator available to the genetic algorithm has a probability of being fired (henceforth referred to as an operator probability). This study makes a distinction between this and any parameters associated with a given operator (henceforth referred to as an operator parameter). For example, a GA could be constructed

[3]No optimisation without representation! See [Radcliffe 92] for a more detailed discussion of this issue.

so that applied parameterised uniform crossover is used 70% of the time, with parameter 0.1 (which means that 90% of the child's genes are from the first parent chosen), along with mutation 30% of the time, with a bitwise mutation rate of 0.02. The term 'operator settings' will be taken to mean both of the terms above.

Finally, another term that will be used in this study is 'control parameter', which refers to all of the genetic algorithm design decisions: operator settings, selection pressure, population size, etc.

Both crossover and mutation will be discussed in more detail below.

Crossover

This is simply an exchange of information between two (rarely more) candidate solutions in the population. This is best illustrated by considering a commonly-used crossover operator for binary strings.

Two-point crossover picks two points, at random, along the strings and swaps the contents of the string between those points, to produce a child as shown in Figure 1–1.

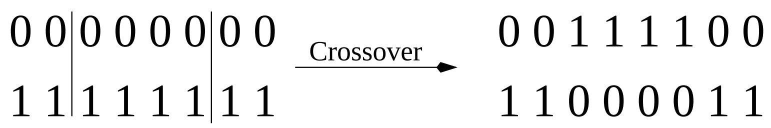


Figure 1–1: Two-Point Crossover

The usual rationale for why crossover can be a useful search operator is that the recombinative process can bring together portions of separate strings associated with high fitness to produce even fitter children (and conversely, bring poor portions of strings together so they can be purged from the population).

This is the basis of the *building block hypothesis* [Goldberg 89] — a formal statement of the ability of crossover to combine portions of different candidate solutions (schema or building blocks) associated with high fitness to produce fitter children.

Mutation

This can be simply described as taking a given candidate solution and randomly changing part of it. In combination with selection this effectively performs a form of hill-climbing. For a binary string, this involves flipping each bit in the string with a (low) probability p_m as illustrated in Figure 1–2 below:

$$0\ 0\ 0\ 0\ 0\ \boxed{0}\ 0\ 0 \xrightarrow{\text{Mutate}} 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0$$

Figure 1–2: Binary Mutation

Early genetic algorithm research viewed mutation as a background operator, used infrequently in order to restore diversity in the population that has been destroyed by selection and drift. However, the tide of argument has turned in the opposite direction — partly due to the fact that mutation-only optimisation techniques such as simulated annealing can obtain comparable results to a genetic algorithm. Mutation now is seen to be a search operator in its own right.

1.2.5 The Fitness Function

The role of the fitness function is to provide a measure of the quality of a candidate solution. Usually, devising a suitable function is quite straightforward, but not always. For example, in constraint satisfaction problems such as timetabling [Corne *et al.* 93] both soft and hard constraints have to be weighed against each other, requiring some form of penalty function, or other such means to combine the differing objectives into a single fitness value.

1.2.6 Population and Selection

This is where the above components are brought together. As stated above, the genetic algorithm works upon a population of candidate solutions and selects the most suitable; however, there are choices to be made:

- The size of the population (large populations have a larger and more diverse number of candidate solutions, but take longer to evaluate). This is usually fixed.

- Whether to arrange the candidate solutions spatially, in order to emulate the process of geographical speciation, and maintain diversity between solutions.

- Should an entirely new population of children be generated (a *generational* GA), or should children be incrementally inserted into the current population (a *steady-state* GA)?

- What selection scheme (e.g. ranking, fitness-proportionate) should be used? Issues here include the selection pressure, and the potential for parallel implementation.

- What selection pressure should be applied? Too high, and the population will quickly converge to a poor solution, too low, and the GA will take too long to find a solution of the required quality.

1.3 Why Adapt Operator Probabilities?

Adaptation of operator probabilities is an attempt to make the genetic algorithm a more effective optimiser. But what is meant by this? There are possible criteria by which adapting operator probabilities could be of benefit:

1. Reducing the amount of time that is spent finding suitable values for GA control parameters.

2. Increasing the quality of solutions obtained.

3. Allowing the GA to find a solution of a given quality more quickly.

It is agreed in the genetic algorithm community that the performance of the genetic algorithm is dependent upon the operator probabilities used. It is also agreed that finding appropriate operator probabilities is quite hard, especially when it is realised that these vary with the problem being considered, and the other components of the genetic algorithm.

There is no reason why operator probabilities must be fixed. One argument against this is that the ability of an operator to produce new, fitter, solutions is not only dependent upon the problem being attempted, but may also vary depending upon the current state of the population (for example: if all the solutions in the population are identical, then crossover is useless).

This idea that an effective assignment of operator probabilities may vary with time, suggests that a time-varying set of probabilities may improve solution quality, or the computational effort required to find a solution of a given quality, and that a static set of operator probabilities is, in effect, a compromise.

Would it be possible to develop a schedule in advance which applies the operators at the appropriate probabilities, at the right time?

This is seen to be to be difficult when you take into account that the process of finding appropriate operator probabilities for the basic, static, case is itself quite hard — varying with the population size, problem, selection method, representation, and the operators used. In many cases, a through search of the probability space would be more difficult than the problem that is to be solved — hence researchers tend to adopt operator probabilities based upon early work by DeJong [DeJong 75] and Grefensette [Grefensette 86], however these may well be suboptimal for the problem at hand.

Therefore, it may be advantageous to employ some form of method that attempts to automatically adjust the operator probabilities as the genetic algorithm runs, according to the abilities of the operators present to produce children of improved fitness. Work performed in this area previously has fallen into two groups:

- The direct encoding of operator probabilities into each member of the population, allowing them to 'co-evolve' with the solutions.

- The use of a global 'learning-rule' to adapt operator probabilities according to the quality of solutions generated by each operator.

Unfortunately, no comparisons have been made between these approaches. Also, the *ad hoc* nature of many of these investigations leave doubts as to whether the work performed is generally applicable, either because of the simple nature of the problems being attempted, or the lack of statistical rigour employed.

However, most work in this area has concentrated on whether these methods result in adaptation taking place, with the implication that this must be desirable. The problem with this approach is that it does not address the 'bottom line' - does applying operator adaptation to the genetic algorithm make it a better optimiser?

The aim of the project is therefore to conduct a systematic investigation into such methods in order to ascertain their utility.

1.4 Outline Of This Dissertation

This chapter has introduced the genetic algorithm paradigm, and highlighted some of the issues involved in the design of effective algorithms. The difficulty of setting effective operator probabilities was then expressed. It was argued that possible gains exist if these probabilities could be adjusted adaptively by the genetic algorithm, to account for the possibility that the appropriate operator settings may be time-varying.

Chapter 2, *A Guide to Operator Adaptation* will firstly review the debate about the relative merits of the two main operators used in GAs, crossover and mutation, and argue that each are valid search operators in their own right. The difficulty in setting good operator probabilities will also be discussed in more detail. This will set the scene for the main part of this chapter: a literature review on methods for dynamic adaptation of operator probabilities.

Chapter 3, *The Test Problems* will introduce the reader to part of the methodology employed in this work: namely the use of a variety of test problems, ranging from well-studied theoretical problems, to a hard Operations Research/Scheduling problem. A review of previous work performed on these problems will be provided and an overview of the GA operators used for each of the test problems given.

Chapter 4, *A Genetic Algorithm For Operator Adaptation*, describes in detail the genetic algorithm used in this investigation, and the operator adaptation methods implemented.

Chapter 5, *An Investigation Of Adaptation Using Co-evolution*, examines adaptation techniques based upon the co-evolutionary metaphor. The performance of the methods will be compared against a genetic algorithm with fixed operator probabilities. Also, the question of whether useful adaptation actually takes place will be examined.

Chapter 6, *An Investigation Of Adaptation Using COBRA*, examines the simplest of the 'learning rule' adaptation methods, COBRA, and compares its performance against a genetic algorithm with fixed operator probabilities.

Chapter 7 *Conclusion*, ties together the work described in previous chapters and attempts to draw general conclusions about the effectiveness of operator adaptation.

Chapter 8 *Further Work*, outlines issues that where raised in this study and describes additional work that could be performed in order to answer these outstanding questions.

Chapter 2

A Guide To Operator Adaptation

2.1 Overview

The issue of dynamic operator adaptation has firm roots in both the problems experienced by genetic algorithm practitioners in finding effective operator probabilities, and in the arguments on the relative virtues of crossover and mutation. A brief review of both topics will be provided.

Next, support for the idea that the most effective operator settings should be varied during a genetic algorithm run will be presented.

Finally, the issue of operator adaptation proper will then be reviewed. The work can be divided by two criteria: the type of adaptation method used, and whether the settings are being adapted at the level of the population or the individual solution. The review will be organised by the first of the criteria.

2.2 Finding Effective Operator Settings

As discussed in Chapter 1, genetic algorithm performance can be measured by the following two criteria:

1. The quality of solutions obtained.

2. The time taken for the genetic algorithm to find a solution of a given quality.

It has long been acknowledged that the parameters that control a genetic algorithm can have a significant effect upon performance. However, finding good control parameters is still somewhat of a black art. With the large choice of designs and control parameters available, it is not feasible to do an exhaustive search of all the possibilities. Instead researchers usually use operator settings based upon a few empirical studies.

The first of these was performed by DeJong [DeJong 75] who set out to compare GAs with the best available gradient techniques. To do this he devised five test functions of various difficulties to gradient techniques. The following two measures where then used to evaluate GA performance:

- The *on-line* average — the average fitness of all solutions evaluated during the search.
- The *off-line* average — the average fitness of the best solutions in each generation.

Due to the lack of available computing resources, only a limited search of the design decisions was made. From these experiments a set of values was found that performed well over the suite of problems. The values obtained were: population size 50–100, crossover rate 0.60 (operator probability), and bitwise mutation probability[1] 0.001 (operator parameter). These became the conventional wisdom for genetic algorithm practitioners.

Another attempt was made by Grefenstette [Grefensette 86] who used a genetic algorithm to address the control parameter problem by treating it as a meta-problem. A 'meta-GA' was used to locate control parameters which where evaluated by running another genetic algorithm with these parameters to solve the actual problem. This approach provided an efficient way of search for the control parameters. However, no information as to how sensitive performance is to the set of parameters found was provided. The recommended values found in this study were: population size 30, crossover rate 0.95, and mutation rate 0.01.

A more recent study [Schaffer *et al.* 89] performed a more exhaustive search over a slightly wider range of problems and provided some evidence for problem-

[1] Often refered to as a mutation rate.

independence when setting control parameters. The values found in this instance were: population size 25–30, and crossover rate 0.75, and mutation rate 0.005–0.01.

Theoretical work by Hart and Belew [Hart & Belew 91] questions this claim of problem-independence, with a result which essentially means that the most effective set of control parameters must, at least, be dependent upon the fitness function.

However, even if the claims of function independence bear out in practice, the fact remains that many genetic algorithm implementations do not use the binary representation and operators used in these studies. Therefore it is not certain whether the results here can be usefully generalised to these applications.

2.3 Crossover Verses Mutation

From the very origins of the field, one of the major issues has been the relative importance of the two main genetic operators: crossover and mutation.

Each operator has its advocates: mutation is used as the primary operator in the 'Evolutionary Programming' and 'Evolution Strategy' [Bäck *et al.* 91] communities — implementations of which have also performed well as function optimisers; also the quality of results achieved by other mutation-only techniques such as simulated annealing support the claim that mutation alone can form the basis of an effective optimisation technique — other studies confirm this view [Schaffer *et al.* 89]. Interestingly, a recent study [Juels & Wattenberg 94] found that simple stochastic hill-climbing methods were able to achieve results comparable, or superior, to GAs for certain problems. At the extreme, Fogel [Fogel & Atmar 90] argues that crossover has no general advantage over mutation.

The opposing camp point to empirical studies [Schaffer & Eshelman 91], and [DeJong 75] conclude that the presence of crossover enhances performance. Most theoretical work on crossover takes this view, and relegates mutation to a background operator — a method to maintain diversity in the population.

The current consensus is more eclectic — crossover and mutation are seen as having different properties that complement each other. An example of this view is the theoretical study by Spears [Spears 93] which argues that crossover plays a more *constructive* role, preserving and combining useful information between solutions. However, mutation plays a more effective *disruptive* role. The argument concludes with the suggestion that mutation and crossover could be seen as two forms of a more general exploration operator that perturbs candidate solutions based upon the information available.

2.4 Why Vary?

There is evidence, both empirical and theoretical, that the most effective operator settings do vary during the course of a genetic algorithm run.

Two empirical studies with non-adaptive operator schedules which vary operator settings over the course of a genetic algorithm run indicate that improvements in performance can be attained. Work by Fogarty [Fogarty 89] studied a variety of schedules for the mutation operator parameter, for a genetic algorithm to optimise a rule-base for a multiple burner furnace. The work concluded that varying the mutation parameter improved genetic algorithm performance. Davis [Davis 91] advocates the use of a time-varying schedule of operator probabilities, and states that, in his experience, performance is improved — especially when a large number of operators are used.

Theoretical work [Mühlenbein 92] has analysed the mutation operator for a few binary coded problems and concluded that the mutation parameter should be decreased the nearer to the optimum the GA is. Time dependency was also discovered for mutation parameters by Hesser and Männer [Hesser & Männer 91].

So by what criterion should we judge when an operator is being more useful at a given point? I propose that the ability of an operator to produce new, preferably fitter, children may be what is required — this has been suggested before by [Spears & DeJong 91], but the emphasis here is on the potential of an operator to produce children of increased fitness: 'operator productivity'. Clearly this is necessary for optimisation to progress — the aim of the genetic algorithm is, after all, to uncover new, fitter, points in the search space. In fact, the overall performance of a genetic algorithm depends upon it maintaining an acceptable level of productivity throughout the search.

This concept is based upon work by Altenberg [Altenberg 94] which introduced the somewhat more general concept of evolvability — the ability of the operator/representation scheme to produce offspring that are fitter than their parents. This idea has some support from work by Mühlenbein discussed above which tried to derive an optimal value for the mutation parameter so as to maximise the probability of an improvement being made.

2.5 Adaptation Methods

Considering the argument made above, the purpose of dynamic operator adaption is to exploit information gained, either implicitly or explicitly, of the current ability of the operators to produce children of increased fitness.

Other methods do exist that adjust operator setting on other criteria, such as the diversity of the population[2], but these will not be considered in this study.

As mentioned earlier, operator adaptation methods fall into two groups:

- The direct encoding of operator probabilities into each member of the population, allowing them to 'co-evolve' with the solutions.
- The use of a 'learning-rule' to adapt operator probabilities according to the quality of solutions generated by each operator.

This classification will form the basis of the following discussion.

2.5.1 Co-evolutionary Methods

These methods encode the operator settings onto each member of the population and allow them to evolve with the candidate solutions. The rationale behind

[2]For example [Coyne & Paton 94].

this is as follows: solutions which have encoded operator probabilities that produce children of improved fitness will be the ancestors of the dominant members of later populations, and so the useful operator information will spread through the population. The converse is also true, members of the population with poor operator settings will eventually be weeded out by selection.

The original work in this area appears to have originated from the 'Evolution Strategy' community [Bäck *et al.* 91]. The mutation operator in such algorithms involves changing each gene by a value taken from a Gaussian distribution, with a standard deviation described by a gene elsewhere on the chromosome — this parameterises the amount of disruption that mutation produces when creating a child. These operator parameters are allowed to evolve to suitable values. Work extending this to adapting the mutation parameter for more conventional genetic algorithm implementations has reported some success [Bäck 91][Bäck 92], in the sense that the mutation parameter was seen to adapt to the theoretical optimum for the theoretically tractable (simple) problem being considered.

Similar work has encoded the crossover operator to use into each chromosome [Schaffer & Eshelman 91][Spears 95] — the GA was observed to evolve to use the more effective operator for that problem. Other publications detail the encoding of where to place crossover points [Schaffer & Morishima 87] [Levenick 95]. In both cases adaptation was seen to occur, and the former study reported performance as good as, or better than, a traditional genetic algorithm for DeJong's test suite.

2.5.2 Learning Rules

Another approach is to periodically measure the performance of the operators being used, and utilise this information to adjust the operator parameters accordingly. Previous work in this area has adjusted the operator probabilities on a population scale, but there seems no reason, in principle, why this method could not be applied at the level of individual chromosomes.

Three such techniques exemplify this approach. The first two bear some similarity, in that they attempt to pass credit onto not only the operator used to produce a given child, but also to the operators that where used to produce the child's ancestors. This is in order to credit operators that, although may not produce particularly good children, set the scene for another operator to make improvements. The third techinque does not do this; although there is no reason why such a feature could not be added, if desired.

The first was devised by Davis [Davis 89]. The precise algorithm is given there. The algorithm outlined appears quite complicated at first sight: records are kept for each member of the population about what operators were used to produce them and their ancestors, and any improvements that the operators were able to attain.

A similar technique that requires less bookkeeping is provided by Julstrom [Julstrom 95]. Each member of the population has a tree attached to it depicting the operators used to create it. When a child of improved fitness is produced, this tree is used to assign credit to each operator. A queue is also maintained that records the operators' credit over the most recent chromosomes.

Both methods periodically process this information to adjust the operator probabilities accordingly. All this requires a lot of additional bookkeeping, but no empirical evidence has been given to justify the added complexity. Both approaches are observed to adapt operator probabilities accordingly. Davis does not use his method on-line, but to obtain a non-adaptive time-varying schedule for later use — this was found to improve performance over a genetic algorithm with fixed operator probabilities.

The third of the learning-rule techniques is Cost Based Operator Rate Adaptation (COBRA) [Corne *et al.* 94], devised in order to improve performance in timetabling problems. Initial operator probabilities are provided and the genetic algorithm periodically swaps them between operators, giving the highest probability to the operator that has been producing the most gains in fitness. This adaptation process is described in more detail in 4.6. When applied to 4 real-life timetabling problems, a significant improvement in speed to solution was observed compared to a similar genetic algorithm with fixed operator probabilities.

The virtue of COBRA, as for co-evolutionary methods, is its simplicity. First, the information available about the usefulness of the operators present, due to the stochastic nature of the genetic algorithm, is very likely to be noisy — therefore it is possible that the more sophisticated adaptation methods may not live up to their potential as a result. Also, from a pragmatic point of view, simplicity is preferred as we do not want a potential method of problem-solving to become more complicated than the problem we are trying to solve!

2.6 Conclusion

A case has been made for the possible utility of dynamically adapting operator control parameters, in order to fully exploit the differing properties of crossover and mutation, and previous work reviewed. Also, the desirability of simpler adaptation methods was highlighted. Finally, no comparative study between these two approaches has been attempted — this will be addressed in this study.

Chapter 3

The Test Problems

3.1 Overview

In order to properly study the effectiveness (or otherwise) of dynamic operator adaptation methods a set of test problems need to be chosen. Such problems can be selected from two sources:

- Well-described theoretical problems with known properties: in order to help highlight interesting properties of the adaptation mechanism.
- Real world problems. As the ultimate aim of this study is to increase the overall usefulness of GAs, it would be pointless if operator adaptation methods were ineffective on these, more complex, problems.

Therefore, the first member of the test suite is a hard scheduling/operations research problem. The other members have been selected on account of their theoretical interest. The test problems selected are:

- The flowshop sequencing ($n/m/P/C_{max}$) problem.
- The 'Counting Ones' problem.
- Goldberg's order-3 'tight' deceptive problem.
- The 'Royal Road' problem.
- The 'Long Path' problem.

In this chapter the selected test problems are outlined, a brief review of relevant work performed for each problem given, and an overview of the representation and operators used by a genetic algorithm to attempt these problems provided.

3.2 The Flowshop Sequencing Problem

Scheduling problems are common in industry, of which one class is the flowshop sequencing, or $n/m/P/C_{max}$ problem. This problem will be used to assess the performance of operator adaptation in hard problems that the genetic algorithm would be expected to face in the real world.

3.2.1 The Flowshop

These are quite common in the chemical industry [Ku *et al.* 87], where the continuous (but efficient) large-scale production of products required in only small amounts is undesirable; but sporadic batch processing is inefficient. To overcome this, a production line approach is taken: chemical precursors are fed through a string of reactors joined in a serial fashion.

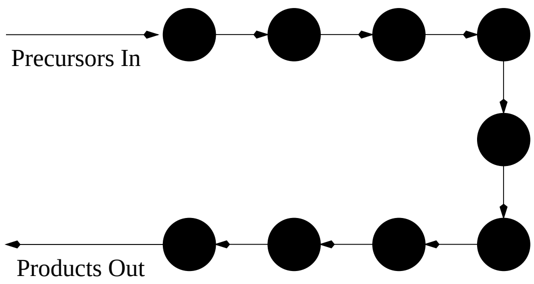


Figure 3–1: A Serial Flowshop

Flowshops operate in a *multi-product* fashion — different products can be synthesised by feeding the required precursors into the reactor and adding reagents as appropriate.

In all multi-product flowshops, each product has its own characteristic processing time in each reactor. Therefore, the efficiency of the line is dependent upon the order in which the products are produced — poor sequences can lead to temporary blockages in the flowline as completed stages cannot proceed, because a stage ahead in the flowline is still being processed (Figure 3–2). Obviously, this has a detrimental effect on throughput, and the sequence of jobs fed into the flowline is of importance.

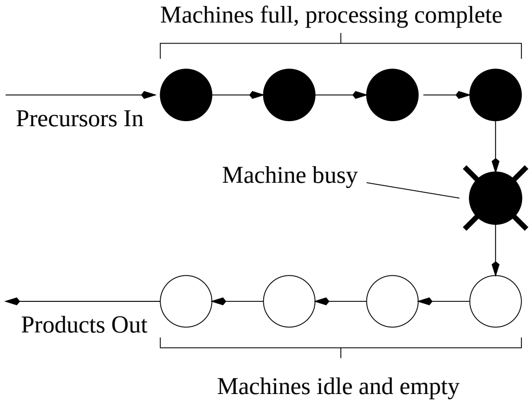


Figure 3–2: Blocking of a Flowshop at a Single Machine

3.2.2 The $n/m/P/C_{max}$ Problem

The task is therefore to find a sequence of jobs for the flowshop to process, so as to minimise the *makespan* — the time taken for the last of the jobs to be completed. This task is known to be NP-hard [Kan 76] (the number of possible sequences is $n!$) and can be formalised as follows: n jobs have to be processed (in the same order) on m machines; the aim is to find a job permutation $\{J_1, J_2, ..., J_n\}$ so as to minimise C_{max}.

C_{max} is defined as follows: given processing times $p(i, j)$ for job i on machine j and the job permutation above, we can find the completion times by the following equations:

$$C(J_1,1) = p(J_1,1)$$
$$C(J_i,1) = C(J_{i-1},1) + p(J_i,1)\ for\ i = 2,...,n$$
$$C(J_1,j) = C(J_1,j-1) + p(J_1,j)\ for\ j = 2,...,m$$
$$C(J_i,j) = max\{C(J_{i-1},j), C(J_i,j-1)\} + p(J_i,j)\ for\ i = 2,...,n;\ j = 2,...,m$$
$$C_{max} = C(J_n,m)$$

Standard benchmarks exist for this problem, and will be used in this study of operator adaptation. Taillard [Taillard 93] produced a set of test problems which, using a very lengthy Tabu Search procedure, were still inferior to their lower bounds, with problem sizes ranging from 20 jobs and 5 machines, to 500 jobs and 20 machines. There are 10 instances of each problem — all processing times were generated randomly from a $U(1,100)$ distribution. These problems are considered a tough test for any combinatorial optimisation algorithm.

3.2.3 Previous Work

This problem can be solved optimally for a 2-machine problem using Johnson's Rule [Johnson 54]. For the m machine problem, however, approximate methods must be used. Early work concentrated upon devising good 2-unit approximations to Johnson's rule (for example [Dannenbring 77]), with limited success — the techniques where not found to scale up to the problem sizes required. Other approaches to this problem include Linear Programming [Karimi & Ku 88], Tabu Search [Reeves 93] and Simulated Annealing [Osman & Potts 89].

Recently, work as shown that the genetic algorithm can be used to find suitable sequences. Cleveland and Smith [Cleveland & Smith 89] used a GA to sequence a computer board assembly and test facility. Work by Mott [Mott 90] examined the sequencing of chemical flowshops.

A comparative study using the Taillard benchmarks has been performed by Reeves [Reeves 95], who compared the genetic algorithm with simulated annealing, and a neighbourhood search heuristic. The study found that the GA found results comparable to simulated annealing, but performs better for large problem sizes.

3.2.4 Representation and Operators

The representation used for this problem is a permutation of the jobs to be placed into the flowline, numbered 1 to n. Various crossover operators exist for this representation: the most effective operator was found to be problem-specific [Starkweather *et al.* 91].

A feature of this representation is that simple multi-point or uniform crossover will often not produce a child that is not itself a permutation — to circumvent this it is common to add a 'repair' step to the operator to legalise the child. An example of this approach is the 'Modified PMX' operator devised by Cartwright and Mott [Mott 90].

This operator performs two-point crossover upon the two strings selected. The repair procedure then analyses one string for duplicates: when one is found it is replaced by the first duplicate in the second string. This process is repeated until both strings are legal (Figure 3–3).

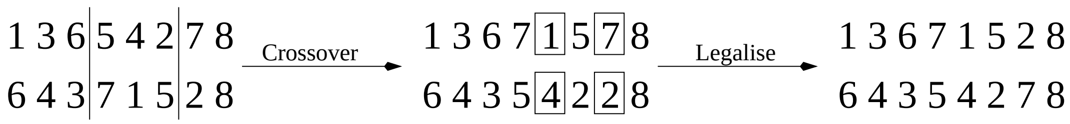


Figure 3–3: The Modified PMX Operator

This operator was designed with the intention of causing as little disruption as possible to the strings during repair — it was found to give good results when used in a genetic algorithm for this problem, and will be used in this study.

Two types of mutation operators are available for this representation. The first, termed *swap* mutation picks two elements of the permutation and swaps them (Figure 3–4).

The second type of mutation operator — *shift* mutation — randomly picks an element and shifts it to another randomly selected position (Figure 3–5).

Reeves [Reeves 95] compared the effectiveness of these two operators and found that shift mutation gave the best performance of the two operators. Therefore shift mutation will be the mutation operator used in this study.

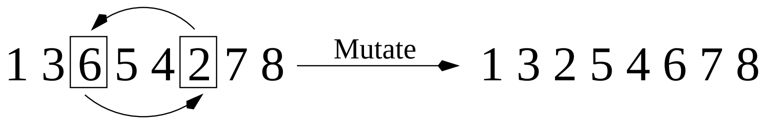


Figure 3–4: Permutation Swap Mutation

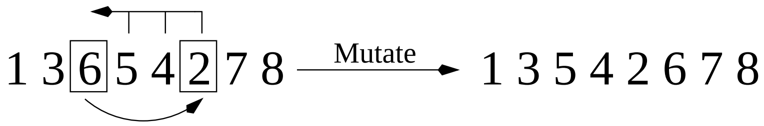


Figure 3–5: Permutation Shift Mutation

3.3 The Counting Ones Problem

The simplest of the problems considered here: for a string of binary digits, the fitness of a given string is the number of ones the string contains. The aim is to obtain a string containing all ones.

This problem has been the subject of quite extensive theoretical study, and the optimal mutation parameter has been derived [Mühlenbein 92] to be $p_m = 1/l$, where l is the length of the binary string. This problem is considered straightforward for a hill-climbing algorithm due to the lack of local optima[1].

The representation used is a string of binary digits. The operators used in this, and the other 3 problems, are binary mutation as described in 1.2.4, and uniform crossover.

Uniform crossover takes two strings and swaps each gene with a fixed probability (usually 0.5) to produce children (Figure 3–6).

---

[1] This does not necessarily mean that the problem is easy for GAs. For large problem sizes, drift may result in no ones being present at some loci throughout the population. Therefore crossover will be unable to make any improvements, and the chances of mutation being able to restore diversity at that locus diminishes with the length of the string.

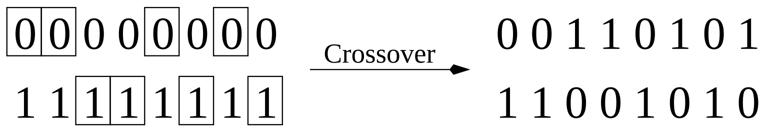


Figure 3–6: Uniform Crossover

The question is: can operator adaption find effective operator probabilities for this simple problem?

3.4 Goldberg‘s Order-3 Deceptive Problem

The concept of *deception* was introduced by Goldberg [Goldberg 87]. This class of problems are constructed so as to have a global optimum and, another, local optimum (the deceptive optimum). A deceptive problem of order l is constructed so that all the building blocks of size less the l lead the search towards the deceptive optimum[2]. According to the building block hypothesis[3], the GA will then combine the fitter building blocks to quickly find the deceptive optimum.

The deceptive problem used as a test problem consists of the concatenation of subproblems of length 3, the overall fitness is the sum of fitnesses of these deceptive subproblems. The bits representing each subproblem are kept close to each other[4] — hence the problem is termed *tight*. The fitness for each 3-bit section of the string is given in Table 3–1. As the problem uses a binary representation, the operators used are as for the counting ones problem.

Both genetic algorithms and hill-climbers will be expected to locate the deceptive optimum — it is a difficult problem for both. However, not all GA-hard

[2] In other words, building blocks of the deceptive optimum have higher fitness than building blocks of the global optimum.

[3] See 1.2.4 for an introduction.

[4] The bits are assigned to each subproblem as follows: 111222333444...nnn.

String	Value	String	Value
000	28	100	14
001	26	101	0
010	22	110	0
011	0	111	30

Table 3–1: The Order-Three Deceptive Problem

problems are deceptive, and not all deceptive problems are GA-hard. Grefensette [Grefenstette 93] provides a critique of deception, and the building block hypothesis.

The rationale behind this problem is to see if adaptation methods are affected by deceptive problems. If the most productive operator is applied more frequently, it could be reasonably argued that we may see the following effects: firstly the genetic algorithm may be attracted to the nearest optimum faster — thus finding the deceptive optimum more quickly, and secondly the chances of stumbling upon the global optimum may be diminished due to the increased speed of convergence.

Therefore it may be the case that a GA with operator adaptation would find the deceptive optima more quickly, with the possibility of reduced solution quality.

3.5 The Royal Road Problem

Work by Forrest and Mitchell [Forrest & Mitchell 93] provides the seminal study of this problem. The motivation for designing this problem was to use the building block hypothesis to devise a class of non-deceptive problems that the genetic algorithm should perform well on.

The Royal Road function used in this study (R1) is computed quite simply: a binary string gets 8 added to its fitness for each of the length-8 building blocks starting at positions 1, 8, 16, etc., it contains. So for a string of length 64 (as used in this study): $R1(111111110...0) = 8$, $R1(0111111110...0) = 0$, and $R1(111...1) = 64$. The operators used for this problem are described in 1.2.4 and 3.3.

According to the building block hypothesis, crossover should be able to combine these length-8 building blocks to quickly make progress towards the optimum; to put it another way, a 'royal road' is laid out for the GA to follow to the optimum. In contrast, a steepest-ascent hill-climber[5] would be ineffective as a large number of single-bit positions would have to be mutated simultaneously for an increase in fitness to be achieved.

However, the results obtained were unexpected. It was found that the GA was outperformed by a stochastic hill-climbing algorithm. The reason for this was suggested to be the presence of 'hitchhikers' - a string with good building blocks will have more children related to itself containing not only copies of the good building blocks, but the poor blocks as well. Therefore, the rise in frequency of good building blocks at locus in the population, could well wipe out good building blocks elsewhere, which are hard to regain.

This is a problem of premature convergence which is particularly pronounced in this problem: the reason that it is included as a test problem. Could operator adaptation, by speeding up the progress of the search, make this problem worse?

3.6 The Long Path Problem

Recent work [Horn *et al.* 94] has presented a class of problems where a genetic algorithm convincingly outperformed a range of hill-climbing algorithms.

The problem was designed to be hard for hill-climbers — not by virtue of the existence of local minima (there are none) — but that the path to the optimum, although always leading upwards for a hill-climber, is extremely long (hence the name).

The particular problem used in this study (Root2path) is designed such that out of the l possible[6] 1-bit mutations of a binary string representing one point on

---

[5] A hill-climber that systematically mutates each bit, and only moves in an uphill direction.

[6] l represents the length of the binary string

the path, only one will result in increased fitness: the other mutations produce points with decreased fitness. The analysis of this problem showed that the path length increases in proportion to $2^{l/2}$ — an ever decreasing fraction of the search space. Points off the path have a fitness given by the 'counting zeros' fitness function, which directs the hill-climber quickly to the start of the path (00...0). The operators used by the GA were described earlier in 1.2.4 and 3.3.

The fitness of a given string is given by the pseudo-code for the function *Hill Position*, below[7]:

```
HillPosition[str] := IF (PathPosition[str])      /* If str is on path */
                     THEN RETURN PathPosition[str]+Length[str];
                          /* Return path position plus problem length */
                     ELSE RETURN Nilness[str];
                              /* Return number of zeros in the string */

PathPosition[str] := CASE
          (str=="0") RETURN 0;                   /* First step on path */
          (str=="1") RETURN 1;                  /* Second step on path */
          (str=="00 rest-of-string")    /* On 1st half of path. Recur */
                  RETURN PathPosition[rest-of-string];
          (str=="11 rest-of-string")    /* On 2nd half of path. Recur */
                  RETURN 3*2^Floor[(Length[str]-1)/2]-2-
                         PathPosition[rest-of-string];
          (str=="101" OR "1011 all-zeros")        /* At halfway point */
                  RETURN 3*2^(Floor[(Length[str]-1)/2]-1)-1;
          OTHERWISE RETURN false;           /* str in not on the path */
```

This problem is selected because it is a case where it is known that the presence of crossover assists search (presumably allowing larger steps to be taken when ascending the path), when compared to hill-climbers which take a long time to find the optimum, due to the long path length. Could this property affect the effectiveness of operator adaptation in any way?

---

[7] Taken directly from [Horn *et al.* 94] - this only works for strings with odd l.

Chapter 4

A Genetic Algorithm For Operator Adaptation

4.1 Overview

This chapter begins by describing the basic genetic algorithm[1] (i.e. without operator adaptation) used in this study. This will provide a benchmark for comparison of the results obtained in this project.

The extensions made to the basic GA to accommodate the operator adaptation techniques under investigation will be described. The adaptation methods implemented are:

- Co-evolution with highly disruptive operators and an external co-evolution crossover probability.
- Co-evolution with highly disruptive operators and an encoded co-evolution crossover probability.
- Co-evolution with operators of low disruption and an external co-evolution crossover probability.
- Co-evolution with operators of low disruption and an encoded co-evolution crossover probability.

---

[1] A description of the representation and operators used for each problem was given in the previous chapter and is not repeated here.

- Co-evolution of operator parameters with fixed operator probabilities.
- Co-evolution of both operator parameters and probabilities.
- Cost Based Operator Rate Adaptation (COBRA).

This chapter only describes the functionality that is used as a part of this study. For a full description of the program and its functionality see Appendix A.

4.2 The Basic Genetic Algorithm

The problem-independent components of the basic genetic algorithm used in this study are now described. The fitness function, representation, and operators used are outlined in the test problem descriptions in the previous chapter.

An unstructured (panmictic) population is used. For the purposes of this study the population size is kept fixed at a size of 100. Two population models were used as part of this study, in order to see if this has any effect upon the effectiveness (or otherwise) of operator adaptation.

The first population model uses *steady-state* reproduction and a *kill-worst* replacement policy, where a child replaces the worst member of the population only if it is fitter.

The second population model uses *generational* replacement with *elitism*. For this model a new population is built up by firstly copying across the best member of the current population, and then generating the remainder of the new population by operating upon the current population. When the new population is generated, it then becomes the current population, and the process is repeated.

In both cases, in order to operate on the population, a parent is selected using rank-based selection with selection pressure 1.5. Thus, the probability of being chosen is dependant upon the *ranked* rather than the absolute fitness: an advantage of this approach is that the scaling performed helps to prevent a 'super-fit' individual from taking over the population.

The operator to use is then selected (each operator has a probability of being selected), and if the operator is crossover, a second parent is selected at random. The operator is then applied to the parent(s) and the child produced.

As stated in 1.2.4, a distinction is made between the probability of an operator being used (operator probabilities), and any parameters associated with the operator (operator parameters). The GA used makes this distinction explicit — operator probabilities are used to decide which operator to use, the operator selected is then informed of the parameters are to be used.

4.3 Co-evolution of Operator Probabilities Using Disruptive Operators

This method, in common with the other co-evolution methods described here, encodes each of the operator probabilities as a real number, with the constraint that the sum of the operator probabilities must be equal to one. An example of such a string is given in Figure 4–1.

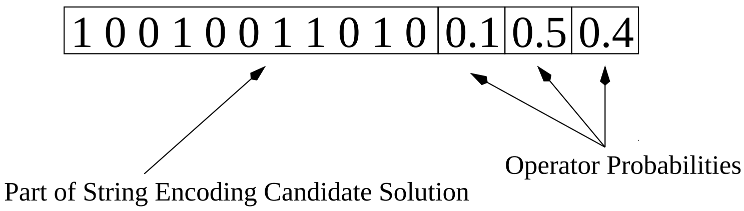


Figure 4–1: Representing Operator Probabilities

These probabilities are themselves operated upon when a child is produced: this gives rise the two variants of this technique. The first variant (call this method 1) possesses an externally set co-evolution crossover rate, which sets the probability of using the co-evolution crossover operator. The second variant (call this method 2) encodes this onto the string also — introducing an additional level of operator adaptation, often termed *meta-learning*.

The operators used in this case are designed to be disruptive: in other words, children tend to be quite different from their parents. This is to see if the choice of co-evolution operators is important for operator adaptation.

The crossover operator used is the real-coded variant of Random Respectful Recombination (R^3) [Radcliffe 91]. For each real coded gene, the value of the child is simply randomly chosen from the interval bounded by the values of the two parents (Figure 4–2).

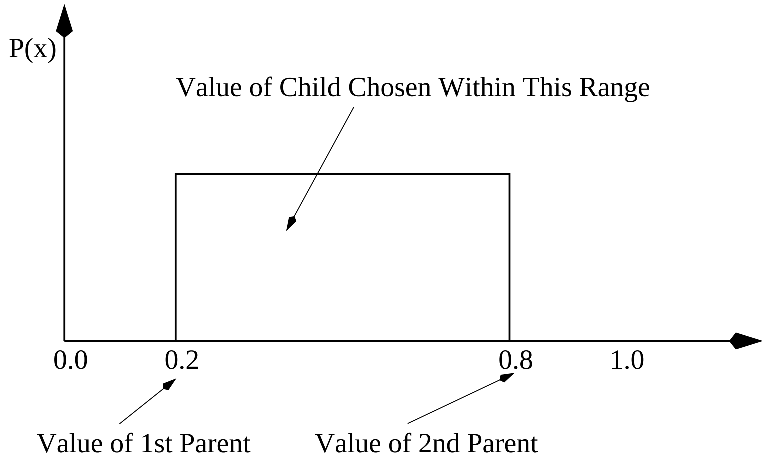


Figure 4–2: The R^3 Operator For Real-Coded Representations

The mutation operator is applied to each real-coded gene with a probability of $1/n$ where n is the number of genes which can be mutated. Mutating a real-coded gene simply involves replacing the present value with a randomly chosen value between 0 and 1.

The operation process is then as follows: a parent is chosen, the encoded operator probabilities determine which operator is used, and the selected operator is applied to the solution section of the string. Then the co-evolution operator is selected, and is used to operate upon the encoded operator probabilities. After this the operator probabilities are renormalised so that they sum to one.

4.4 Co-evolution of Operator Probabilities Using Operators of Low Disruption

This is similar to the co-evolution method above — the difference is the operators that are applied to the encoded operator probabilities. Unlike the operators used above, which have the effect of producing a child that is markedly different from its parent, the operators in this case are designed so that the child produced is quite similar to its parent. Previous work on co-evolving operator probabilities has used operators of this type.

Therefore the crossover operator used is parameterised uniform crossover with parameter 0.1. This means that 90% of the child's genes will be the value of that gene for the first parent, and the remaining 10% of the child's genes are from the second parent.

The mutation operator is simply changing the value of each gene to a value between 0 and 1, given by a Gaussian distribution with mean equal to the value of that gene on the parent and a standard deviation of 0.05. As before, the operator probabilities are renormalised.

Again there are two variants (methods 3 and 4) which mirror the variants described above. A comparison of these 4 adaptation methods will hopefully indicate how sensitive adaptation is to the operators used for co-evolution, and secondly, whether introducing higher-level adaptation (in the form of encoding the co-evolution crossover operator) brings any benefits.

4.5 Co-evolution of Operator Parameters

The effect of adapting the operator parameters is also to be investigated. The operator parameters are encoded to a similar fashion to the operator probabilities, real-coded values between 0 and 1. The meaning of a particular operator parameter is dependent upon the operator it is associated with.

For example, in the case of parameterised uniform crossover, the encoded parameter is taken to be the probability that a given gene in a child is from the second parent. In the case of binary mutation, the parameter is scaled to a bitwise mutation probability of 0 to $5/l$ (where l is the length of the binary string).

As before, two variants were implemented (methods 5 and 6), both using the operators of low disruption because, as mentioned above, this is the type of operator used in previous investigations, and is therefore most likely to produce positive results.

The first variant has fixed operator probabilities, as in the basic genetic algorithm described previously — the difference is that the encoded operator parameters are allowed to co-evolve. The second variant encodes and co-evolves both operator probabilities and parameters in order to see if any interactions between the two can be usefully exploited.

4.6 Cost Based Operator Rate Adaptation

Cost Based Operator Rate Adaptation (COBRA) [Corne *et al.* 94] represents the simplest of the learning-rule type adaptation methods (a review of the study that introduced COBRA is given in 2.5.2).

Unlike the co-evolution methods discussed above, COBRA adapts operator probabilities at the level of the population rather than the individual. Also, the adaptation process is decoupled from the main GA. Comparison of COBRA with co-evolution methods may shed light about which approach is most effective.

A description of how COBRA is implemented is shown below:

Given k operators $o_1, ..., o_k$, let $b_i(t)$ be the *benefit*[2], $c_i(t)$ the *cost*[3], and $p_i(t)$ the probability of a given operator, i at time t. We then apply the following algorithm:

---

[2]The paper used the average increase in fitness when a child was produced that was fitter than its parents.

[3]In other words, the amount of computational effort to evaluate a child.

1. The user decides on a set of fixed probabilities p_i.

2. After G evaluations[4], rank the operators according their values of b_i/c_i, and assign the operators their new probabilities according to their rank[5].

3. Repeat step 2 every G evaluations.

The variables in the adaptation method come from two sources: firstly the gap between operator probability readjustments G, and secondly the ranked operator probabilities provided by the user. The effect that these variables have upon the genetic algorithm will be investigated.

[4] G is the gap between operator probability readjustments.

[5] i.e. the highest probability to the operator with the highest value of b_i/c_i.

Chapter 5

An Investigation Of Adaptation With Co-evolution

5.1 Overview

First, the effect of varying crossover probability on a genetic algorithm with fixed operator probabilities will be investigated. This will be followed by a look at what operators are producing the most improvements to see in how they match the results obtained for a conventional GA.

Next, results obtained pertaining to the effect upon GA performance, and any adaptation that takes place, will be presented and discussed for each of the co-evolution techniques described in Chapter 4.

Two measures of performance are used: the quality of solution obtained after 10000 evaluations, and the number of evaluations after which improvements in solution quality were no longer obtained (or 10000 generations — whichever is smaller)[1]. The rationale is that performance can be affected both ways: for instance it may be possible that solution quality can be exchanged for a still acceptable solution in less time (depending on the application this may be a good tradeoff).

Finally, a discussion of more general points raised by this investigation will be provided.

[1]Usually the genetic algorithm has converged well before 10000 evaluations, therefore further improvements in performance are unlikely

Samples of 50 GA runs were taken in each case, with a population size of 100 and a rank-based selection pressure of 1.5. To compare performance between a GA using an adaptation technique and a conventional GA, a *t-test* is applied between an appropriately tuned conventional GA, and a tuned GA using the adaptation technique. In all cases, a positive value of t indicates that the adaptive GA gives worse performance than the conventional GA.

In order to maintain readability, the results are detailed in full in Appendices B and C. Instead a summary will be provided, and a reference to the location of the detailed information will be given.

5.2 Results For A Standard Genetic Algorithm

In order to provide a basis for comparison, a genetic algorithm with 2 operators[2] was run on the test problems, for both generational[3] and steady-state population models. An exhaustive search was made of the operator probabilities: a genetic algorithm was run for crossover probabilities 0.0 to 1.0 with steps of 0.05.

Observations made on the basis of these results are given in the summaries below.

5.2.1 The $n/m/P/C_{max}$ Problem

For this problem, one of the Taillard [Taillard 93] benchmark problems was used. The problem selected has 20 jobs to be sequenced for a flowline of 20 machines, the reactor residence times were generated with a Taillard benchmark RNG seed of 479340445.

When a steady-state population model was used, no clear trends were observed regarding the effect of crossover upon the quality of solution (best mean makespan

---

[2]For a description of the operators see Chapter 3.

[3]By this I mean a generational population model with elitism — this is for the purposes of readability.

was found for crossover probability 0.05) obtained, except for when crossover is used exclusively, which lead to an increase in makespan. However, a trend was observed for speed of solution: an intermediate crossover probability (0.55) promoted faster search with some reduction in solution quality. The fastest time to solution was observed when crossover was used exclusively, however as mentioned above, this degraded solution quality markedly.

The results obtained for generational population model were poorer, both in terms of solution quality (t=3.81, >99.9% significance between best cases) and time to solution (t=8.22, >99.9% significance between best cases); this may be an effect of the lower selection pressure used. There also appears to be no easily discernible trends, in both solution quality or speed, unlike the steady-state model.

The choice of operator probabilities in this case appeared to only affect the speed to solution. Speed was also affected by the population model used.

One encouraging observation is that the GA routinely managed to outperform the upper bound on solution quality for this problem (2297). The best makespan found during this study was 2243. Previous work by [Reeves 95] obtained similar results.

The results for this problem are given in full in B.2.1 and B.3.1.

5.2.2 The Counting Ones Problem

This problem used a string length of 100. For the steady-state model the optimum was always found, except at high crossover probabilities (0.95 and 1.0) when the optimum was occasionally not reached; presumably because mutation was not applied often enough to combat drift — the resulting loss of diversity prevented the optimum from being reached. The speed to solution, on the other hand, was markedly enhanced by the presence of crossover, correlating well with the crossover probability. For the fastest speed to the *optimum* a crossover probability of 0.8 was found to be the best choice.

As for the $n/m/P/C_{max}$ problem, both the quality of solution (t=1.43, 80%–90% significance — the optimum was not always found for all probabilities) and speed (t=18.04, >99.9% significance) were made worse by the use of a generational model. High crossover probabilities were found to make the GA more effective: the solution quality was found to peak at 0.8 (genetic drift affects solution quality thereafter), the speed of search increased steadily with crossover probability.

For both population models, crossover appears to be the more effective operator overall, but solution quality suffers if mutation is excluded.

The results for this problem are given in full in B.2.2 and B.3.2.

5.2.3 The Order-3 Deceptive Problem

This problem used a string length of 30. The first observation was that, for both population models, there was a clear trade-off between solution quality and speed of search. Low crossover probabilities gave solutions of higher quality which took longer to find. Crossover appeared to achieve greater improvements by finding the deceptive optimum more quickly.

Finally, the speed of search was significantly slower (t=6.42, >99.9%) for the generational model. Solution quality was comparable for both population models.

As suggested earlier, this problem presents a trade-off to the adaptation mechanism – it will be interesting to observe where this trade off will lie.

The results for this problem are given in full in B.2.3 and B.3.3.

5.2.4 The Royal Road Problem

This problem used string length of 64. For a genetic algorithm with a steady state model, the choice is quite clear: use crossover exclusively. The solution quality was maximised when the crossover probability is 1.0, and there appeared to be no clear trends in the number of evaluations required to find these solutions.

The behaviour of a generational GA was strikingly different. The solution quality peaked at a crossover probability of 0.35 and then steadily decreased, with the speed of search following the opposite pattern — slow search equating with high solution quality. This demonstrates how the effect of the operator probabilities is dependent upon the other GA components.

Comparisons with the seminal work [Forrest & Mitchell 93] on this problem are made difficult because it used a different performance criterion: the number of evaluations required to find the optimum[4]. However, on the basis of the steady-state results, crossover is performing most of the search, which is encouraging as the Royal Road problem was designed so that crossover would perform well. Also, the generational results indicate that faster convergence degrades solution quality — which was the reason cited for the failure of the genetic algorithm when compared with a stochastic hill-climber.

Overall, the steady-state GA is able to find results of superior quality (t=3.62, >99.9%), and in less time (t=4.19, >99.9%). The latter may be ascribed to, as before, the higher selection pressure of the steady-state population model.

The results for this problem are given in full in B.2.4 and B.3.4.

5.2.5 The Long Path Problem

This problem used a string length of 29. A high crossover probability when using a steady-state population model was found to be clearly deleterious — the best solution quality was found for a crossover probability of 0.05. Furthermore, no effect of crossover probability upon the speed of search was observed.

When a generational GA is used, solution quality was found to be maximised for a crossover probability of 0.55 (extreme values severely reduced solution quality). Also, increasing the crossover probability steadily increased the speed to final solution. Again, the choice of population model affected the choice of operator probabilities.

In this case, it appears that a generational model may be more useful. First, the quality of solution attainable, at best, was the same for both models (t=1.028, 50–80%), and the speed to solution was indistinguishable (t=0.393, 20–50%). However, the generational model allowed solution quality to be exchanged for a faster search.

---

[4]Conclusions upon the effectiveness of a given method are strongly affected by the performance criterion used.

The results for this problem are given in full in B.2.5 and B.3.5.

5.3 Operator Productivities

Plots of *operator productivities* are now given to see which operator is providing the most improved children at a given stage of a GA run. For the purposes of this investigation the *productivity* of a given operator is the average improvement (if the child is worse than the parent, the improvement is zero) produced over a fixed number of evaluations in the GA's immediate past. For this study, the productivities will be calculated over the previous 100 evaluations.

In all cases below the plots were obtained by running a genetic algorithm for both population models with crossover probability 0.5. The plots obtained are also given in Appendix C.2. It is essential to note that the error bars were not included with the graphs for the sake of clarity — needless to say there is a *large* amount of noise associated with the operator productivities.

5.3.1 The $n/m/P/C_{max}$ Problem

The plots of operator productivity obtained are given below (Figure 5–1).

For both population models, crossover was observed to be consistently the more productive operator of the two. This accounts for the increased speed of search observed for the steady-state model when crossover was used.

The generational model, however, maintained its initially high productivity; this is because, unlike the steady-state model which only accepts children of increased fitness, the generational model accepts whatever children are generated. So the strings of worse quality are available for improvement - as opposed to the steady-state model where it is hard to improve strings of already high fitness.

5.3.2 The Counting Ones Problem

The plots obtained for this problem are given below (Figure 5–2).

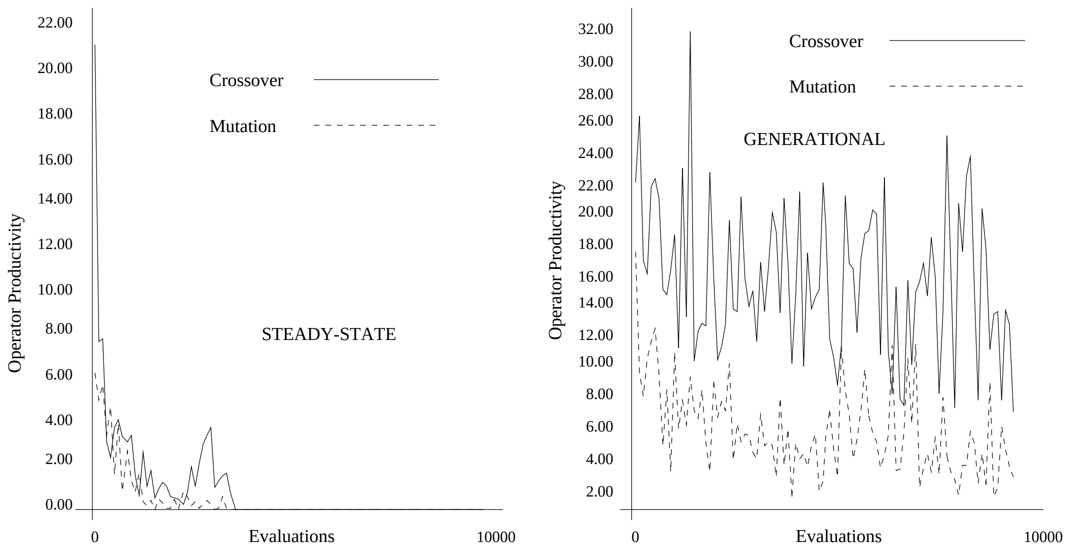


Figure 5–1: Operator Productivities For The $n/m/P/C_{max}$ Problem

For both population models, crossover was observed to be consistently the more productive operator of the two — accounting for the increased speed of search when crossover is used. Also for both models, operator productivity was seen to decline during the course of the GA run. This is simply an effect of the difficulty of making improvements to already very fit strings.

5.3.3 The Order-3 Deceptive Problem

The plots obtained for this problem are shown in Figure 5–3.

As for the previous problems considered here, crossover appears to be the more productive operator. This corresponds well with the observed relationship between increased speed of search and high crossover probability. The associated decrease in solution quality is probably due to the increased speed of search finding the deceptive optimum more often.

5.3.4 The Royal Road Problem

The plots obtained for this problem are given here (Figure 5–4).

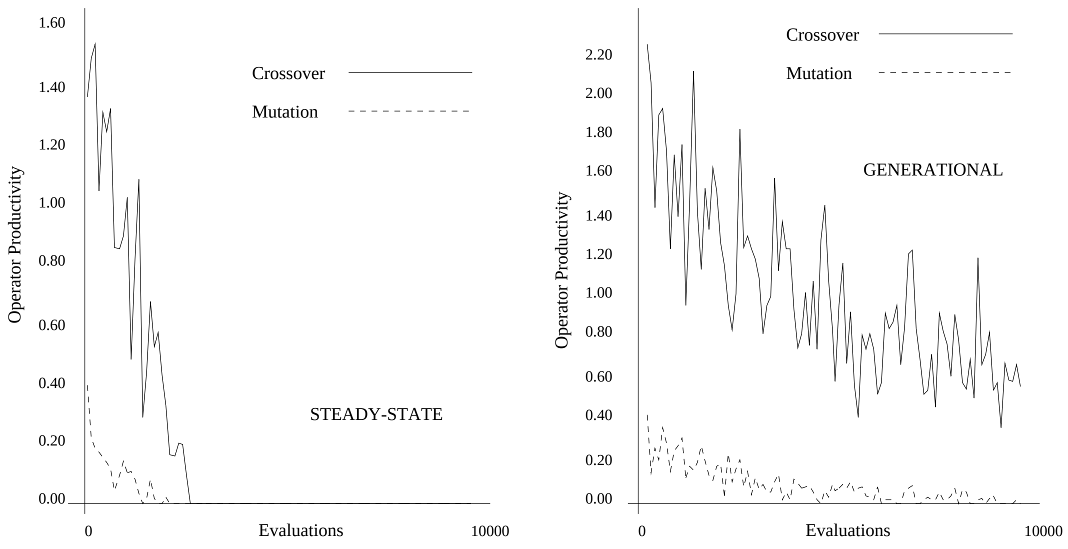


Figure 5–2: Operator Productivities For The Counting Ones Problem

For both population models, crossover was the more productive operator — considering this class of problem was designed as an example of when crossover would be more effective than mutation this comes as little surprise.

The steady-state model appeared to exhibit intermittent improvements: this is due to the fact that 8-bit blocks of ones have to be constructed before an increase is attained. This contrasts with the counting ones problems which has a smooth path to the optimum.

The generational model displayed increasing crossover productivity during the GA run — this is possibly a result of the increased proportion of ones contained in the strings later in the run.

5.3.5 The Long Path Problem

The plots obtained for this problem are displayed in Figure 5–5.

For both population models, the improvements attained were intermittent. Considering that, in the case of mutation, an improvement is attainable by a one-bit mutation on only *one* of the loci on the string, this is quite reasonable to expect.

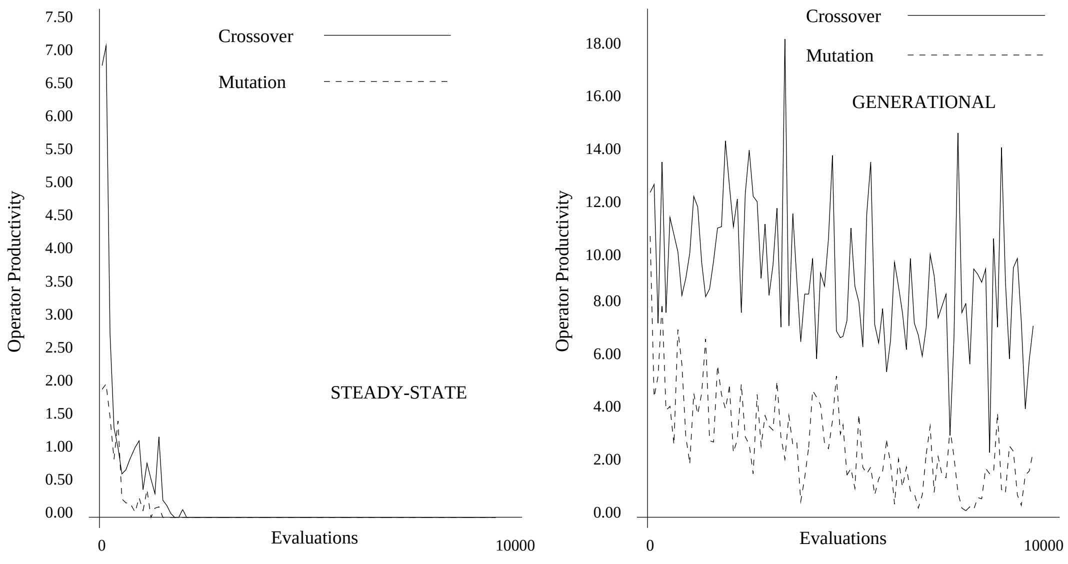


Figure 5–3: Operator Productivities For The Order-3 Deceptive Problem

For the steady-state model, two additional observations can be made: there was a periodicity between improvements attained, and also improvements due to mutation were then quickly followed by improvements due to crossover. This may be due the following process: a child is produced of improved fitness and enters the population, this event will have a periodicity associated with it due to the property of mutation outlined above (the probability of a random one-bit change improving a string on the path is $1/l$, where l is the length of the string). This new information is then spread throughout the population by crossover, which accounts for the crossover improvements following improvements due to mutation. However, this requires further investigation, especially as the role of crossover in this class of problems is poorly understood.

5.4 Co-evolution With Highly Disruptive Operators

The effect that using co-evolution with disruptive operators upon GA performance will be considered for each of the test problems in turn. For co-evolution with an externally set co-evolution crossover probability (method #1), an exhaustive

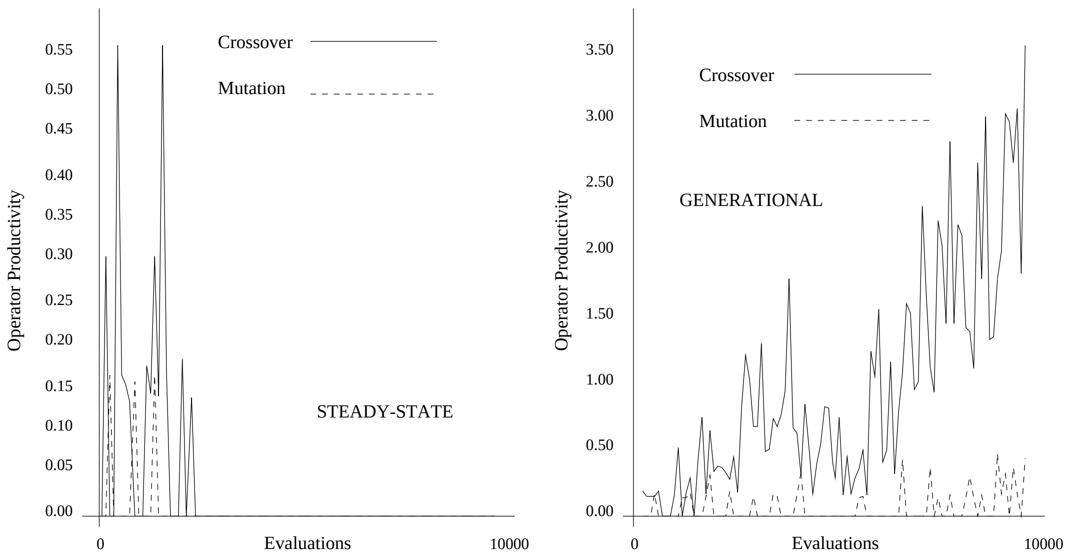


Figure 5–4: Operator Productivities For The Royal Road Problem

search was made: from 0.0 to 1.0 with steps of 0.05. The effect of using meta-learning (method #2) is also examined. Performance will be compared against a conventional GA with an effective crossover probability.

The exhaustive search is made for two reasons. First, to see if there are any interesting trends for the co-evolution crossover probability. Second, it may be the case that no improvement is attained, but the GA is less sensitive to the parameters used by the adaptation method, than to the operator probabilities — which means that the GA will be easier to tune.

Plots of the evolved operator probabilities with time will then be examined to see what, if any, adaptation has taken place. These will be compared with the plots of operator productivity made earlier, and conclusions about the adaptation taking place made.

5.4.1 The $n/m/P/C_{max}$ Problem

For both population models, no trends in either solution quality, or speed to solution with co-evolution crossover probability were observed. Also, there was no significant difference between the makespan or speed to solution obtained between this method and a conventional GA.

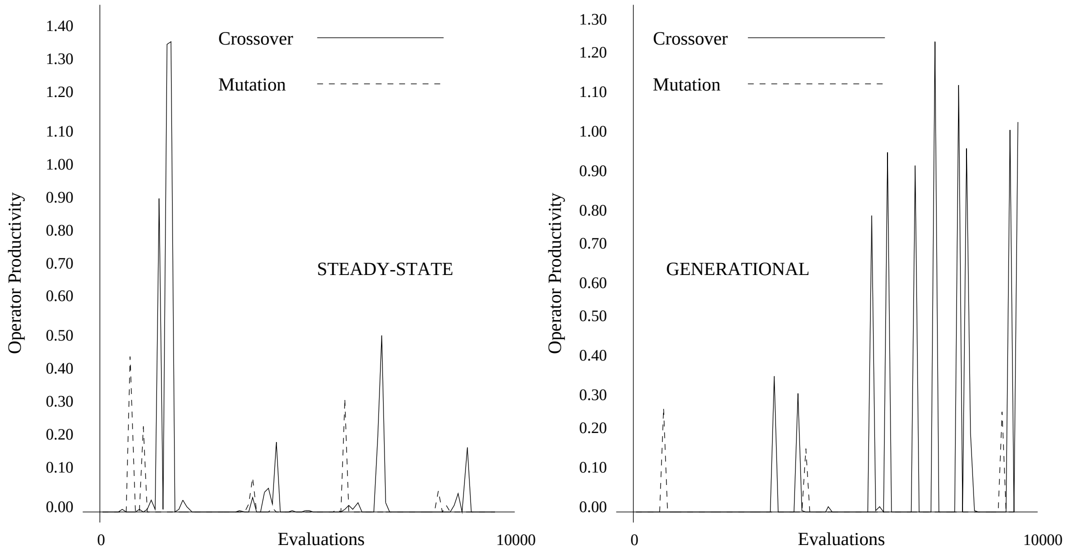


Figure 5–5: Operator Productivities For The Long Path Problem

To examine if any adaptation is taking place, plots of the evolved crossover probability against the number of solution evaluations made were plotted for typical GA runs (Figure 5–6):

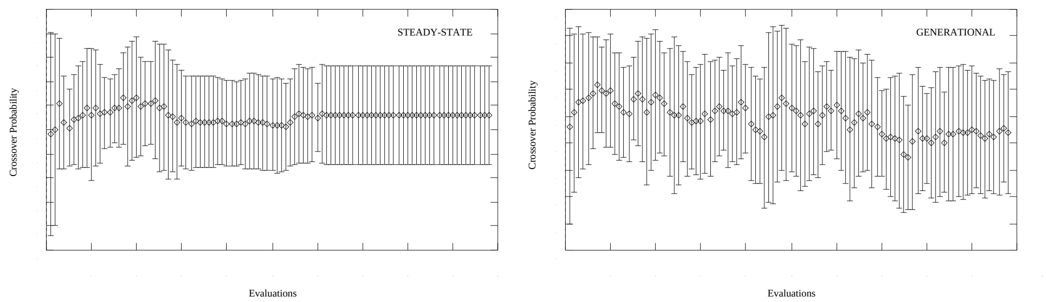


Figure 5–6: Plots For The $n/m/P/C_{max}$ Problem Using Method #1

It appears that the average crossover probability remains at about 0.5. This corresponds well with the trends in solution quality which indicate that an intermediate crossover probability is desirable. However, crossover productivity was observed to be significantly higher. Adaptation towards a higher crossover probability would have expected on these grounds.

When the GA uses meta-learning the results obtained were similar: solution quality and speed to solution were not significantly different to those obtained by the conventional GA, and the evolved crossover probability remained at around 0.5 (Appendix C.3.1).

The results for this problem are given in full in B.2.1 and B.3.1.

5.4.2 The Counting Ones Problem

For the steady-state GA, the optimum was found for all co-evolution crossover probabilities. No trends in the speed to solution were observed, and was found to be significantly worse than the equivalent conventional GA. With meta-learning, the reduction in speed to solution compared to the conventional GA was found to be greater.

The generational GA was found to exhibit a slight reduction in solution quality when using co-evolution, the observed reduction in speed to solution was more noticeable. Again, with meta-learning this reduction in GA performance was found to be even more marked for both solution quality and speed. The t-tests for both population models and co-evolution methods are summarised in Table 5–1.

Performance Measure	Steady-State GA	Generational GA
Solution Quality (#1)	not significant	1.16 (80–90%)
Speed to Solution (#1)	2.62 (>90%)	2.59 (>99%)
Solution Quality (#2)	not significant	3.37 (>99.9%)
Speed to Solution (#2)	3.70 (>99.9%)	4.35 (>99.9%)

Table 5–1: T-test Results For The Counting Ones Problem

Plots of the evolved crossover probability during the course of the GA run (Figure 5–7) indicate that, as for the $n/m/P/C_{max}$ problem, it stays around 0.5. The same behaviour was observed when meta-learning was used (Appendix C.3.2).

Consideration of both the desirable static crossover probabilities (0.8 in both cases), and the fact that crossover is always the most 'productive' operator would indicate that, if adaptation was taking place, the evolved crossover probability would rise towards 0.8. It seems doubtful, therefore, that adaptation is occurring.

The results for this problem are given in full in B.2.2 and B.3.2.

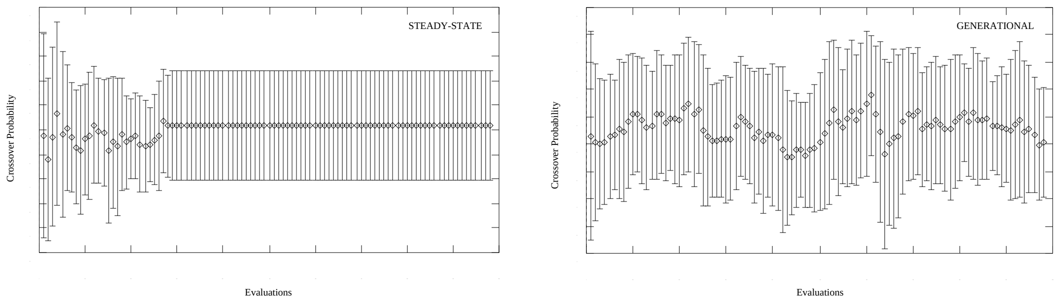


Figure 5–7: Plots For The Counting Ones Problem Using Method #1

5.4.3 The Order-3 Deceptive Problem

For the steady-state GA no clear trends in solution quality or speed of search were observed. Also, the quality of solution obtainable was not quite as high, lying in the range 287.08–288.60 compared with 289.28. The solution quality obtained was more representative of that obtained for crossover probabilities of around 0.5 — the speed to solution reflects this fact as well.

Similar results were observed for the generational model, and when meta-learning was used. As this problem contains a trade-off between solution quality and speed it seems that the GA settles to a point equidistant between the two extremes.

Plots of the evolved crossover probability against the number of solution evaluations made were plotted for typical GA runs. In common with the problems considered above, the evolved crossover probability remained at around 0.5 (Appendix C.3.3).

The results for this problem are given in full in B.2.3 and B.3.3.

5.4.4 The Royal Road Problem

When co-evolution was used with a steady-state GA, no trends in co-evolution crossover probability were found for either performance measure. However, solution quality was found to be much worse (t=5.02, >99.9%) — similar to that obtained with a crossover probability of about 0.6 with a conventional GA. A similar reduction in solution quality was found using meta-learning (t=6.23, >99.9%).

When a generational population model was used, performance was comparable to the corresponding conventional GA, even when meta-learning was used.

In all cases, plots of the evolved crossover probability against the number of solution evaluations show that the evolved crossover probability stays at about 0.5 (Appendix C.3.4). This is despite the operator productivities indicating that crossover is producing most of the improvements, and that a very high crossover probability is desirable for a steady-state GA. The observation that performance is comparable in a generational GA may be due to the fact that an intermediate crossover probability is preferred, rather than any useful adaptation taking place.

The results for this problem are given in full in B.2.4 and B.3.4.

5.4.5 The Long Path Problem

When a steady-state GA is used, solution quality was found to be slightly worse (t=1.30, 80–90%), and speed remained comparable to a conventional GA. When meta-learning was used both solution quality (t=2.13, >95%) and speed (t=1.73, >90%) were reduced significantly.

No such reduction in GA performance was found to occur for a generational GA, even when meta-learning was used. The results obtained were consistent for a GA tuned to maximise fitness.

As for the other test problems, plots of the evolved crossover probability against the number of solution evaluations show that the evolved crossover probability stays at about 0.5 (Appendix C.3.5). The fact that performance is unaffected for a generational GA way be due to an optimal crossover probability of 0.55 (conveniently close) rather than any adaptation taking place.

The results for this problem are given in full in B.2.5 and B.3.5.

5.4.6 Is Adaptation Taking Place?

It appears that this form of co-evolution does not appear to improve GA performance — in fact the opposite is true. Additionally, adaptation does not appear to take place — the evolved crossover probability remains stubbornly around 0.5.

This is even for problems where a high crossover probability is favoured (e.g. counting-ones). In addition, crossover is the more productive operator for most of the problems considered here — adaptation to an increased crossover probability would be reasonable to expect.

In fact, the problems that seem to be least affected by these adaptation methods are those which have intermediate preferred crossover probabilities — a matter of luck rather than adaptation.

The inability of adaptation to take place may be due to two factors:

- The encoded operator probabilities receive information about their effectiveness indirectly — via the fitness information due to the encoded solution. It may be the case that the resulting selection pressure on the crossover probability may be too low for adaptation to take place[5].

- The operators used are too disruptive and destroy any good information that is found by selection — especially if, as argued above, the selection pressure on the operator probabilities is expected to be low.

The first of these factors could indicate that adaptation methods should make information about operator productivities explicit, and separate adaptation from the main GA.

However, the use of operators of low disruption would deal with the second point and, if adaptation was observed to take place, assure us that effective operator probabilities can be evolved.

[5]Especially as the operator productivity information is very noisy.

5.5 Co-evolution With Operators Of Low Disruption

The experiments described above will be repeated using co-evolution operators of low disruption, in order to investigate the effect upon GA performance, and to see if adaptation takes place. Performance will be compared against a conventional GA with an effective crossover probability (Section 5.2).

5.5.1 The $n/m/P/C_{max}$ Problem

No trends were observed in solution quality or speed to solution with co-evolution crossover probability for either model. Also, performance remains comparable to a conventional GA, except for a slight (t=1.80, 80–90%) reduction in speed of search with the steady-state population model.

The use of meta-learning was found to be detrimental. Speed of search was found to be reduced for both steady-state (t=3.11, >99.8%) and generational (t=1.65, >90%) population models.

An example plot of the evolved crossover probability is given in Figure 5–8. Similar plots were obtained when meta-learning was used (Appendix C.3.1).

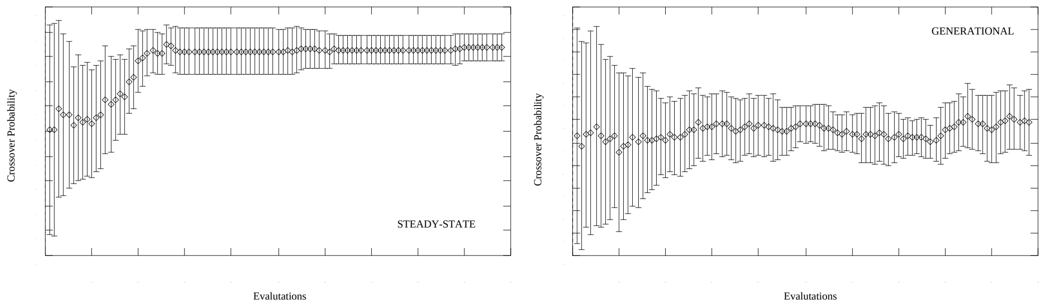


Figure 5–8: Plots For The $n/m/P/C_{max}$ Problem Using Method #3

This indicates that adaptation does take place, for the steady-state model, although the evolved crossover probability often remains around 0.5 which suggests that adaptation is unreliable. However, this lends support to the hypothesis that disruptive operators hinder adaptation.

Adaptation was not observed when the generational model is used. This may well be a result of the lower selection pressure associated with this model.

The results for this problem are given in full in B.2.1 and B.3.1.

5.5.2 The Counting Ones Problem

When a steady-state model was used the optimum was always found for all values of the co-evolution crossover probability except for 0.0 and 1.0. Speed to solution was worse than for a conventional GA. With meta-learning, the optimum was not always located, and speed of search made worse still (Table 5–2).

A generational GA exhibits degradation of both solution quality and time to solution — the use of meta-learning makes the degradation larger.

Performance Measure	Steady-State GA	Generational GA
Solution Quality (#3)	not significant	1.49 (80–90%)
Speed to Solution (#3)	1.92 (>95%)	1.82 (80–90%)
Solution Quality (#4)	not significant	3.65 (>99.9%)
Speed to Solution (#4)	4.21 (>99.9%)	3.18 (>99.9%)

Table 5–2: T-test Results For The Counting Ones Problem

An example plot of evolved crossover probability is given in Figure 5–9. The use of meta-learning gave similar results (see C.3.2 and C.3.2).

Adaptation does appear take place (in the case above to around 0.8), although not for all runs (occasional runs exhibit little adaptation); indicating that adaptation is unreliable. As before, adaptation is not observed when the generational model is used — a possible result of the lower selection pressure associated with this model.

One point that needs to be addressed is that performance is degraded significantly even when adaptation to a high crossover probability takes place. Examination of Figure 5–9 shows that it takes some time for a high crossover probability

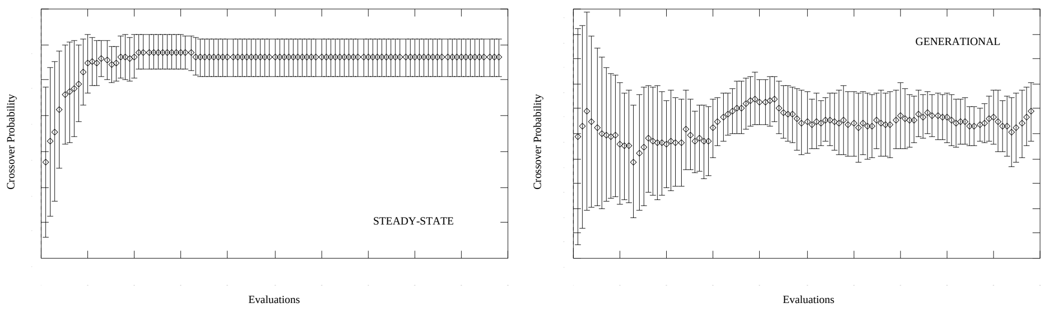


Figure 5–9: Plots For The Counting Ones Problem Using Method #3

to be reached (about 2000 evaluations), by then most of the useful search has been performed and the GA is getting the last few bits right. It would appear that good operator probabilities are more important early on in a GA run than later.

The results for this problem are given in full in B.2.2 and B.3.2.

5.5.3 The Order-3 Deceptive Problem

With a steady-state GA no clear trends in solution quality or speed of search were observed. The quality of solution obtained was found to lie in the range 287.64–289.16; comparable with a conventional GA tuned towards solution quality. The speed to solution was that expected for such tuning. Similar behaviour occurs when a generational model was used. No trends in performance with co-evolution crossover probability were observed for either population model.

Examination of example plots of evolved crossover probability (Figure 5–10) show that adaptation was not occurring — even though crossover was the more productive operator. In fact, when a number of such runs are examined some do evolve towards high crossover probability, but an equal number evolve towards a low probability. It seems likely therefore that these are due to 'neutral evolution' (random fluctuations being propagated in the absence of selection pressure).

Meta-learning was observed to have no effect upon the lack of ability to adapt. Example plots are given in full in C.3.3.

The results for this problem are given in full in B.2.3 and B.3.3.

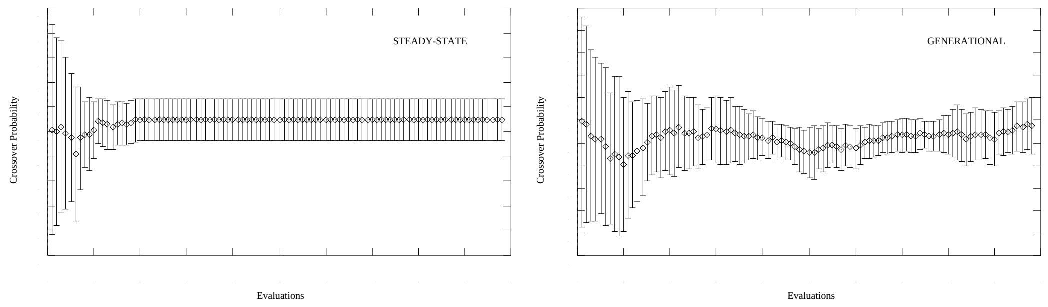


Figure 5–10: Plots For The Order-3 Deceptive Problem Using Method #3

5.5.4 The Royal Road Problem

Solution quality when using this form of co-evolution was degraded for a steady-state GA. Meta-learning appeared to make the situation worse: in terms of both quality and speed. The performance degradation was much reduced with the use of a generational model — only speed to solution was impaired.

Performance Measure	Steady-State GA	Generational GA
Solution Quality (#3)	5.11 (>99.9%)	not significant
Speed to Solution (#3)	not significant	1.33 (80–90%)
Solution Quality (#4)	5.64 (>99.9%)	not significant
Speed to Solution (#4)	1.62 (80–90%)	1.52 (80–90%)

Table 5–3: T-test Results For The Royal Road Problem

Examination of example plots of evolved crossover probability (See Appendix C.3.3) again indicate that adaptation was not occurring — even though crossover was the more productive operator, and that a high crossover probability is preferred when a steady-state population model was used.

The results for this problem are given in full in B.2.4 and B.3.4.

5.5.5 The Long Path Problem

A steady-state GA using this adaptation technique was observed to exhibit poorer performance than a conventional GA. Meta-learning was found to affect performance even more, in terms of both solution quality and speed. The effect upon a generational GA was less: only with meta-learning was a detrimental effect upon search speed observed.

Performance Measure	Steady-State GA	Generational GA
Solution Quality (#3)	2.42 (>99%)	not significant
Speed to Solution (#3)	not significant	not significant
Solution Quality (#4)	4.88 (>99.9%)	not significant
Speed to Solution (#4)	2.03 (>95%)	1.69 (>90%)

Table 5–4: T-test Results For The Long Path Problem

Examination of plots of evolved crossover probability (Appendix C.3.3) indicate that no adaptation was taking place when a steady-state model was used.

There appeared to be adaptation to a higher crossover probability late in the GA run when a generational population model was used. This did not seem to lead to improved GA performance though. The late increase in crossover probability could be attributed to the greater productivity of crossover late in the run as shown in Figure 5–5. But if this was the case, a similar result would have been observed for the Royal Road problem which exhibits a similar trend in crossover productivity when a generational GA is used.

The results for this problem are given in full in B.2.5 and B.3.5.

5.5.6 Should Operator Parameters Be Adapted Instead?

The results above demonstrate that use of operators of low disruption is required for adaptation to take place. Even then, the resulting adaptation appears to be unreliable, being dependent upon the problem and population model used. Also meta-learning often makes performance worse still, although it is not clear why as, in most cases, no adaptation is taking place anyway.

An interpretation of these results is that the selection pressure upon the encoded operator probabilities is weak — a serious difficultly when it is noted that all the test problems show that crossover is the more productive operator (often by quite a large margin). Many problems would not show this clear difference in operator probability; it would seem unlikely that this approach would be able to exploit any subtle trends in operator productivity that may be present (which is really the point of the exercise!).

This leads to the suggestion that, for reliable adaptation of operator *probabilities*, the adaptation mechanism has to be decoupled from the main GA, and the operator productivity information made explicit. This supports the use of a 'learning rule' type adaptation method such as COBRA.

However, much of the previous work on co-evolution has adapted operator *parameters* such as the bitwise mutation probability. It could be the case that if these operator parameters were encoded, the selection pressure upon them would be greater than for the operator probabilities. If this was so, useful adaptation may take place. This will be the next topic of investigation.

5.6 Adaptation Of Operator Parameters

This investigation does not apply to the $n/m/P/C_{max}$ problem as the operators used have no parameters to be adapted. Both of the variants described in Section 4.5 (co-evolution methods 5 and 6) will be investigated. The first of the variants (fixed operator probabilities, co-evolved operator parameters) should indicate whether any useful adaptation of operator parameters is taking place, and the effect upon performance. The other variant (co-evolved operator parameters and probabilities) will examine if any useful interactions between operator parameters and probabilities can be exploited.

When co-evolving operator parameters only, the static crossover probability was selected on the basis of its effectiveness for a conventional GA. The crossover probabilities used are given in Table 5–5. The results obtained by this method will be compared against a conventional GA with the same crossover probability.

Plots of the mutation parameter (i.e. the bitwise mutation probability) will also be examined to see if adaptation takes place. Also, in the cases of the Counting Ones and Deceptive problems, theoretical work [Mühlenbein 92], has derived optimal values: $1l$ and $3/l$ respectively (where l is the length of the binary string).

Problem	Generational GA	Steady-State GA
Counting Ones	0.80	0.80
Order-3 Deceptive	0.05	0.20
Royal Road	0.35	0.95
Long Path	0.55	0.20

Table 5–5: Crossover Probabilities used for Co-evolution Method #5

As before, a exhaustive search of the externally set co-evolution crossover probability was conducted (from 0.0 to 1.0 with steps of 0.05) in order to analyse the sensitivity of GA performance.

5.6.1 The Counting Ones Problem

Adapting operator parameters only was found to adversely affect performance (Table 5–6): the steady-state GA experienced a degradation in speed to solution; whereas the generational GA delivers solutions of worse quality while taking longer to do so than a conventional GA. In either case, no trends in performance with the co-evolution crossover probability were observed.

Performance Measure	Steady-State GA	Generational GA
Solution Quality (#5)	not significant	4.59 (>99.9%)
Speed to Solution (#5)	3.12 (>99.9%)	3.79 (>99.9%)
Solution Quality (#6)	1.47 (80–90%)	4.72 (>99.9%)
Speed to Solution (#6)	4.83 (>99.9%)	4.89 (>99.9%)

Table 5–6: T-test Results For The Counting Ones Problem

To investigate if any useful adaptation took place, a plot of the mutation parameter against evaluations was made (Figure 5–11). For the generational GA, adaptation was observed on most of the runs examined — the bitwise mutation probability was found to adapt to a value comparable to $1/l$ — the theoretical optimum. However, no such adaptation was observed for the steady-state model; the plots obtained looked appeared to be drifting aimlessly. It appears strange that the generational GA produced greater deterioration in performance, although it was the model for which successful adaptation occurred. More work is required to find out why.

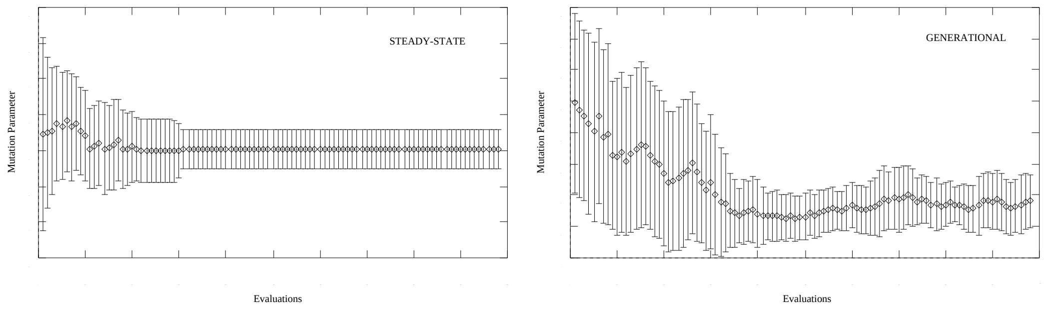


Figure 5–11: Plots of Mutation Parameter For The Counting Ones Problem

As expected from the results obtained earlier, co-evolution of both operator parameters and probability lead to a marked deterioration in performance (Table 5–6).

The results for this problem are given in full in B.2.2 and B.3.2.

5.6.2 The Order-3 Deceptive Problem

When co-evolution was applied to only the operator parameters, performance was unaffected, apart from a significant *improvement* in search speed (Table 5–7) for the steady-state GA. Co-evolution of both operator parameters and probabilities lead to a movement of the trade-off between speed and quality towards greater speed as observed earlier.

Performance Measure	Steady-State GA	Generational GA
Solution Quality (#5)	not significant	not significant
Speed to Solution (#5)	-2.01 (>95%)	not significant

Table 5–7: T-test Results For The Order-3 Deceptive Problem

Plots of the evolved mutation parameter against evaluations indicate that, for both population models, the mutation parameter remained at about $3/l$ — the theoretical optimum — which explains why increased performance was observed for the steady-state GA. However, as this value lies in the middle of the range it is unclear whether this is due to adaptation — the same observation is possible in the absence of selection pressure.

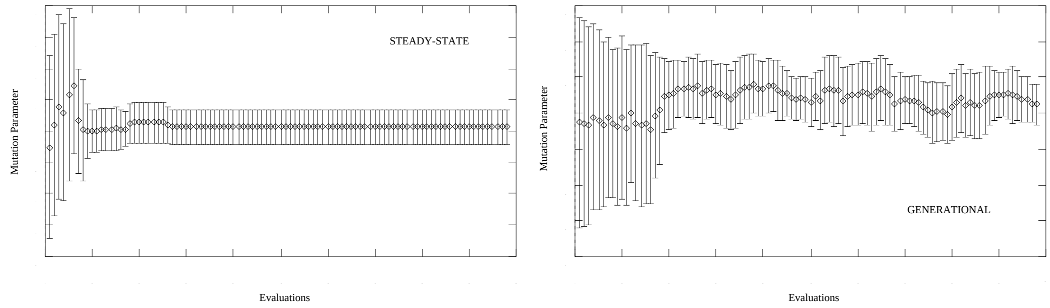


Figure 5–12: Plots of Mutation Parameter For The Order-3 Deceptive Problem

The results for this problem are given in full in B.2.3 and B.3.3.

5.6.3 The Royal Road Problem

When performance was compared to a conventional GA (Table 5–8) it was found that, in most cases, solution quality was degraded in favour of increased speed. The only exception was in the case of a steady-state GA when both operator probabilities and parameters were adapted — both solution quality and speed were adversely affected, due to the inability to adapt to the high crossover probability required for this problem.

Performance Measure	Steady-State GA	Generational GA
Solution Quality (#5)	6.83 (>99.9%)	3.38 (>99.9%)
Speed to Solution (#5)	-4.43 (>99.9%)	-2.87 (>99.5%)
Solution Quality (#6)	5.89 (>99.9%)	3.93 (>99.9%)
Speed to Solution (#6)	2.49 (>98%)	-2.55 (>98%)

Table 5–8: T-test Results For The Royal Road Problem

Plots of the evolved mutation parameter against evaluations (Figure 5–13) indicate that, for both population models, the mutation parameter remained at about $3/l$. It appears that a higher value of the mutation parameter increases search speed at the expense of quality.

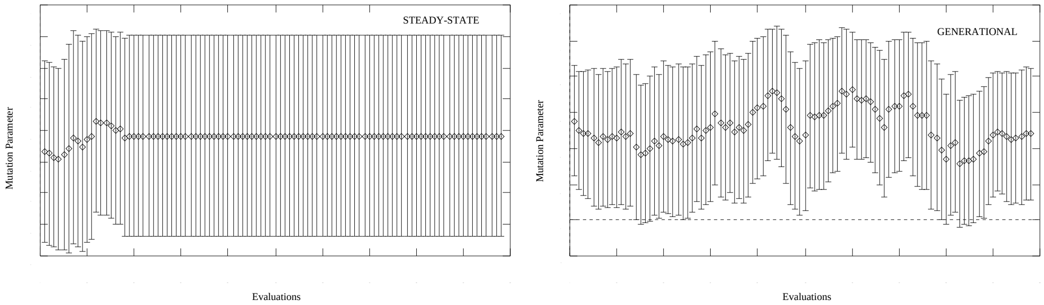


Figure 5–13: Plots of Mutation Parameter For The Royal Road Problem

It remains unclear whether adaptation actually occurs, because the value obtained lies in the middle of the range — this is also possible in the absence of selection pressure; a study of the effect of the mutation parameter would help to resolve this question. Also, it could be argued that in view of the stepwise fitness function, information on mutation parameter performance would be intermittent — which may make adaptation difficult.

The results for this problem are given in full in B.2.4 and B.3.4.

5.6.4 The Long Path Problem

For both population models, only the speed of solution was adversely affected (Table 5–9). The co-evolution of both operator probabilities and parameters displayed a greater effect upon performance.

Performance Measure	Steady-State GA	Generational GA
Solution Quality (#5)	2.06 (>95%)	3.38 (>99.9%)
Speed to Solution (#5)	not significant	not significant
Solution Quality (#6)	2.57 (>98%)	3.93 (>99.9%)
Speed to Solution (#6)	not significant	not significant

Table 5–9: T-test Results For The Long Path Problem

Plots of the evolved mutation parameter against evaluations (Figure 5–14) indicate that, for a steady-state GA, the mutation parameter remained at about $3/l$; whereas adaptation to a value of $1/l$ was commonly observed when a generational model was used.

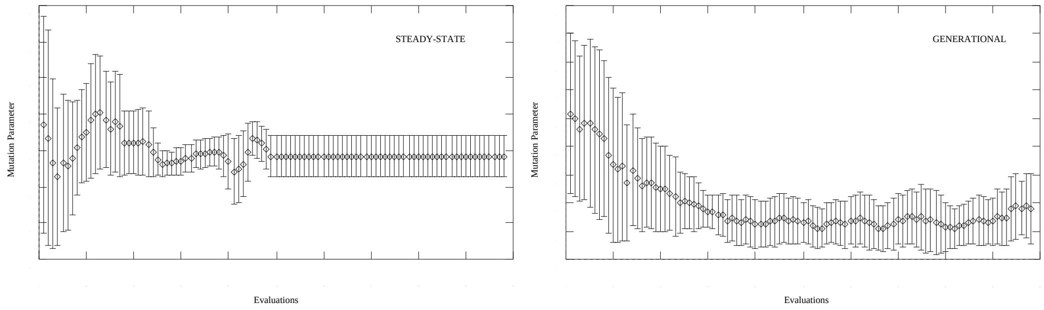


Figure 5–14: Plots of Mutation Parameter For The Long Path Problem

Further work is required to ascertain what the 'optimal' mutation parameter is; if only to confirm whether $3/l$ happens to be the preferred value for a steady-state GA, or that adaptation is failing to take place.

The results for this problem are given in full in B.2.5 and B.3.5.

5.6.5 Adaptation Does Not Necessarily Equate With Improved Performance

The effect on performance observed indicates that genetic algorithm performance is quite sensitive to the operator parameters. The attempts at adaptation for these problems appear to be more successful, although adaptation is still somewhat unreliable and problem-dependent.

However, it is clear that even when adaptation takes place, performance is not necessarily improved (more likely the opposite). Therefore the implicit assumption that if the genetic algorithm can evolve operator probabilities (or parameters) then it must be a good thing should be questioned — it appears to be more effective to give the genetic algorithm this information rather than have it work it out for itself.

Chapter 6

An Investigation Of Adaptation Using COBRA

6.1 Introduction

An alternative to the co-evolutionary approach examined in the previous chapter is to employ some sort of 'learning-rule' method which adapts operator probabilities separately from the main genetic algorithm.

The disappointing results obtained for co-evolution based methods would lend support for this decoupling of the adaptation mechanism (which hopefully would make the adaptation mechanism less sensitive to the other GA components), and the *explicit* measurement of operator productivity information (thus ensuring that the adaptation mechanism gets the required information). The learning-rule method that will be examined is also the simplest: COBRA, which was described in full in 4.6.

To use COBRA, three decisions have to be made: first, a measure of benefit has to be decided upon; second, the initial operator probabilities have to be assigned; and third, a suitable gap G between re-ranking probabilities has to be decided upon.

The measure of benefit in this study will simply be the operator productivity measured over the last G evaluations. Instead, investigation will instead turn to the effect that the initial operator probability and G have upon performance.

6.2 The Investigation

This section will detail and comment upon the results obtained by the investigation of this adaptation technique. These experiments seek to answer some outstanding questions left by the initial study of this technique [Corne *et al.* 94].

In that study, the operator probabilities used were those suitable for the conventional GA. However, this may not necessarily be the best choice for COBRA. Also, is the GA made more, or less sensitive to the operator probabilities provided than if static probabilities are used? For each of the problems, the performance of crossover probabilities from 0.05 to 0.95 (in steps of 0.05 with the exception of 0.5) will be examined.

The value of G used has to balance the need for rankings to reflect the current state of the GA (requiring a small G), and for a large enough sample of operator performance to be made so to prevent the adaptation mechanism from being affected by noise in the information that it receives. The previous study made no attempt to investigate the effect of varying G, this question will now be investigated by examining a range of values of G (from 200 to 2000 evaluations in steps of 200).

Samples of 50 GA runs were taken in each case, with a population size of 100 and a rank-based selection pressure of 1.5. To compare performance between a GA using an adaptation technique and a conventional GA, a *t-test* is applied between an appropriately tuned conventional GA, and a tuned GA using the adaptation technique. In all cases, a positive value of t indicates that the adaptive GA gives worse performance than the conventional GA.

6.2.1 The $n/m/P/C_{max}$ Problem

First, the effect that G has upon GA performance was investigated. For a steady-state GA, an initial crossover probability of 0.35 was used — no significant effect upon performance was found, nor any noticeable trends with respect to G observed. Again, no impact upon performance was observed for the generational GA (an initial crossover probability of 0.65 was used) — any differences between differing values of G were insignificant.

An examination of the effect of the initial crossover probability for a steady-state GA showed that the best solution quality and speed were obtained for intermediate crossover probabilities (0.45 and 0.55) — the speed of search was found to be the more sensitive measure. No trends in solution quality were observed with a generational model, but speed to solution was found to be highest for intermediate probabilities (around 0.4).

Comparison of the *sensitivity* of GA performance to crossover probability with a conventional GA showed that a GA using COBRA was of equal sensitivity for a generational model and of lower sensitivity when a steady-state model was used - the effect of a poor choice of initial probability appears to be mitigated somewhat (Table 6–1).

Performance Measure	Steady-State GA	Generational GA
Solution Quality (Static)	2386.56-2420.38	2441.16-2452.62
Solution Quality (COBRA)	2395.68-2410.92	2437.76-2454.08
Speed to Solution (Static)	5064.36-7716.84	7714.00-8914.00
Speed to Solution (COBRA)	5049.50-6641.06	7824.00-8914.00

Table 6–1: Range of GA Performance For The $n/m/P/C_{max}$ Problem

The results obtained are given in full in Appendices B.2.1 and B.3.1.

6.2.2 The Counting Ones Problem

No significant effect upon performance was observed with varying G for either of the population models considered here. The choice of initial operator probability was found to be the factor that determined the performance of the genetic algorithm.

For a steady-state GA, the optimum was always found for most values of the initial crossover probability — values at the extremes were found to exhibit slight deterioration in solution quality. Speed to solution was found to increase with crossover probability, reflecting the behaviour of the conventional GA.

Similar trends in solution quality were observed for a generational GA — values at the extremes were found to exhibit some deterioration in solution quality. Speed to solution was optimised for an initial crossover probability of 0.8. Search speed was observed to degrade the further away the initial crossover probability was set from this value.

Table 6–2 shows how GA performance is affected by the crossover probability. A steady-state GA showed slightly increased sensitivity to initial crossover probability when COBRA was used. With a generational GA, the use of COBRA resulted in a reduction of the sensitivity of the GA to the crossover probability used.

Performance Measure	Steady-State GA	Generational GA
Solution Quality (Static)	99.90-100.00	93.38-99.96
Solution Quality (COBRA)	99.84-100.00	98.36-99.96
Speed to Solution (Static)	2172.42-5645.96	7714.00-9228.00
Speed to Solution (COBRA)	2172.42-6525.94	7714.00-8486.00

Table 6–2: Range of GA Performance For The Counting Ones Problem

The results obtained are given in full in Appendices B.2.2 and B.3.2.

6.2.3 The Order-3 Deceptive Problem

When COBRA was used on a steady-state GA, solution quality increased with the gap size, G, but solution quality was still worse than for a conventional GA with equal crossover probability (0.2). However this reduction in solution quality was countered by an increase in speed — indicating that COBRA may be useful for increasing search speed, although at the possible expense of solution quality. Similar trends were found with a generational model when a crossover probability of 0.05 was used.

The trends in performance with initial crossover probability were similar for both population models: intermediate crossover probabilities provided solutions of comparatively high quality (but still less than obtainable by a conventional GA), but at the expense of speed - as the crossover probability was adjusted to more extreme values, the solution quality was degraded, with a corresponding speed increase.

COBRA also had an effect upon the sensitivity of the GA to the crossover probability (Table 6–3). In all cases but search speed with a generational population model, the performance of a GA using COBRA was found to be less sensitive to the crossover probability than a conventional GA.

Performance Measure	Steady-State GA	Generational GA
Solution Quality (Static)	286.28-289.28	284.20-289.68
Solution Quality (COBRA)	286.28-288.48	284.92-288.04
Speed to Solution (Static)	2192.46-4209.68	4484.00-6212.00
Speed to Solution (COBRA)	1978.92-3408.86	4244.00-6412.00

Table 6–3: Range of GA Performance For The Order-3 Deceptive Problem

The observed trade-off between speed and quality is due to COBRA assigning the higher probability to the more productive operator (crossover) at the earliest opportunity - as observed for a conventional GA, crossover then quickly finds the deceptive optimum.

The results obtained are given in full in Appendices B.2.3 and B.3.3.

6.2.4 The Royal Road Problem

The performance of a steady-state GA using COBRA was unaffected by the gap size, G, compared to a conventional GA with the same crossover probability (0.95) — both quality and speed of search were comparable.

When a generational population model was used, no effect of G upon speed was observed, but solution quality was found to improve with increasing G, until at G=2000 a solution quality comparable to that of a conventional GA with the same crossover probability (0.35) was obtained. The reason for this may lie in the fact that crossover is the more productive operator, so the high probability (0.65) will be assigned to crossover at the earliest opportunity. The results for a static GA indicate that this is not the most effective value. On the basis of this, operator

Performance Measure	Steady-State GA	Generational GA
Solution Quality (Static)	25.76-40.64	22.72-35.52
Solution Quality (COBRA)	26.88-40.32	25.28-35.84
Speed to Solution (Static)	2875.60-3951.50	5626.00-7804.00
Speed to Solution (COBRA)	2606.74-4012.26	5596.00-7620.00

Table 6–4: Range of GA Performance For The Royal Road Problem

productivity is not necessarily the only factor that needs to be taken into account when assigning operator probabilities.

The effect of initial crossover probability upon performance for a steady-state GA using COBRA was a steady increase in solution quality, matched with a corresponding decrease in speed, with increasing crossover probability. COBRA appears to have set up a trade-off that was not apparent in the conventional GA. A generational GA using COBRA also exhibited a tradeoff: solution quality was maximised using intermediate crossover probabilities, but at the expense of speed of solution which was maximised at extreme probabilities.

The sensitivity of performance to the crossover probability when using COBRA was reduced in most cases, with similar maximum solution quality, but increased speed to solution. COBRA seems to be a technique to increase speed of search, rather than to increase solution quality, which it appears ready to sacrifice.

The results obtained are given in full in Appendices B.2.4 and B.3.4.

6.2.5 The Long Path Problem

When a steady-state model was used, COBRA lead to a deterioration in solution quality (t=1.32, 80–90% best case) when compared to a conventional GA with the same crossover probability (0.2). The solution quality was found to decline further with increased gap size G. No trends in speed to solution with G were observed, which remained comparable to that obtained by the conventional GA.

The behaviour exhibited by a generational GA using COBRA was markedly different: solution quality rose with increased G (presumably the larger samples used prevent spurious re-ranking due to noise in the operator productivity information) and performance was similar to a conventional GA (with crossover probability 0.55). Speed to solution was comparable to the conventional GA and was unaffected by G.

No trends in performance with initial crossover probability were observed for a steady-state GA using COBRA — the only effect of using COBRA was a reduction in solution quality (t=2.65, >99%). Using COBRA with a generational GA gave comparable performance, although a trade-off was set up between quality and speed - quality was maximised with a intermediate crossover probability, speed was maximised with an extreme crossover probability.

The sensitivity of COBRA to initial crossover probability is summarised in Table 6–5. The poor performance shown by COBRA for a steady-state GA is quite evident, thus the lower sensitivity of solution quality is a moot point. The generational GA showed a marked reduction in sensitivity in terms of solution quality and speed of search.

Performance Measure	Steady-State GA	Generational GA
Solution Quality (Static)	28287.40-48024.94	35823.60-48686.72
Solution Quality (COBRA)	40955.46-44722.96	41407.06-49177.08
Speed to Solution (Static)	5129.38-6735.60	3976.00-7848.00
Speed to Solution (COBRA)	5575.30-7437.26	4286.00-5702.00

Table 6–5: Range of GA Performance For The Long Path Problem

The question of the poor performance of COBRA with a steady-state model begs an answer. It may be due to the operator productivity information, which in this case indicates that crossover should be the dominant operator, is at odds with the fact that, for a static GA, the preferred crossover probability is 0.2. This explanation accounts for the observation that the degradation in solution quality was less for small G — the intermittent nature of the productivity information meant that peaks in mutation productivity were not averaged out and switches to the lower, preferred crossover probability did occur.

The results obtained are given in full in Appendices B.2.5 and B.3.5.

6.3 Discussion

In practical terms, COBRA appears to be the more promising of the two approaches for adapting operator probabilities, although no actual improvements in performance over what could be obtained by a suitably tuned conventional genetic algorithm were reported.

An advantage of this approach is the observed reduction in sensitivity to the provided operator probabilities which is useful in two respects: first, the genetic algorithm may require less tuning to attain acceptable performance; second, some applications of genetic algorithms will have to deal with different problems of the same type with no time for re-tuning (e.g. a dynamic scheduling system). In this case the GA will be more robust towards unexpected situations.

Performance appears affected less by the gap between re-ranking G, than the initial operator probabilities. The trade-off between quality and speed observed for some problems was also found to be controllable by adjusting the initial operator probabilities, which may be useful in some situations. Also, it does not necessarily follow that the most suitable operator probabilities for COBRA are those that were found to be suitable for a conventional GA.

When should COBRA *not* be used? The answer appears to be when the operator productivities suggest that a high probability for a operator is desirable, although the preferred operator probability is low for a conventional GA. It is in these cases that adaptation on the basis of operator probabilities is misleading.

From this observation it would appear that operator productivity is not the sole criterion upon which assignments of operator probabilities should be made — other aspects such as the role mutation plays in maintaining diversity in the GA population are also important.

Chapter 7

Conclusion

7.1 Overview

This dissertation began with a discussion of the potential utility of adapting operator probabilities during a genetic algorithm run. Support for this was gained from the realisation that finding good operator probabilities for a conventional GA is itself very hard, and work that showed that varying operator probabilities during the course of a run increased performance.

Early on, the distinction was made between operator probabilities (the probability that a given operator is used) and operator parameters (any parameters associated with the operator, e.g. the bitwise mutation probability). This distinction is often blurred in most work in genetic algorithms.

Two classes of operator adaptation methods were identified: co-evolution methods which encode the operator probabilities onto each member of the GA population so that probabilities that lead to children of improved fitness will be selected for, and external learning-rule methods that adjust operator probabilities explicitly on the basis of some measure of benefit for each operator.

To study the effect that these methods have upon GA performance, five test problems were selected; each with differing properties. The operator adaptation methods studied were then described in detail and investigated — the conclusions drawn are discussed in the following sections.

7.2 Tuning A GA IS Hard

Before any study of operator adaptation can be performed, a comparative benchmark has to be set up. To this end, the effect of operator probabilities upon GA performance for each of the problems was examined. Two population models were also considered to study the impact of other GA components upon the operator probabilities to be used.

From this investigation, the following points can be made:

- The operator probabilities used do have a significant effect upon GA performance, and appropriate values vary from problem to problem.

- The population model used has a big impact both upon performance and upon the behaviour of the GA with different operator probabilities.

- There is often a trade-off between different performance measures — gains in one are often offset by losses in another. This means that the 'right' set of operator probabilities depends on what is trying to be achieved.

Therefore the task of finding suitable operator probabilities is quite hard — it is very dependent upon the features of the problem at hand, the other components of the GA, and the aims of the user.

This investigation was then extended by a study of the operator productivities (the ability of a given operator to produce children of increased fitness). For all of the problems investigated crossover was consistently the more productive operator; but comparison with the static GA shows that for some problems, a low crossover rate is preferred as is gives the best solution quality. This seems to indicate that operator productivity is the only criterion with which to assign operator probabilities to a conventional GA.

7.3 Co-evolutionary Methods

The following questions were investigated: Does adaptation take place? What is the effect upon performance? What operators should be used on the encoded operator probabilities? If an externally set crossover probability for the encoded probability is used, are there any trends? Should this probability be encoded as well, introducing a form of meta-learning? Finally, should operator parameters be co-evolved?

The choice of co-evolution operators was found to have a dramatic effect: disruptive operators were found to remove the ability to adapt as they destroy any information gained by selection. The use of operators of low disruption improved matters somewhat, but the occurrence of adaptation was found to be problem dependent and unreliable. The use of meta-learning was found to make no difference. This is surprising when you consider that crossover was consistently the more productive operator, often by a large margin.

Needless to say, when no adaptation was found to take place the effect upon performance was often found to be detrimental. However, performance was seen to decline even when adaptation took place. Part of the reason for this is that it takes time for the crossover probability to evolve to the right value, by which time much of the useful search has already been performed and the impact of the evolved crossover probability is much reduced — getting the operator probabilities right at the start of the GA run appears to be important. Meta-learning was found to affect performance in a similar manner, occasionally making matters worse. No trends in the externally set co-evolution crossover probability were found, but as adaptation does not occur reliably, if at all, this is not particularly surprising.

Encoding of operator parameters was found to be more successful: adaptation was observed more often, however performance was still found to be degraded — the time taken to adapt precludes efficient search at the early stages of the GA run. Genetic algorithm performance was found to be sensitive to the operator parameters used.

The above suggest that this is not a good adaptation technique to use. So why has previous work advocated this approach? The reason may lie in the concentration of work on adaptation as an end in itself — in some cases, the genetic algorithm does adapt its operator settings. The difference with this study is that an emphasis has been placed upon the effect upon GA performance, questioning the implicit assumption that operator adaptation *must* be a good thing — this is not necessarily the case.

The above does not mean that adaptation is a waste of time — instead the shortcomings of co-evolution have to be overcome. A possible problem with co-evolution is that it is an *implicit* adaptation method: operator productivity information is gained and acted upon in an indirect fashion.

7.4 Adaptation Using COBRA

Another approach is to separate the adaptation mechanism from the main GA and make the measurement of operator performance and the act of adaptation explicit. This can be achieved by a 'learning-rule' adaptation method. The simplest of these was investigated: COBRA which periodically re-ranks the operator probabilities according to some measure of performance (operator productivity was used in this study).

Performance was found to be relatively robust towards the gap between operator probability re-ranking (most often a slight upward trend in performance was observed as a result of the better sampling of the operator productivities). Instead it was found that the initial operator probabilities were the main factor affecting performance.

No improvement in performance was found to occur when COBRA was being used, however the GA was often made less sensitive to the operator probabilities provided when COBRA was used (it appears to reduce to effect of bad choices), which in some applications may be useful — it may well be easier than looking over a larger number of conventional GA runs to obtain equivalent performance.

Also, the performance of COBRA seemed to indicate a bias towards speed of search, rather than quality. The extent of this trade-off can be controlled by the initial operator probabilities — an ability that adaptation by co-evolution lacks (when you can get it to work).

COBRA was found to be detrimental for some problems (such as Long Path with a steady-state population model). This appears to manifest itself in cases where the productivity of an operator is high (thus COBRA assigns it a high probability), but the preferred probability of the operator is low. In these cases COBRA should not be used.

7.5 Operator Productivity: The Right Measure To Use?

Operator productivity was found to be both a poor basis with which to assign operator probabilities for a static GA, and to mislead methods such as COBRA. This suggests that operator productivity is but one facet of the role that the operators play in a genetic algorithm.

The concept of operator productivity ignores the interactions between operators. One instance is the maintenance of diversity lost by processes such as drift and premature convergence (crossover is useless if all the members of the GA population are identical). A role of mutation in this case is to counteract these effects so that crossover remains effective. An example of this is the Counting Ones problem, which can fail to reach the optimum if the last few loci that need to be ones in the population are all zeros — crossover is then useless.

Another aspect that is ignored is the ability of crossover to spread and combine improvements gained by other operators throughout the population — this may be case for Long Path problems where improvements by mutation were quickly followed by improvements in mutation.

7.6 Final Points

Operator adaptation was not found to be as useful as the initial arguments had promised, but some utility was discovered (in performance sensitivity), and more uses may arise with different problems than those considered here. However, some results of practical importance have been found:

- Adaptation is not necessarily a good thing.

- Other factors apart from operator productivity are of importance when assigning operator probabilities — this requires some knowledge of the problem, what difficulties it can pose for a GA, and what is trying to be achieved.

- For adaptation to occur reliably, the adaptation mechanism should be separated from the main GA and the information upon which decisions are made should be explicitly measured.

- If improvements in performance occur, they are likely to be in speed of search, to the possible detriment of solution quality.

It is useful to note that the assignment of operator probabilities is, in effect, providing knowledge[1] to the GA on how to solve the problem. This is addition to the knowledge provided by the representation, operators, fitness function, etc. This constitutes a form of knowledge acquisition, of which the choice of adaptation method is a part.

Finally, it is now becoming clear that different problems present difficulties to optimisation techniques for different reasons. Thus it is important rather than looking for 'magic bullets', research instead turns to characterising the different pitfalls that optimisers may face, how they can be overcome, and how these pitfalls can be detected when tackling real world problems.

---

[1] Albeit in a somewhat implicit form.

Chapter 8

Further Work

8.1 Overview

This chapter provides suggestions for further work that attempt to answer some outstanding questions raised by this study. One of the conclusions of this work is that knowledge of the problem, and what difficulties it presents to the GA should be taken into account. Although these questions are important, they pertain to genetic algorithms research in general, not just operator adaptation. Therefore the suggestions below will concentrate upon issues that are more directly related to operator adaptation.

8.2 Co-evolutionary Methods

The results obtained by this technique were disappointing, but three suggestions to improve adaptation are described below:

First, one reason cited for the degradation in performance observed, was that by the time a suitable operator probability was evolved, most of the useful search had already been performed, and the positive effect of the evolved operator probability would be slight. The effect of this could be minimised by distributing the initially generated operator probabilities around a value that is known to be effective at the start of the GA run.

Second, a change in the fitness function could be used to place additional selection pressure upon the encoded operator probabilities. Previous work [Spears 95] added the difference of the raw fitnesses of the parent and child to the child's fitness function, thus rewarding children that are both good *and* better than their parents. Adaptation was found to be improved as a result — also, performance was improved for both the conventional and adaptive GA as a result.

Third, most of the work on adaptation by co-evolution was been performed by the Evolution Strategy (ES) community. A major difference between the ES and GA approaches is that the ES takes a 'survival of the fittest' approach to selection, whereas the GA takes a 'reproduction of the fittest approach'. The question is: does this make any difference?

8.3 Adaptation Using COBRA

This method appears to bring some benefits, but no improvements in performance were attained. This is the question to be addressed — what properties must a problem have for COBRA to produce improvements in performance?

Fortunately, one class of problems are known to exhibit improved performance when COBRA is used: timetabling problems. In addition, recent work [Corne 95] has shed some light upon what makes timetabling problems easy or hard for a genetic algorithm — it is possible (by adding or removing constraints as appropriate) to vary the properties of the problem in a controlled manner. Therefore it would be advantageous to use timetabling problems as a test-bed by taking examples for which COBRA was found to be effective and modifying them to see what effect it has.

A second point to be addressed is the measure of benefit used. For the Long Path problem considered here, operator productivity was found to mislead the adaptation mechanism — the reason cited was that operator productivity captures only one facet of the role that the operators play in a genetic algorithm. An investigation of the effect of using different measures of operator performance may be in order. Suggestions include the measures used in alternative learning rule methods, and re-ranking operators on the basis of diversity measurements (i.e assign a high rank to mutation when the population diversity is below a certain level).

8.4 Final Remarks

For the sake of completeness it would be desirable to investigate the other learning rule methods (i.e. [Davis 89] and [Julstrom 95]) to see how they compare with COBRA — would the added complexity they bring prove to be worthwhile? It is also interesting to note that Davis used his adaptation technique to derive time-varying schedules for his operator probabilities — this is worth examining further.

Also in many cases, the presence of more than two operators is found to improve GA performance. However, this greatly increases the search space of possible operator probabilities — it would be reasonable to expect that operator adaptation would be of greater use in these situations. An investigation is required to see if this is the case.

Finally the issue of operator adaptation cannot be separated from the issues of setting static operator probabilities, and the roles that the operators play in the genetic algorithm. Investigations into these questions would be relevant to operator adaptation, and operator adaptation could be used as a investigative tool in this area.

Bibliography

[Altenberg 94] L. Altenberg. The Evolution of Evolvability in Genetic Programming. In K. E. Kinnear, editor, *Advances in Genetic Programming*. MIT Press, 1994.

[Bäck 91] T. Bäck. Self-Adaptation in Genetic Algorithms. In *Proceedings of the 1st European Conference on Artificial Life*, pages 263–271. MIT Press, 1991.

[Bäck 92] T. Bäck. The Interaction of Mutation Rate, Selection and Self-Adaptation Within a Genetic Algorithm. In Manner and Manderick [Manner & Manderick 92].

[Bäck *et al.* 91] T. Bäck, F. Hoffmeister, and H. P. Schwefel. A Survey of Evolution Strategies. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 2–9. San Mateo: Morgan Kaufmann, 1991.

[Cartwright & Harris 93] Hugh M. Cartwright and Stephen P. Harris. Analysis of the distribution of airborne pollution using genetic algorithms. *Atmospheric Environment*, 27:1783–1791, 1993.

[Cleveland & Smith 89] Gary A. Cleveland and Stephen F. Smith. Using Genetic Algorithms to Schedule Flow Shop Releases. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms and their Applications*, pages 160–169. San Mateo: Morgan Kaufmann, 1989.

[Corne 95] D. Corne. Personal Communication, August 1995.

[Corne *et al.* 93] D. Corne, H.-L. Fang, and C. Mellish. Solving the Module Exam Scheduling Problem with Genetic Algorithms. In Paul W. H. Chung, Gillian Lovegrove, and Moonis Ali, editors, *Proceedings of the Sixth International Conference in Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pages 370–373. Gordon and Breach Science Publishers, 1993.

[Corne *et al.* 94] D. Corne, P. Ross, and H.-L. Fang. GA Research Note 7: Fast Practical Evolutionary Timetabling. Technical report, University of Edinburgh Department of Artificial Intelligence, 1994.

[Coyne & Paton 94] J. Coyne and R. Paton. Genetic Algorithms and Directed Adaptation. In Terry C. Fogarty, editor, *Selected Papers: AISB Workshop on Evolutionary Computing, Lecture Notes in Computer Science No 865*, pages 103–114. Springer Verlag, 1994.

[Dannenbring 77] D. G. Dannenbring. An Evaluation of Flowshop Sequencing Heuristics. *Manag. Sci.*, 23:1174–1182, 1977.

[Darwin 59] C. Darwin. *On the Origin of Species*. John Murray, London, 1859.

[Davis 89] L. Davis. Adapting Operator Probabilites in Genetic Algorithms. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms and their Applications*, pages 61–69. San Mateo: Morgan Kaufmann, 1989.

[Davis 91] L. Davis, editor. *Handbook of Genetic Algorithms*. New York: Van Nostrand Reinhold, 1991.

[DeJong 75] K. A. DeJong. *Analysis of Behavior of a Class of Genetic Adaptive Systems.* Unpublished PhD thesis, The University of Michigan, 1975.

[Fang 92] Hsiao-Lan Fang. Investigating genetic algorithms in scheduling. Unpublished M.Sc. thesis, Department of Artificial Intelligence, University of Edinburgh, 1992.

[Fogarty 89] T. C. Fogarty. Varying The Probability of Mutation In The Genetic Algorithm. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms and their Applications*, pages 104–109. San Mateo: Morgan Kaufmann, 1989.

[Fogel & Atmar 90] D. B. Fogel and J. W. Atmar. Comparing genetic operators with guassian mutations in simulated evolutionary processes using linear systems. *Biological Cybernetics*, 63:111–114, 1990.

[Forrest & Mitchell 93] Stephanie Forrest and Melanie Mitchell. Relative Building Block Fitness and the Building Block Hypothesis. In L. Darrell Whitely, editor, *Foundations of Genetic Algorithms 2*. San Mateo: Morgan Kaufmann, 1993.

[Goldberg 87] D. G. Goldberg. Simple genetic algorithms and the minimal deceptive problem. In L. Davis, editor, *Genetic Algorithms and Simulated Annealing*, pages 74–88. San Mateo: Morgan Kaufmann, 1987.

[Goldberg 89] David E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning.* Reading: Addison Wesley, 1989.

[Grefensette 86] J. J. Grefensette. Optimisation of Control Parameters for Genetic Algorithms. *IEEE Tranactions on Systems Man & Cybernetics (SMC)*, 16(1):122–128, 1986.

[Grefenstette 93] J. J. Grefenstette. Deception Considered Harmful. In L. Darrell Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 75–91. San Mateo: Morgan Kaufmann, 1993.

[Hart & Belew 91] W. E. Hart and R. K. Belew. Optimising an arbitary function is hard for the genetic algorithm. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 108–114. San Mateo: Morgan Kaufmann, 1991.

[Hesser & Männer 91] J. Hesser and R. Männer. Towards an optimal mutation probability for genetic algorithms. In H. P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature - Proceedings of 1st Workshop, PPSN 1*, volume 496 of *Lecture Notes in Computer Science*, pages 23–32, Dortmund, Germany, 1-3 Oct 1991. Springer-Verlag, Berlin, Germany.

[Holland 75] John H. Holland. *Adaptation in Natural and Artificial Systems.* Ann Arbor: The University of Michigan Press, 1975.

[Horn *et al.* 94] J. Horn, D. E. Goldberg, and K. Deb. Long Path Problems. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Parallel Problem Solving from Nature, PPSN III*, pages 149–159. Springer Verlag, 1994.

[Ingber & Rosen 92] L. Ingber and B. Rosen. Genetic algorithms and very fast simulated annealing - a comparison. *Mathematical and Computer Modeling*, 16(11):87–100, Nov 1992.

[Johnson 54] S. M. Johnson. Optimal two and three stage production schedules with set-up times included. *Naval Res. Logist. Q.*, 61, 1954.

[Juels & Wattenberg 94] Ari Juels and Martin Wattenberg. Stochastic Hill-climbing as a Baseline Method for Evaluating Genetic Algorithms. Technical report, UC Berkeley, 1994.

[Julstrom 95] Bryant A. Julstrom. What have you done for me lately? adapting operator probabilities in a steady-state genetic algorithm. In L. J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 81–87, San Francisco, Ca., 1995. Morgan Kaufmann.

[Kan 76] A. H. G. Rinnooy Kan. *Machine Sequencing Problems: Classification, complexity and computations*. Martinus Nijhoff, The Hague, 1976.

[Karimi & Ku 88] I. A. Karimi and H. M. A. Ku. Scheduling in Multi-product Batch Processes with Finite Interstage Storage: A Mixed Linear Program Formulation. *Ind. Eng. Chem. Res.*, 27:1840–1848, 1988.

[Ku *et al.* 87] H. M. A. Ku, D. Rajagopalan, and I. A. Karimi. Scheduling in Batch Processes. *Chem. Eng. Prog.*, 83(8), 1987.

[Levenick 95] Jim Levenick. Metabits: Generic endogenous crossover control. In L. J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 88–95, San Francisco, Ca., 1995. Morgan Kaufmann.

[Manner & Manderick 92] R. Manner and B. Manderick, editors. *Parallel Problem Solving from Nature, 2*. Elsevier Science Publisher B.V., 1992.

[Mott 90] G. F. Mott. Optimising Flowshop Scheduling Through Adaptive Genetic Algorithms. Chemistry Part II Thesis, Oxford University, 1990.

[Mühlenbein 92] H. Mühlenbein. How Genetic Algorithms Really Work: Mutation & Hillclimbing. In Manner and Manderick [Manner & Manderick 92], pages 15–26.

[Osman & Potts 89] I. H. Osman and C. N. Potts. Simulated annealing for permutation flow-shop scheduling. *OMEGA*, 17:551–557, 1989.

[Radcliffe 91] N. J. Radcliffe. Equivalence Class analysis of genetic Algorithms. *Complex Systems*, 5(2):183–205, 1991.

[Radcliffe 92] N. J. Radcliffe. Non-Linear Genetic Representations. Technical report, Edinburgh Parallel Computing Centre, 1992.

[Reeves 93] C. R. Reeves. Improving the efficiency of tabu search in machine sequencing problems. *J.Opl.Res.Soc.*, 44:375–382, 1993.

[Reeves 95] C. R. Reeves. A genetic algorithm for flowshop sequencing. *Computers & Ops. Res.*, 22:5–13, 1995.

[Ross & Corne 95] Peter Ross and Dave Corne. Applications of Genetic Algorithms. *AISB Quarterly*, 89:23–30, 1995.

[Schaffer & Eshelman 91] J. D. Schaffer and L. J. Eshelman. On Crossover as an Evolutionarily Viable Strategy. In R.K. Belew and L.B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 61–68. San Mateo: Morgan Kaufmann, 1991.

[Schaffer & Morishima 87] J. D. Schaffer and A. Morishima. An Adaptive Crossover Distribution Mechanism for Genetic Algorithms. In J. J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*. Lawrence Erlbaum Associates, 1987.

[Schaffer *et al.* 89] J. D. Schaffer, R. A. Caruana, L. J. Eshelman, and R. Das. A study of control parameters affecting online performance of genetic algorithms for function optimisation. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms and their Applications*, pages 51–61. San Mateo: Morgan Kaufmann, 1989.

[Spears & DeJong 91] W. M. Spears and K. A. DeJong. An Analysis of Multi-Point Crossover. In J. E. Gregory, editor, *Foundations of Genetic Algorithms*. San Mateo: Morgan Kaufmann, 1991.

[Spears 93] W. M. Spears. Crossover or Mutation? In Darrell Whitley, editor, *Foundations of Genetic Algorithms 2*. San Mateo: Morgan Kaufmann, 1993.

[Spears 95] W. M. Spears. Adapting Crossover in Evolutionary Algorithms. In *Proceedings of the Evolutionary Programming Conference*. 1995.

[Starkweather *et al.* 91] T. Starkweather, S. McDaniel, K. Mathias, D. Whitley, and C. Whitley. A comparison of genetic sequencing operators. In R.K. Belew and L.B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 69–76. San Mateo: Morgan Kaufmann, 1991.

[Taillard 93] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of operations research*, 64:278–285, 1993.

Appendix A

Adaptive: A Genetic Algorithm For Operator Adaptation

A.1 Overview

The experiments described in this study were performed on an implementation of the genetic algorithm called *Adaptive*, which implements facilities for operator adaptation. The features of the program are listed below:

- Written in ANSI C for portability.
- Five problems are built in: Counting Ones, Order-3 'tight' Deceptive, Royal Road, Long Path, and $n/m/P/C_{max}$.
- Four population models are implemented.
- Talliard benchmark generator built in.
- Binary and permutation representations and operators supported.
- Statistics of operator performance, etc. are obtainable.
- Implements 7 types of operator adaptation.
- Easily extensible - new problems/operators can be easily added.

The following sections will give details on the functionality available, how to use the program, and the compilation procedure.

A.2 How To Use The Program

This describes how to use version 1.4 of *Adaptive*. This begins with an introduction on how to run the program and then describes each item of functionality in turn and how to use it.

A.2.1 Running The Program

To run the program type '`adaptive`' at the command line followed by any options (each preceded with '-'). For example:

```
adaptive -t1 -l100 -o0[0.75] -o1[0.25] -i1
```

This assumes that the executable program is in the current directory. The program sends all its output to the standard output (stdout); this may be redirected to a file or piped to another program (e.g. mail) as desired.

The options are available by running the program with the option '`-h`' which gives the list of options below. Each option will be described in further detail later. Default values are given in brackets.

```
-a<int>         Assign adaption method (NONE)
-b<float>       Adjust selection bias (1.5)
-c<int>         If > ZERO, GA won`t stop if converged (1)
-d<int>         Assign operator settings report interval (10001)
-e<int>         Max. number of evals (10000)
-f<int>         Talliard RNG seed (479340445)
-g<int>         Operator stats report interval (10001)
-i<int>         Number of times GA is to be run (100)
-j<int>         Size of operator log buffer (100)
-l<int>         Assign string length (20)
-m<int>         Assign population model (STEADY STATE)
```

```
-n<int>            Assign plot interval (10001)
-o<int>[<float>]   Assign operator probabilities (varies)
-p<int>            Assign population size (100)
-q<int>[<float>]   Assign operator parameters (varies)
-r<int>            Assign reporting interval (10001)
-s<long>           Assign RNG Seed (1)
-t<int>            Assign test problem (FLOWSHOP)
-u<0.0-1.0>        FP uniform xover parameter (0.1)
-v<float>          FP mutation stddev (0.1)
-x<0.0-1.0>        Coevolution xover rate (0.8)
```

A.2.2 Selecting The Problem

Five test problems are available, to select which the option '`-t`' is invoked, followed by an integer depicting which test problem is to be used. The test problems[1], with their assigned integers is given below:

`Flowshop Sequencing:`	`0`
`Counting Ones:`	`1`
`Order-3 'tight' deceptive problem`	`2`
`Royal Road problem (R1)`	`3`
`Long Path problem (Root2Path)`	`4`

So for example, '`-t1`' sets the Counting Ones problem as the problem to be attempted. The default is the $n/m/P/C_{max}$ problem.

Two other options are of importance. The first sets the size of the problem to be tackled, this is accomplished by using the '`-l`' flag followed by an integer (the default is 20). The program will check if the length is valid (e.g. the size of the deceptive problem must be divisible by 3), and, if not, will modify the length to ensure that it is and inform the user.

The second option only applies when the $n/m/P/C_{max}$ problem is selected. This program implements the random number generator used to generate the

[1]The Flowshop Sequencing problem is often also called the $n/m/P/C_{max}$ problem.

processing times matrices used in the Talliard benchmarks. To use any of the benchmark problems, the option '`-f`' is used, followed by a long integer which is the seed (the default is 479340445) given to the Talliard random number generator, for the benchmark problem to be attempted.

A.2.3 The Operator Settings

The operator probabilities are set using the option '`-o`' followed by an integer depicting which operator probability is to be assigned and a floating point number, enclosed in square brackets which is equal to or less than 1.0.

A value of the floating point number that is negative is taken by the program to mean that that operator is not to be used; otherwise the program assigns the value as the operator probability. For example, '`-o1[0.5]`' assigns probability 0.5 to operator 1. The program automatically renormalises the operator probabilities before the GA proper is run, so as to ensure that the operator probabilities sum to one.

The operators that are available depend upon the problem that is to be tackled. Problems using a permutation representation (i.e. the $n/m/P/C_{max}$ problem) have the following operators[2] available:

```
Modified PMX with random mating (0.8):                 0
Shift Mutation (0.2):                                  1
Swap Mutation (Not Used):                              2
Modified PMX with fitness-biased mating (Not Used):    3
RRR with random mating (Not Used):                     4
RRR with fitness-biased mating (Not Used):             5
```

Binary-coded problems (all the others) have the following operators available:

```
Uniform Crossover with random mating (0.8):              0
Binary Mutation (0.2):                                   1
```

[2]RRR is the acronym for Random Respectful Recombination.

```
Uniform Crossover with fitness-biased mating (Not Used):   2
One-pt Crossover with random mating (Not Used):            3
One-pt Crossover fitness-biased mating (Not Used):         4
```

In both cases the default operator probabilities are given in brackets.

Operator parameters are set similarly, but with the '-q' option. Parameters will be ignored for the permutation and binary one-point crossover operators as they are not used. The meaning of the parameter provided depends upon the operator: for uniform crossover (default 0.5) it is the probability that the allene of the child at a given locus was from the second parent: for binary mutation it is the bitwise mutation rate, linearly rescaled to the range $1/l$ to $5/l$.

A.2.4 Setting The GA Population

Two options are relevant to setting to GA population. The first is the size of the population, the second the population model used.

The Population Size

This is adjusted by using the option '-p' immediately followed by an integer (e.g. `-p150` sets the population size to 150). The default is 100.

The Population Model

This is set using the option '-m' immediately followed by an integer representing the population model to be used (e.g. `-m1` makes the program use a generational model). The default population model is steady-state. The models supported, with their assigned integers, are given below.

```
Steady-State:                       0
Generational:                       1
Generational with Elitism:          2
(N+N) Evolution Strategy:           3
```

A.2.5 Setting The Adaptation Technique

This is adjusted by using the option '`-a`' immediately followed by an integer representing the adaptation method to be used (e.g. `-a7` makes the program use COBRA). The default is no adaptation. The techniques implemented are given below with their assigned integers.

`No Adaptation:`	`0`
`Co-evolution using disruptive operators`	`1`
`Co-evolution using disruptive operators with meta-learning`	`2`
`Co-evolution using non-disruptive operators`	`3`
`Co-evolution using non-disruptive operators with meta-learning`	`4`
`Co-evolution of operator parameters`	`5`
`Co-evolution of operator parameters and probabilities`	`6`
`COBRA`	`7`

Extra Options For Co-evolution

The co-evolution methods are described in more detail in the main dissertation.

When co-evolution methods 1, 3, 5, and 6 are being used, a co-evolution crossover probability can be set by using the option '`-x`' immediately followed by a real number in the range 0.0 to 1.0. For example,`-x0.5` sets the co-evolution crossover probability to 0.5. The default value is 0.8.

For co-evolution methods 3, 4, 5, and 6, the parameters of the non-disruptive co-evolution operators can be changed. The standard deviation of mutation operator (default is 0.05) is changed by the option '`-v`' immediately followed by a real number. The uniform crossover parameter is changed by the option '`-u`' immediately followed by a decimal number in the range 0.0 to 1.0.

When deciding which operators to adapt, the program only co-evolves those which have been assigned operator probabilities greater than zero. This can be controlled by the use of the '`-o`' as described in A.2.3.

Extra Options For COBRA

COBRA is described in detail in the main dissertation.

The initial operator probabilities are set in the same way as the static operators, i.e. using the option '`-o`' as described in A.2.3.

The number of evaluations between operator probability re-rankings is set by changing the length of the operations log (which records the effect of a given number of previous operations). This is adjusted by using the '`-j`' followed by an integer which is the new log size.

A.2.6 Obtaining Additional Statistics

The following options detail how to obtain additional information about the state of the GA during a given run.

Population Summary

The option '`-r` followed by an integer, sets the interval (number of evaluations) between the program proving a detailed summary of the GA population.

Operator Information

The option '`-d`' followed by an integer, sets the interval (number of evaluations) between the program providing a detailed summary of the GA operator probabilities and parameters currently being used. The default interval is 10001 evaluations.

The option '`-e`' followed by an integer, sets the interval (number of evaluations) between the program providing a detailed summary of the current performance of the operator being used. This includes information on evolved operator probabilities/current operator probabilities[3], operator productivities and how many times each operator has been used recently. Again, the default interval is 10001 evaluations.

When these statistics are being complied, the program records the effect of operations in the past. The length of time that a record of an operation is kept

[3] This depends upon the adaptation mechanism being used.

can be adjusted by the option '`-j`' followed by an integer which is the length of the operator log. The default log length is 1000 operations.

Finally, the option '`-n`' followed by an integer, sets the interval (number of evaluations) between the program providing the mean and best fitnesses of the current GA population. The default interval is 10001 evaluations.

A.2.7 Miscellaneous

This details the other options available to the user.

Seeding The RNG

The random number generator used requires a seed. The default is one, however, this can be changed by using the option '`-s`' followed by a long integer which is the RNG seed.

Selection Pressure

A member of the GA population is selected to be operated upon on the basis of its fitness. The method used is *rank-based* selection. The selection pressure can be adjusted by the option '`-b`' followed by a real number between 1.0 and 2.0. The default is 1.5.

Stopping Conditions

This is controlled by 3 options. The first '`-c`' is followed by an integer. If the integer is greater than zero, the genetic algorithm will not stop when converged (i.e. when all the members of the population are the same). The default is not to stop when converged.

The number of evaluations to be performed before the GA stops is governed by the '`-e`' option, with an integer giving the number of evaluations. The default is 10000.

The third option is to run the GA multiple times. This is adjustable via the option '`-i`' followed by an integer which is the number of times the GA is to be run. All runs are identical with the exception that the RNG seed is incremented by one each time the GA is run again. This option is designed for when a sample over a large number is runs is required; in this case batch processing is desirable. The default setting is 50 iterations.

A.3 Compiling The Program

The program was written in ANSI C and so should be able to be complied for any machine with a suitable complier. The complier used in this study is GCC and a suitable Makefile is included in the demo directory. To compile the program follow the instructions below.

1. Type `cd ~andrewt\andrewtDEMO\PROJECT\` at the unix prompt.
2. Type `make` at the unix prompt to compile.
3. The executable produced will be called `adaptive` in the same directory.

The program can then be run by typing `adaptive` at the unix prompt.

Appendix B

Results Summary

B.1 Overview

Tables depicting the results obtained are given here, containing the mean and standard deviation of the performance over 50 runs with differing random number seeds. Two measures of performance are used:

- The quality of solution obtained after 10000 evaluations.
- The number of evaluations after which improvements in solution quality were no longer obtained (or 10000 generations - whichever is smaller).

The relevant crossover probabilities were studied in each case. For COBRA, a study of the gap between operator probability re-ranking was also made.

B.2 Results From A Steady-State GA

B.2.1 The $n/m/P/C_{max}$ Problem

Fitnesses For A Standard GA

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	2388.02 (69.00)	2259-2571	0.55	2401.68 (73.02)	2262-2595
0.05	2386.56 (70.58)	2257-2600	0.60	2395.78 (69.65)	2267-2592
0.10	2387.02 (71.80)	2246-2586	0.65	2402.62 (68.92)	2272-2581
0.15	2388.50 (71.58)	2246-2605	0.70	2398.84 (72.07)	2266-2585
0.20	2391.54 (67.60)	2252-2571	0.75	2403.76 (72.99)	2273-2597
0.25	2393.28 (73.24)	2256-2577	0.80	2407.88 (73.57)	2269-2594
0.30	2392.66 (76.80)	2253-2576	0.85	2412.06 (68.73)	2280-2587
0.35	2388.82 (69.31)	2243-2579	0.90	2412.62 (71.17)	2276-2600
0.40	2395.32 (76.89)	2249-2597	0.95	2420.92 (73.53)	2267-2611
0.45	2390.46 (72.83)	2263-2590	1.00	2475.38 (69.15)	2347-2703
0.50	2397.54 (75.24)	2262-2619	-	-	-

Evaluations Reqd. For A Standard GA

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	8218.38 (1470.41)	2512-9931	0.55	5064.36 (1616.38)	2380-9327
0.05	7716.84 (1640.75)	4044-9906	0.60	5269.70 (1870.82)	2135-9617
0.10	8023.78 (1564.42)	4366-9978	0.65	5197.28 (1820.64)	2827-9984
0.15	7799.44 (1694.82)	3868-9950	0.70	5762.26 (1832.00)	2313-9670
0.20	7017.02 (1564.87)	4043-9990	0.75	5500.00 (2303.09)	2499-9704
0.25	6644.76 (1793.17)	2735-9779	0.80	6019.88 (2288.24)	2071-9990
0.30	6636.74 (1859.50)	2223-9949	0.85	6023.40 (2328.22)	1737-9870
0.35	5990.58 (1763.17)	3392-9255	0.90	6328.64 (2280.96)	1849-9969
0.40	6048.18 (1629.73)	2813-9442	0.95	6547.06 (2809.42)	1332-9943
0.45	5836.18 (1758.96)	3101-9642	1.00	1614.20 (406.13)	722-2390
0.50	5487.82 (1971.21)	2037-9669	-	-	-

Fitnesses For Co-evolution Method #1

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	2398.00 (72.09)	2258-2591	0.55	2398.80 (72.73)	2256-2571
0.05	2396.18 (74.40)	2251-2566	0.60	2397.56 (72.03)	2263-2582
0.10	2400.54 (69.95)	2270-2591	0.65	2399.30 (74.41)	2258-2600
0.15	2398.42 (72.73)	2270-2587	0.70	2393.60 (71.86)	2249-2581
0.20	2397.46 (73.98)	2263-2596	0.75	2396.92 (69.23)	2259-2591
0.25	2402.10 (71.66)	2264-2577	0.80	2401.98 (69.11)	2251-2589
0.30	2398.28 (77.36)	2249-2613	0.85	2397.56 (68.50)	2249-2578
0.35	2398.38 (75.71)	2269-2610	0.90	2402.32 (71.25)	2274-2605
0.40	2398.56 (75.59)	2245-2575	0.95	2397.68 (73.91)	2249-2592
0.45	2394.46 (71.13)	2246-2596	1.00	2393.70 (69.02)	2256-2562
0.50	2398.64 (69.81)	2264-2584	-	-	-

Evaluations Reqd. For Co-evolution Method #1

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	5062.66 (1628.49)	1691-9205	0.55	5549.68 (1892.15)	2273-9795
0.05	5877.22 (1853.12)	2958-9641	0.60	5726.60 (1804.30)	2076-9061
0.10	5580.46 (2035.23)	2551-9889	0.65	5325.46 (1948.13)	2392-9872
0.15	5614.60 (2030.46)	2332-9681	0.70	5982.80 (2026.55)	2111-9990
0.20	5809.22 (2116.85)	2619-9862	0.75	5175.56 (1567.02)	2140-9056
0.25	5230.20 (1618.66)	2111-9571	0.80	5407.86 (1957.98)	2356-9841
0.30	5786.22 (2022.51)	2411-9893	0.85	5419.94 (2032.83)	1963-9773
0.35	5566.96 (1765.41)	2768-9563	0.90	5506.70 (1974.51)	2018-9951
0.40	5452.38 (2049.84)	2091-9965	0.95	6024.22 (2052.55)	2296-9864
0.45	5644.16 (2202.00)	9959-2119	1.00	5665.82 (1949.79)	1912-9450
0.50	6063.42 (2003.02)	3139-9920	-	-	-

Results For Co-evolution Method #2

Measure	Mean (Std Dev)	Range
Fitness	2400.42 (73.17)	2263-2596
Evals. Reqd.	5468.44 (1747.79)	1992-9272

Fitnesses For Co-evolution Method #3

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	2400.92 (76.00)	2251-2584	0.55	2402.82 (70.48)	2275-2573
0.05	2402.30 (77.04)	2253-2611	0.60	2403.60 (71.84)	2257-2594
0.10	2401.66 (73.61)	2252-2611	0.65	2402.76 (74.44)	2261-2588
0.15	2405.08 (71.53)	2248-2588	0.70	2403.12 (69.49)	2254-2569
0.20	2402.30 (71.22)	2249-2596	0.75	2400.56 (73.67)	2261-2595
0.25	2402.58 (74.17)	2269-2588	0.80	2402.30 (71.29)	2249-2602
0.30	2398.26 (78.85)	2244-2582	0.85	2402.84 (69.99)	2275-2574
0.35	2402.72 (76.19)	2264-2607	0.90	2397.14 (71.26)	2274-2583
0.40	2402.74 (71.17)	2257-2585	0.95	2400.64 (78.06)	2249-2586
0.45	2403.98 (76.60)	2266-2667	1.00	2398.90 (73.75)	2259-2592
0.50	2402.32 (75.68)	2265-2588	-	-	-

Evaluations Reqd. For Co-evolution Method #3

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	5827.94 (2159.43)	2508-9961	0.55	5899.04 (2154.16)	2169-9778
0.05	5911.50 (2174.71)	2436-9805	0.60	6362.08 (2029.42)	1781-9996
0.10	5887.84 (1944.12)	2096-9832	0.65	6056.94 (2130.96)	2407-9750
0.15	5986.36 (1828.83)	2400-9976	0.70	6282.36 (2215.13)	1925-9928
0.20	5962.26 (2344.92)	2140-9866	0.75	6152.04 (2151.11)	1541-9968
0.25	6270.48 (2053.45)	1441-9820	0.80	5641.30 (1714.27)	2305-9772
0.30	6057.00 (2101.50)	2543-9932	0.85	5668.20 (2261.55)	2348-9797
0.35	5720.80 (1870.98)	2560-9963	0.90	5389.00 (1994.64)	2240-9851
0.40	5880.48 (2033.32)	2662-9933	0.95	6050.58 (2400.45)	2002-9974
0.45	5571.82 (2000.00)	2292-9384	1.00	5813.14 (2037.62)	2114-9701
0.50	5713.10 (2098.51)	2627-9405	-	-	-

Results For Co-evolution Method #4

Measure	Mean (Std Dev)	Range
Fitness	2405.30 (73.15)	2284-2597
Evals Reqd.	6285.76 (2227.99)	2151-9922

Fitnesses For COBRA with p(Xover)= 0.35 and Varying Gap Size

Gap Size	Mean (Std Dev)	Range	Gap Size	Mean (Std Dev)	Range
200	2398.10 (73.11)	2260-2604	1200	2397.10 (72.93)	2250-2587
400	2396.86 (69.88)	2247-2595	1400	2398.08 (73.46)	2249-2589
600	2398.20 (72.14)	2265-2611	1600	2391.56 (70.62)	2243-2599
800	2401.14 (73.20)	2260-2598	1800	2393.60 (69.82)	2243-2573
1000	2398.70 (74.63)	2272-2592	2000	2394.86 (71.37)	2243-2583

Evaluations Reqd. For COBRA with p(Xover)= 0.35 and Varying Gap Size

Gap Size	Mean (Std Dev)	Range	Gap Size	Mean (Std. Dev)	Range
200	5767.14 (2142.31)	2096-9924	1200	5862.30 (2083.41)	2435-9985
400	5979.40 (2093.04)	2420-9870	1400	5099.92 (1698.11)	1992-9893
600	5435.56 (1803.91)	2586-9650	1600	5992.94 (1916.23)	2450-9899
800	6080.88 (1950.75)	1311-9953	1800	5597.48 (1927.51)	2712-9300
1000	5270.60 (2006.25)	2072-9234	2000	5469.92 (1784.74)	2577-9730

Fitnesses For COBRA with Gap Size = 1000

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.05	2406.36 (70.49)	2259-2597	0.55	2400.78 (74.21)	2262-2595
0.10	2409.16 (72.35)	2234-2591	0.60	2395.84 (69.66)	2267-2592
0.15	2406.88 (74.80)	2256-2618	0.65	2403.42 (68.43)	2272-2581
0.20	2407.02 (77.52)	2271-2601	0.70	2398.24 (72.18)	2266-2585
0.25	2410.92 (73.81)	2266-2601	0.75	2404.22 (72.99)	2273-2597
0.30	2400.52 (72.41)	2260-2573	0.80	2408.16 (74.23)	2248-2590
0.35	2398.70 (74.63)	2272-2592	0.85	2409.34 (68.88)	2280-2582
0.40	2399.54 (70.72)	2275-2589	0.90	2406.50 (72.69)	2276-2600
0.45	2395.68 (76.85)	2236-2625	0.95	2405.98 (74.97)	2270-2611

Evaluations Reqd. For COBRA with Gap Size = 1000

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.05	6257.40 (2287.78)	2136-9976	0.55	5049.50 (1638.45)	2380-9327
0.10	6335.56 (2419.19)	2299-9993	0.60	5209.80 (1840.70)	2135-9417
0.15	5799.62 (2366.88)	2033-9997	0.65	5157.46 (1766.96)	2827-9551
0.20	5682.40 (2281.50)	2097-9773	0.70	5799.82 (1893.47)	2313-9889
0.25	6116.66 (2372.59)	2286-9974	0.75	5487.08 (2289.59)	2499-9704
0.30	5647.92 (2140.38)	1810-9958	0.80	6089.40 (2309.17)	2071-9990
0.35	5270.60 (2006.25)	2072-9234	0.85	6050.42 (2376.22)	1737-9986
0.40	5881.34 (2102.17)	2514-9884	0.90	6418.20 (2525.68)	1849-9881
0.45	5652.84 (1880.80)	2172-9529	0.95	6641.06 (2294.97)	1690-9920

B.2.2 The Counting Ones Problem

Fitnesses For A Standard GA

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00-0.90	100.00 (0.00)	100-100	1.00	99.42 (0.83)	96-100
0.95	99.90 (0.30)	99-100	-	-	-

Evaluations Reqd. For A Standard GA

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	7379.08 (612.61)	6156-8671	0.55	2560.98 (491.53)	1881-3973
0.05	5645.96 (525.51)	4571-6659	0.60	2484.98 (510.34)	1782-4162
0.10	4800.82 (442.44)	3874-5697	0.65	2406.44 (618.36)	1702-4471
0.15	4241.36 (370.37)	3482-5185	0.70	2319.20 (573.00)	1692-4005
0.20	3969.82 (374.83)	2956-4905	0.75	2178.68 (777.33)	1536-5327
0.25	3527.10 (446.65)	2831-4690	0.80	2172.42 (702.14)	1548-4856
0.30	3332.00 (361.30)	2546-4376	0.85	2483.48 (1280.15)	1476-7000
0.35	3161.86 (449.18)	2466-4477	0.90	2558.12 (1573.16)	1502-8632
0.40	2914.80 (312.99)	2313-3692	0.95	2226.46 (1600.86)	1438-8815
0.45	2759.72 (464.10)	1958-4210	1.00	1608.24 (112.65)	1426-2015
0.50	2560.54 (396.99)	1968-4048	-	-	-

Fitnesses For Coevolution Method #1

p(Xover)	Mean (Std Dev)	Range
ALL	100.00 (0.00)	100-100

Evaluations Reqd. For Coevolution Method #1

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	2632.02 (519.61)	1974-5082	0.55	2594.08 (471.27)	1822-3626
0.05	2563.30 (314.41)	1819-3383	0.60	2687.82 (598.76)	1936-5436
0.10	2731.12 (493.27)	2038-3959	0.65	2611.66 (437.66)	1971-3935
0.15	2749.00 (498.53)	1924-4380	0.70	2673.58 (696.25)	1923-6175
0.20	2544.14 (352.56)	2039-3663	0.75	2655.80 (467.02)	1996-3976
0.25	2597.12 (498.04)	1878-4916	0.80	2599.90 (601.92)	1817-5125
0.30	2738.08 (474.17)	1925-4290	0.85	2702.68 (616.89)	1771-5046
0.35	2581.92 (364.55)	2007-3528	0.90	2583.72 (426.20)	1738-3588
0.40	2728.84 (622.91)	1994-5787	0.95	2767.38 (621.27)	2098-4610
0.45	2676.28 (729.41)	1871-5906	1.00	2478.68 (418.23)	1783-3754
0.50	2579.62 (467.73)	1892-4006	-	-	-

Results For Co-evolution Method #2

Measure	Mean (Std Dev)	Range
Fitness	100.00 (0.00)	100-100
Evals. Reqd.	2674.82 (640.45)	1926-5663

Fitnesses For Co-evolution Method #3

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	99.98 (0.14)	99-100	1.00	99.96 (0.28)	98-100
0.05-0.95	100.00 (0.00)	100-100	-	-	-

Evaluations Reqd. For Co-evolution Method #3

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	2989.18 (1229.79)	1725-7857	0.55	3083.18 (1490.19)	1655-8993
0.05	2708.32 (1227.75)	1723-9582	0.60	2732.74 (929.80)	1732-5420
0.10	3222.36 (1805.61)	1559-9441	0.65	3148.30 (1752.27)	1619-9757
0.15	3025.96 (1437.23)	1654-8865	0.70	3199.52 (1364.73)	1645-7939
0.20	3210.48 (1616.42)	1726-9833	0.75	3009.36 (1510.09)	1696-8245
0.25	3218.26 (1816.73)	1711-9705	0.80	2624.58 (842.42)	1727-6180
0.30	2764.12 (1205.49)	1579-7729	0.85	2935.98 (1172.32)	1717-9084
0.35	2828.76 (1330.05)	1625-8622	0.90	2669.22 (1082.81)	1649-7708
0.40	3074.38 (1582.08)	1667-7828	0.95	2862.96 (1251.57)	1720-9473
0.45	2475.10 (841.97)	1708-5676	1.00	2580.72 (770.34)	1599-5537
0.50	2862.92 (1038.74)	1675-6396	-	-	-

Results For Co-evolution Method #4

Measure	Mean (Std Dev)	Range
Fitness	99.98 (0.14)	99-100
Evals Reqd.	3406.64 (1933.15)	1742-8857

Fitnesses For Co-evolution Method #5

p(Xover)	Mean (Std Dev)	Range
ALL	100.00 (0.00)	100-100

Evaluations Reqd. For Co-evolution Method #5

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	3312.92 (967.66)	2316-7527	0.55	3546.78 (1451.88)	1989-9545
0.05	3204.42 (519.14)	2257-4362	0.60	3359.80 (1383.82)	1932-7197
0.10	3218.00 (1437.89)	1951-8478	0.65	3182.36 (1473.61)	1863-9040
0.15	3174.74 (1267.16)	1922-8700	0.70	2908.40 (1354.61)	1749-7934
0.20	3063.70 (1098.13)	1922-7157	0.75	3195.48 (1197.02)	1826-6330
0.25	3401.48 (1474.25)	1687-8906	0.80	2841.18 (1058.80)	1883-5970
0.30	3273.68 (1449.81)	1916-8534	0.85	2914.40 (1235.93)	1783-7334
0.35	3368.56 (1556.23)	1861-7802	0.90	3558.40 (1744.03)	1945-9426
0.40	3605.76 (1559.17)	1885-9517	0.95	3071.30 (1380.24)	1866-9383
0.45	3142.88 (1229.23)	1812-7260	1.00	3711.94 (1905.03)	1928-9844
0.50	2791.48 (1011.43)	1774-6320	-	-	-

Fitnesses For Co-evolution Method #6

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	99.94 (0.23)	99-100	0.55	99.84 (0.46)	98-100
0.05	99.72 (0.56)	98-100	0.60	99.98 (0.14)	99-100
0.10	99.86 (0.40)	98-100	0.65	99.94 (0.31)	98-100
0.15	99.88 (0.38)	98-100	0.70	99.96 (0.19)	99-100
0.20	99.90 (0.45)	97-100	0.75	99.86 (0.34)	99-100
0.25	99.76 (0.51)	98-100	0.80	99.88 (0.47)	97-100
0.30	99.86 (0.40)	98-100	0.85	99.90 (0.45)	97-100
0.35	99.90 (0.36)	98-100	0.90	99.94 (0.31)	98-100
0.40	99.88 (0.47)	97-100	0.95	99.76 (0.74)	96-100
0.45	99.84 (0.50)	98-100	1.00	99.80 (0.56)	97-100
0.50	99.76 (0.58)	97-100	-	-	-

Evaluations Reqd. For Co-evolution Method #6

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	3893.78 (2223.21)	1896-9623	0.55	3475.62 (1625.64)	1875-7756
0.05	3771.04 (1651.21)	1724-8787	0.60	3581.26 (1388.56)	1907-7502
0.10	4204.56 (2425.60)	1721-9997	0.65	3407.60 (1509.23)	1708-8528
0.15	3409.78 (1462.18)	1716-8031	0.70	3700.28 (1770.74)	1891-9896
0.20	4182.98 (1978.00)	1760-9218	0.75	3293.38 (1465.49)	1833-9352
0.25	3873.14 (2105.05)	1802-9649	0.80	3434.74 (1415.53)	1848-8676
0.30	4082.98 (2159.70)	1995-9856	0.85	3758.32 (1845.60)	1970-9973
0.35	3739.46 (1603.08)	1921-8257	0.90	3675.96 (1819.08)	1728-7937
0.40	4249.72 (2193.31)	1776-9921	0.95	3565.64 (1985.67)	1580-9752
0.45	3911.28 (1723.63)	1796-9989	1.00	3622.88 (1744.81)	1871-9158
0.50	4028.14 (2150.67)	1772-9718	-	-	-

Fitnesses For COBRA with p(Xover)= 0.8 and Varying Gap Size

Gap Size	Mean (Std Dev)	Range
ALL	100.00 (0.00)	100-100

Evaluations Reqd. For COBRA with p(Xover)= 0.8 and Varying Gap Size

Gap Size	Mean (Std Dev)	Range	Gap Size	Mean (Std Dev)	Range
200	2118.34 (565.99)	1548-4317	600-2000	2172.42 (702.14)	1548-4856
400	2139.42 (607.65)	1548-4317	-	-	-

Fitnesses For COBRA with Gap Size = 1000

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std. Dev)	Range
0.05	99.86 (0.44)	98-100	0.20-0.80	100.00 (0.00)	100-100
0.10	99.84 (0.46)	98-100	0.95	99.90 (0.30)	99-100
0.15	99.98 (0.14)	99-100	-	-	-

Evaluations Reqd. For COBRA with Gap Size = 1000

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.05	6525.94 (1720.12)	2755-9946	0.55	2560.98 (491.54)	1881-3973
0.10	5980.20 (1807.74)	2389-9884	0.60	2484.98 (510.33)	1782-4162
0.15	5279.04 (1958.92)	2505-9837	0.65	2406.44 (618.35)	1702-4471
0.20	4135.78 (1268.34)	2364-9919	0.70	2319.20 (573.00)	1692-4005
0.25	3712.16 (1231.82)	2271-8453	0.75	2178.68 (777.33)	1536-5327
0.30	3266.86 (669.99)	2011-4953	0.80	2172.42 (702.14)	1548-4856
0.35	3240.48 (1226.75)	1915-9573	0.85	2483.48 (1280.15)	1476-7000
0.40	2768.14 (363.89)	2060-3776	0.90	2558.12 (1573.15)	1502-8632
0.45	2802.56 (529.91)	1952-4164	0.95	2142.50 (1342.31)	1438-7331

B.2.3 The Order-3 Deceptive Problem

Fitnesses For A Standard GA

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	289.48 (2.707)	284-294	0.55	288.44 (2.808)	284-296
0.05	289.04 (3.130)	284-296	0.60	287.92 (2.770)	282-296
0.10	289.28 (3.219)	280-294	0.65	288.04 (2.416)	284-294
0.15	288.52 (2.586)	284-294	0.70	288.36 (2.644)	284-294
0.20	289.12 (3.077)	282-296	0.75	287.64 (2.322)	282-292
0.25	288.24 (3.010)	282-298	0.80	287.04 (2.408)	282-292
0.30	289.04 (2.306)	284-294	0.85	286.92 (2.629)	282-294
0.35	288.72 (2.845)	282-294	0.90	286.64 (2.278)	282-294
0.40	288.36 (2.904)	282-296	0.95	286.28 (2.623)	280-292
0.45	289.04 (2.441)	284-294	1.00	285.92 (2.497)	280-292
0.50	288.08 (2.799)	280-294	-	-	-

Evaluations Reqd. For A Standard GA

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	3762.70 (2695.86)	722-9809	0.55	3088.68 (2335.55)	680-9823
0.05	4209.68 (2772.85)	1154-9809	0.60	3018.04 (2215.88)	797-9457
0.10	3450.14 (2510.77)	631-9809	0.65	3290.28 (2262.90)	936-8950
0.15	3797.44 (2626.94)	753-9469	0.70	2754.24 (1685.08)	839-7762
0.20	3505.86 (2523.22)	865-9809	0.75	2851.30 (2184.46)	546-9457
0.25	3403.24 (2480.96)	611-9469	0.80	2655.30 (2088.63)	614-8950
0.30	3983.46 (2807.06)	1108-9469	0.85	2616.76 (1681.86)	731-7866
0.35	3537.66 (2690.99)	788-9469	0.90	2676.18 (1765.21)	878-7762
0.40	3595.08 (2641.95)	926-9469	0.95	2095.20 (1552.89)	277-8762
0.45	3740.00 (2622.24)	900-9457	1.00	2192.46 (1106.06)	698-6199
0.50	3004.82 (2374.38)	706-9469	-	-	-

Fitnesses For Coevolution Method #1

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	287.92 (3.345)	280-296	0.55	288.56 (2.714)	282-294
0.05	287.96 (2.953)	284-296	0.60	287.76 (3.037)	282-294
0.10	287.96 (2.698)	284-294	0.65	287.96 (2.383)	282-294
0.15	288.24 (3.166)	284-296	0.70	288.28 (3.200)	282-296
0.20	288.56 (3.324)	282-296	0.75	288.28 (2.713)	284-294
0.25	287.96 (3.212)	280-294	0.80	287.88 (3.057)	282-294
0.30	288.40 (2.623)	284-294	0.85	287.08 (3.129)	280-294
0.35	287.88 (3.210)	282-294	0.90	287.68 (2.477)	282-294
0.40	288.60 (2.307)	284-294	0.95	288.12 (2.725)	284-294
0.45	288.24 (3.115)	282-296	1.00	288.28 (3.418)	282-298
0.50	288.00 (2.561)	282-292	-	-	-

Evaluations Reqd. For Coevolution Method #1

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	3772.16 (2773.32)	773-9771	0.55	2756.62 (1993.11)	793-9743
0.05	2981.12 (2360.51)	490-9786	0.60	2776.46 (2251.32)	567-9388
0.10	3106.10 (2276.07)	882-9703	0.65	3090.82 (2450.13)	269-8953
0.15	2339.66 (1930.12)	440-9719	0.70	2777.12 (2263.51)	850-9183
0.20	2698.22 (2068.91)	837-9559	0.75	3137.98 (2600.86)	1080-9550
0.25	3288.52 (2568.98)	862-9115	0.80	2518.12 (2124.22)	520-9853
0.30	3004.64 (2397.82)	733-9756	0.85	2579.04 (2014.58)	507-9124
0.35	3018.74 (2568.99)	343-9858	0.90	2817.48 (2399.73)	661-9182
0.40	3064.80 (2218.24)	492-9243	0.95	3015.80 (2558.30)	374-9289
0.45	3150.26 (2807.83)	718-9672	1.00	3085.08 (2454.99)	446-9038
0.50	2711.68 (2230.06)	505-9156	-	-	-

Results For Co-evolution Method #2

Measure	Mean (Std Dev)	Range
Fitness	288.16 (2.705)	282-292
Evals. Reqd.	2640.62 (2075.12)	740-9541

Fitnesses For Co-evolution Method #3

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	288.48 (3.384)	282-296	0.55	288.48 (2.579)	284-294
0.05	288.68 (3.240)	282-296	0.60	287.64 (2.583)	282-294
0.10	288.60 (2.807)	284-296	0.65	288.16 (3.196)	282-294
0.15	287.76 (2.930)	280-292	0.70	289.16 (3.126)	284-296
0.20	288.72 (2.960)	282-296	0.75	287.92 (2.799)	282-294
0.25	288.84 (3.022)	282-294	0.80	288.52 (2.648)	284-294
0.30	288.28 (3.200)	280-294	0.85	288.92 (2.568)	282-294
0.35	287.76 (2.550)	282-294	0.90	288.72 (2.960)	284-296
0.40	289.00 (3.181)	280-294	0.95	288.88 (3.327)	282-296
0.45	287.80 (2.375)	282-294	1.00	288.36 (2.762)	282-296
0.50	288.16 (2.708)	282-296	-	-	-

Evaluations Reqd. For Co-evolution Method #3

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	3216.26 (2426.38)	432-9987	0.55	3359.32 (2306.73)	1008-9969
0.05	3042.78 (2210.93)	747-9024	0.60	3310.20 (2622.65)	448-9975
0.10	2785.52 (2213.70)	677-9765	0.65	2925.96 (2355.83)	678-9981
0.15	2831.88 (2189.06)	955-9909	0.70	3286.06 (2384.77)	832-9994
0.20	2867.24 (2358.31)	707-9913	0.75	3301.28 (2189.27)	628-9055
0.25	3108.64 (2379.98)	697-9927	0.80	2435.98 (1591.57)	254-7421
0.30	3236.22 (2588.62)	513-9928	0.85	2914.10 (1574.74)	755-6513
0.35	2783.42 (2375.62)	638-9943	0.90	3186.54 (2646.30)	668-9847
0.40	3098.04 (2476.92)	741-9817	0.95	2678.56 (2084.60)	560-8130
0.45	3064.82 (2413.59)	682-9956	1.00	3025.46 (2448.78)	735-9038
0.50	3254.14 (2490.62)	919-8735	-	-	-

Results For Co-evolution Method #4

Measure	Mean (Std Dev)	Range
Fitness	288.52 (2.648)	282-296
Evals Reqd.	2853.52 (2339.35)	844-9930

Fitnesses For Co-evolution Method #5

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	288.16 (2.525)	282-294	0.55	289.32 (2.611)	282-294
0.05	288.16 (2.989)	282-296	0.60	287.48 (3.195)	282-294
0.10	287.64 (3.064)	282-294	0.65	287.96 (2.842)	284-298
0.15	288.68 (3.140)	282-296	0.70	288.12 (2.812)	282-294
0.20	288.36 (2.456)	284-294	0.75	287.80 (2.835)	282-296
0.25	288.60 (3.130)	284-296	0.80	288.28 (2.562)	284-296
0.30	288.32 (2.753)	282-294	0.85	287.92 (2.827)	282-296
0.35	288.08 (3.123)	282-294	0.90	288.04 (3.310)	280-296
0.40	288.12 (2.978)	282-298	0.95	287.64 (2.674)	282-292
0.45	288.28 (2.912)	282-296	1.00	288.20 (3.779)	280-296
0.50	288.24 (2.818)	280-296	-	-	-

Evaluations Reqd. For Co-evolution Method #5

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	3088.48 (2579.98)	782-9674	0.55	2829.20 (2178.45)	845-9536
0.05	2555.86 (2124.44)	601-9494	0.60	2596.28 (1884.50)	860-7894
0.10	3188.34 (2662.98)	787-9904	0.65	3104.66 (2329.51)	738-9612
0.15	3096.56 (2309.93)	810-9906	0.70	3083.74 (2415.77)	967-9753
0.20	3610.62 (2851.06)	842-9850	0.75	2996.56 (2139.19)	620-9038
0.25	3093.00 (2618.78)	782-9900	0.80	2845.50 (1986.03)	1076-8656
0.30	2903.66 (2337.64)	626-9368	0.85	2695.78 (2173.41)	953-9179
0.35	3101.04 (2449.68)	628-9714	0.90	3615.96 (2718.38)	820-9644
0.40	3376.42 (2367.23)	953-9589	0.95	2770.96 (2279.25)	809-8666
0.45	3594.64 (2855.66)	1015-9603	1.00	3264.32 (2537.66)	1012-9821
0.50	3513.00 (2541.20)	451-9516	-	-	-

Fitnesses For Co-evolution Method #6

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	287.64 (2.674)	284-294	0.55	288.32 (3.233)	282-296
0.05	288.32 (3.055)	282-296	0.60	288.20 (3.000)	284-296
0.10	287.80 (2.720)	282-294	0.65	288.08 (3.148)	282-294
0.15	287.16 (2.436)	280-294	0.70	287.48 (2.707)	282-292
0.20	287.72 (2.592)	282-294	0.75	287.24 (2.853)	282-294
0.25	287.64 (3.116)	282-296	0.80	288.08 (3.369)	282-296
0.30	288.60 (2.891)	282-296	0.85	287.76 (3.063)	282-296
0.35	287.28 (3.193)	282-294	0.90	287.04 (2.945)	282-294
0.40	287.56 (3.106)	282-294	0.95	287.76 (2.983)	280-294
0.45	287.56 (2.836)	282-292	1.00	287.20 (3.046)	280-294
0.50	287.92 (2.621)	284-294	-	-	-

Evaluations Reqd. For Co-evolution Method #6

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std. Dev)	Range
0.00	2795.48 (2290.22)	119-9318	0.55	3209.38 (2511.09)	452-9765
0.05	3078.24 (2627.89)	604-9742	0.60	3115.36 (2346.59)	848-9536
0.10	3712.48 (2909.54)	922-9935	0.65	2903.92 (1931.52)	901-8528
0.15	3208.50 (2843.93)	888-9954	0.70	2972.30 (2297.66)	660-9437
0.20	2796.04 (2238.17)	894-9267	0.75	2785.90 (2311.28)	911-9848
0.25	2769.22 (1965.81)	820-8008	0.80	2421.84 (1693.84)	728-8286
0.30	3604.14 (2460.55)	815-8567	0.85	3133.92 (2400.49)	845-9829
0.35	2595.32 (2088.03)	846-9028	0.90	2099.74 (1564.43)	409-9848
0.40	3036.84 (2478.10)	913-9976	0.95	2985.72 (2503.76)	939-9961
0.45	2792.46 (2283.50)	639-9517	1.00	2613.52 (2179.68)	703-9741
0.50	3004.78 (2556.06)	725-9777	-	-	-

Fitnesses For COBRA with p(Xover)= 0.2 and Varying Gap Size

Gap Size	Mean (Std Dev)	Range	Gap Size	Mean (Std Dev)	Range
200	287.72 (2.592)	284-294	1200	287.92 (2.798)	282-294
400	287.52 (3.336)	282-294	1400	288.08 (2.938)	282-294
600	287.44 (2.624)	280-292	1600	288.32 (2.838)	282-294
800	288.04 (2.756)	282-294	1800	288.40 (2.884)	282-294
1000	287.96 (2.813)	282-294	2000	288.44 (2.920)	282-294

Evaluations Reqd. For COBRA with p(Xover)= 0.2 and Varying Gap Size

Gap Size	Mean (Std Dev)	Range	Gap Size	Mean (Std Dev)	Range
200	2254.68 (1675.05)	689-8867	1200	2118.64 (1599.30)	625-8867
400	2462.80 (1943.33)	625-8950	1400	2232.66 (1762.91)	625-8867
600	2024.86 (1668.47)	585-8867	1600	2219.50 (1566.89)	625-8867
800	2334.04 (1744.37)	625-8867	1800	2360.96 (1852.74)	625-8867
1000	2322.62 (1812.95)	625-8867	2000	2348.54 (1834.14)	625-8867

Fitnesses For COBRA with Gap Size = 1000

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.05	287.64 (2.455)	282-292	0.55	288.44 (2.808)	284-296
0.10	288.04 (3.286)	280-294	0.60	287.92 (2.770)	282-296
0.15	287.64 (2.643)	282-294	0.65	288.04 (2.416)	284-294
0.20	287.96 (2.813)	282-294	0.70	288.44 (2.662)	284-294
0.25	287.44 (2.967)	282-296	0.75	287.64 (2.321)	282-292
0.30	288.36 (2.321)	284-294	0.80	287.00 (2.340)	282-292
0.35	288.24 (2.731)	282-294	0.85	287.00 (2.690)	282-294
0.40	288.00 (2.683)	282-294	0.90	286.72 (2.324)	282-294
0.45	288.48 (1.899)	284-292	0.95	286.28 (2.742)	280-294

Evaluations Reqd. For COBRA with Gap Size = 1000

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.05	2234.30 (1486.19)	854-7762	0.55	3035.04 (2236.72)	680-9823
0.10	2267.14 (1989.88)	631-9398	0.60	3018.04 (2215.87)	797-9457
0.15	2490.96 (1838.96)	540-8659	0.65	3290.28 (2262.90)	936-8950
0.20	2322.62 (1812.95)	625-8867	0.70	2778.62 (1740.39)	839-7762
0.25	2528.10 (2112.13)	611-8950	0.75	2851.30 (2184.46)	546-9457
0.30	2802.46 (2373.16)	730-9457	0.80	2657.52 (2092.26)	614-8950
0.35	3142.36 (2511.04)	788-9457	0.85	2665.74 (1737.18)	731-7762
0.40	2971.72 (2366.22)	496-9283	0.90	2677.66 (1765.05)	878-7762
0.45	3408.86 (2401.81)	900-9457	0.95	1978.92 (1296.03)	277-7762

B.2.4 The Royal Road Problem

Fitnesses For A Standard GA

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	24.64 (8.592)	8-48	0.55	31.84 (9.259)	16-64
0.05	25.76 (9.092)	16-48	0.60	32.32 (7.994)	16-48
0.10	27.36 (7.853)	16-48	0.65	34.88 (8.277)	24-64
0.15	25.44 (6.730)	16-40	0.70	34.72 (8.996)	16-56
0.20	26.88 (9.017)	16-56	0.75	35.36 (7.520)	24-48
0.25	27.04 (7.820)	16-48	0.80	37.12 (6.743)	24-56
0.30	26.40 (6.835)	16-48	0.85	37.44 (6.682)	24-56
0.35	29.60 (7.879)	16-48	0.90	38.88 (7.675)	24-56
0.40	30.24 (7.722)	16-56	0.95	40.64 (7.647)	24-64
0.45	30.40 (8.158)	16-48	1.00	41.28 (9.379)	24-64
0.50	29.12 (7.290)	16-48	-	-	-

Evaluations Reqd. For A Standard GA

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	3443.16 (2556.88)	311-8797	0.55	3170.90 (2030.72)	452-9455
0.05	3650.62 (2627.69)	377-9467	0.60	3138.46 (2076.45)	472-9228
0.10	3648.88 (2616.32)	353-9932	0.65	3529.06 (2118.71)	1255-9418
0.15	3049.28 (2385.13)	396-9303	0.70	3455.26 (2325.48)	423-9549
0.20	3315.50 (2510.36)	267-9114	0.75	3377.84 (2017.25)	1074-9471
0.25	2875.60 (2259.77)	431-9928	0.80	3375.18 (1768.34)	661-9020
0.30	3184.42 (2651.59)	322-9118	0.85	3468.76 (1562.04)	1526-8640
0.35	3951.50 (2613.27)	325-9891	0.90	3508.64 (1784.24)	1084-8488
0.40	3490.22 (2425.02)	422-9909	0.95	3786.42 (1741.13)	1133-9249
0.45	3700.48 (2403.46)	464-9473	1.00	3541.84 (1627.62)	966-8545
0.50	3039.10 (2269.48)	471-9794	-	-	-

Fitnesses For Coevolution Method #1

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	30.08 (9.130)	16-56	0.55	30.08 (8.551)	16-48
0.05	29.76 (8.165)	16-48	0.60	30.08 (8.551)	16-48
0.10	29.44 (9.252)	16-48	0.65	28.32 (8.346)	16-56
0.15	29.76 (9.605)	16-56	0.70	28.48 (7.864)	16-48
0.20	29.76 (8.320)	16-48	0.75	32.48 (7.905)	16-48
0.25	30.56 (8.570)	16-56	0.80	30.08 (9.269)	16-56
0.30	28.32 (7.708)	16-48	0.85	29.28 (8.558)	16-56
0.35	28.48 (6.628)	16-48	0.90	29.12 (7.799)	16-48
0.40	32.80 (9.226)	16-48	0.95	29.60 (7.547)	16-48
0.45	28.80 (7.673)	16-56	1.00	31.20 (7.715)	16-56
0.50	31.68 (8.308)	16-56	-	-	-

Evaluations Reqd. For Coevolution Method #1

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	2919.98 (2167.04)	247-8818	0.55	3163.44 (2257.18)	413-9171
0.05	3405.50 (2567.39)	346-9898	0.60	3678.54 (2503.87)	545-9503
0.10	3189.96 (2470.91)	152-9997	0.65	2563.46 (2107.85)	131-8940
0.15	3248.02 (2861.28)	234-9857	0.70	3301.44 (2580.64)	290-9450
0.20	3164.14 (2243.27)	410-9314	0.75	3666.02 (2369.91)	423-9699
0.25	3513.74 (2591.69)	538-9669	0.80	3734.50 (2886.96)	390-9833
0.30	3046.70 (2292.16)	326-9262	0.85	2950.72 (2062.15)	657-9273
0.35	2945.04 (2179.87)	608-9012	0.90	3572.08 (2653.80)	366-9315
0.40	3383.40 (2258.13)	427-9815	0.95	3169.10 (2266.20)	269-9928
0.45	2759.24 (1961.29)	390-9738	1.00	3860.56 (2454.80)	139-8074
0.50	3612.84 (2379.80)	285-9528	-	-	-

Results For Co-evolution Method #2

Measure	Mean (Std Dev)	Range
Fitness	30.08 (8.400)	16-48
Evals Reqd.	3348.56 (2519.49)	368-9480

Fitnesses For Co-evolution Method #3

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	31.68 (9.186)	16-48	0.55	30.40 (6.788)	16-48
0.05	29.28 (8.706)	16-56	0.60	28.80 (9.191)	16-56
0.10	29.12 (7.962)	16-48	0.65	31.04 (7.613)	16-48
0.15	30.72 (7.733)	16-48	0.70	29.12 (8.430)	16-48
0.20	28.64 (7.853)	16-48	0.75	28.32 (8.794)	16-48
0.25	28.96 (7.655)	16-48	0.80	31.68 (8.758)	16-48
0.30	30.24 (8.806)	16-56	0.85	29.60 (7.376)	16-48
0.35	30.72 (9.102)	16-56	0.90	27.36 (8.630)	16-48
0.40	31.36 (6.758)	16-56	0.95	30.24 (8.358)	16-48
0.45	30.08 (7.076)	16-40	1.00	30.24 (8.046)	16-48
0.50	29.12 (7.633)	16-56	-	-	-

Evaluations Reqd. For Co-evolution Method #3

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	3673.10 (2533.81)	402-8944	0.55	3267.30 (2392.26)	680-9901
0.05	3146.66 (2243.23)	287-9442	0.60	3148.14 (2463.37)	237-9575
0.10	3225.12 (2141.19)	307-7815	0.65	4009.80 (2492.32)	339-9731
0.15	3529.54 (2686.11)	250-9919	0.70	3624.90 (2728.53)	290-9287
0.20	3238.38 (2621.75)	456-9628	0.75	2972.66 (2500.68)	406-9062
0.25	2663.70 (2202.60)	251-8874	0.80	3568.84 (2434.13)	387-9718
0.30	3240.66 (2582.61)	433-9802	0.85	3761.90 (2862.95)	317-9672
0.35	3481.28 (2840.05)	337-9935	0.90	3420.02 (2769.22)	266-9393
0.40	3604.24 (2447.63)	435-9662	0.95	3542.40 (2743.55)	170-9742
0.45	3916.36 (2984.99)	177-9830	1.00	3411.40 (2590.46)	192-9598
0.50	2842.20 (1996.05)	365-8272	-	-	-

Results For Co-evolution Method #4

Measure	Mean (Std Dev)	Range
Fitness	31.04 (8.563)	16-48
Evals Reqd.	3619.46 (2272.37)	434-9012

Fitnesses For Co-evolution Method #5

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	29.28 (7.608)	16-40	0.55	28.16 (6.246)	16-40
0.05	27.84 (7.018)	16-40	0.60	28.00 (7.200)	16-40
0.10	28.16 (7.018)	16-48	0.65	28.16 (8.502)	8-48
0.15	27.84 (7.713)	16-48	0.70	29.44 (7.408)	16-48
0.20	28.16 (7.373)	16-48	0.75	27.04 (7.135)	16-40
0.25	28.48 (7.531)	16-48	0.80	29.76 (8.320)	16-48
0.30	28.80 (7.838)	16-56	0.85	25.76 (6.652)	16-40
0.35	29.60 (8.197)	16-48	0.90	27.20 (6.974)	16-48
0.40	29.28 (6.902)	16-48	0.95	29.60 (6.835)	16-48
0.45	28.16 (8.038)	8-48	1.00	27.20 (6.400)	16-40
0.50	28.96 (6.377)	16-40	-	-	-

Evaluations Reqd. For Co-evolution Method #5

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	2974.50 (1934.98)	263-8539	0.55	3360.12 (2631.22)	343-9882
0.05	3291.12 (2073.91)	375-9962	0.60	3006.00 (2023.90)	383-9452
0.10	2436.08 (1908.43)	162-8721	0.65	2829.40 (1814.77)	303-7218
0.15	2810.38 (2095.50)	195-9587	0.70	3215.18 (2030.48)	446-9387
0.20	2731.28 (1915.33)	310-9664	0.75	2428.20 (1937.12)	334-9288
0.25	3144.00 (2381.54)	330-9797	0.80	3241.28 (2286.47)	366-9912
0.30	2804.52 (1962.50)	364-8145	0.85	2226.36 (1746.09)	200-7252
0.35	3031.38 (2227.21)	309-9972	0.90	2811.74 (2245.40)	269-9760
0.40	2677.38 (1939.56)	417-9126	0.95	3193.68 (2363.10)	227-9503
0.45	2766.36 (2255.43)	248-9957	1.00	2763.46 (2057.50)	268-9114
0.50	3324.46 (2837.14)	242-9720	-	-	-

Fitnesses For Co-evolution Method #6

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	29.44 (6.681)	16-40	0.55	28.96 (8.141)	16-48
0.05	30.56 (7.621)	16-48	0.60	29.92 (8.437)	16-48
0.10	29.92 (7.472)	16-48	0.65	30.56 (9.286)	16-48
0.15	31.20 (10.400)	16-56	0.70	30.72 (6.666)	16-48
0.20	30.88 (8.467)	16-56	0.75	31.20 (8.800)	16-48
0.25	32.48 (7.741)	16-56	0.80	30.08 (8.550)	16-48
0.30	31.20 (8.1971)	16-56	0.85	28.00 (7.200)	16-40
0.35	30.56 (8.863)	16-48	0.90	29.12 (5.935)	16-40
0.40	28.96 (7.820)	16-40	0.95	30.24 (7.721)	16-48
0.45	29.28 (7.438)	16-48	1.00	30.56 (8.109)	16-56
0.50	30.56 (7.788)	16-56	-	-	-

Evaluations Reqd. For Co-evolution Method #6

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	8058.44 (2985.32)	883-9956	0.55	6315.64 (3939.79)	631-9989
0.05	5996.28 (3996.32)	576-9999	0.60	6922.30 (3650.29)	546-9999
0.10	5366.32 (4086.50)	465-9984	0.65	7621.84 (3390.84)	790-9994
0.15	6547.90 (3871.07)	619-9988	0.70	6969.20 (3602.72)	384-9960
0.20	6730.16 (3751.93)	419-9980	0.75	6267.78 (3772.98)	642-9963
0.25	6664.58 (3747.29)	732-9998	0.80	6914.68 (3779.58)	503-9991
0.30	6619.40 (3669.23)	771-9982	0.85	6745.36 (3673.85)	609-9989
0.35	8014.96 (3095.60)	718-9997	0.90	6744.70 (3881.20)	726-9987
0.40	5644.54 (3962.39)	594-9931	0.95	7141.00 (3711.76)	574-9998
0.45	6594.46 (3754.27)	586-9926	1.00	5785.34 (3869.23)	635-9990
0.50	6673.42 (3947.80)	607-9997	-	-	-

Fitnesses For COBRA with p(Xover)= 0.95 and Varying Gap Size

Gap Size	Mean (Std Dev)	Range	Gap Size	Mean (Std Dev)	Range
200	40.00 (8.763)	16-64	1200	40.48 (8.222)	24-64
400	39.36 (9.444)	16-64	1400	40.80 (7.879)	24-64
600	40.80 (8.039)	24-64	1600	40.64 (7.646)	24-64
800	40.48 (8.676)	24-64	1800	40.96 (7.942)	24-64
1000	40.32 (8.152)	24-64	2000	40.64 (7.646)	24-64

Evaluations Reqd. For COBRA with p(Xover)= 0.95 and Varying Gap Size

Gap Size	Mean (Std Dev)	Range	Gap Size	Mean (Std Dev)	Range
200	3887.98 (2054.88)	171-9249	1200	3845.38 (1953.78)	1133-9390
400	3741.48 (1991.50)	729-9249	1400	3826.28 (1824.83)	1133-9249
600	3922.42 (1877.64)	1133-9249	1600	3799.68 (1732.45)	1133-9249
800	3909.62 (1983.24)	1133-9249	1800	3920.18 (1844.91)	1133-9249
1000	4012.26 (2041.04)	1133-9249	2000	3830.12 (1736.22)	1133-9249

Fitnesses For COBRA with Gap Size = 1000

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.05	26.88 (8.728)	16-56	0.55	31.68 (8.757)	16-56
0.10	26.72 (7.607)	16-40	0.60	32.32 (7.993)	16-48
0.15	26.08 (8.128)	16-48	0.65	35.36 (8.172)	24-64
0.20	27.36 (7.853)	16-48	0.70	34.08 (8.881)	16-56
0.25	27.20 (7.838)	16-48	0.75	35.20 (7.332)	24-48
0.30	27.68 (7.368)	16-48	0.80	37.12 (6.742)	24-56
0.35	29.12 (9.433)	16-56	0.85	37.44 (6.681)	24-56
0.40	28.80 (6.596)	16-40	0.90	38.72 (7.565)	24-56
0.45	29.76 (6.604)	16-40	0.95	40.32 (8.152)	24-64

Evaluations Reqd. For COBRA with Gap Size = 1000

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.05	2850.96 (2023.83)	439-9251	0.55	3217.90 (2146.34)	452-9455
0.10	3018.40 (2403.89)	243-8798	0.60	3169.96 (2117.82)	472-9228
0.15	2606.74 (2240.67)	396-9494	0.65	3818.50 (2417.38)	1255-9740
0.20	3669.92 (3047.16)	267-9882	0.70	3334.08 (2305.29)	423-9549
0.25	2938.36 (2495.37)	470-9345	0.75	3367.48 (2001.92)	1074-9471
0.30	3191.70 (2539.58)	322-9753	0.80	3365.68 (1747.21)	661-9020
0.35	3438.16 (2543.06)	325-9537	0.85	3544.18 (1628.19)	1526-8640
0.40	3234.22 (2362.69)	322-9506	0.90	3506.80 (1854.81)	1084-9734
0.45	3318.46 (2237.40)	464-9255	0.95	4012.26 (2041.04)	1133-9249

B.2.5 The Long Path Problem

Fitnesses For A Standard GA

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	47676.46 (3915.30)	24597-49179	0.55	42751.90 (10519.04)	7924-49179
0.05	48024.94 (2907.80)	36123-49179	0.60	43717.72 (10149.51)	2163-49179
0.10	47094.74 (4772.62)	22690-49179	0.65	41613.94 (11844.20)	2920-49179
0.15	47407.36 (4671.75)	23452-49179	0.70	38585.02 (13826.31)	4849-49179
0.20	47312.14 (4764.77)	22305-49179	0.75	37336.56 (14059.46)	1562-49179
0.25	46326.16 (5997.06)	15544-49179	0.80	38942.42 (14246.30)	1712-49179
0.30	46708.88 (5547.38)	18219-49179	0.85	39760.20 (12803.58)	1944-49179
0.35	46503.42 (4823.72)	30649-49179	0.90	32504.72 (15986.98)	2431-49179
0.40	45815.14 (7064.51)	17116-49179	0.95	28387.40 (18939.17)	220-49179
0.45	45095.46 (8276.45)	14621-49179	1.00	19995.22 (20951.85)	29-49179
0.50	47169.46 (3756.33)	33836-49179	-	-	-

Evalutions Reqd. For A Standard GA

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	6484.96 (3600.24)	939-9979	0.55	6462.76 (4016.45)	571-9990
0.05	5615.66 (3810.68)	545-9994	0.60	6308.96 (4057.92)	589-9994
0.10	6421.82 (3713.23)	948-9992	0.65	6641.04 (4057.60)	456-9998
0.15	6217.14 (3857.06)	829-9971	0.70	6735.60 (3752.36)	482-9994
0.20	5129.38 (4088.19)	328-9996	0.75	6568.10 (4023.62)	573-9974
0.25	6282.86 (3857.17)	465-9999	0.80	6354.16 (4177.80)	413-9998
0.30	5978.68 (3806.34)	614-9990	0.85	6571.94 (3970.05)	431-9982
0.35	6431.74 (4082.02)	693-9990	0.90	6752.04 (3865.30)	315-9979
0.40	5779.84 (3776.46)	594-9948	0.95	6501.50 (3618.46)	532-9989
0.45	6018.50 (3960.40)	564-9994	1.00	743.76 (313.81)	393-2039
0.50	5540.08 (3909.23)	609-9984	-	-	-

Fitnesses For Coevolution Method #1

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	45447.40 (7077.03)	18408-49179	0.55	45164.36 (8376.86)	8475-49179
0.05	46259.62 (6049.45)	21492-49179	0.60	46292.24 (7353.22)	15356-49179
0.10	46275.30 (5356.22)	27314-49179	0.65	44193.72 (8013.40)	18235-49179
0.15	43582.96 (10282.70)	7639-49179	0.70	44902.56 (8032.63)	23238-49179
0.20	44561.98 (7961.20)	14622-49179	0.75	46621.08 (5743.78)	24487-49179
0.25	45701.20 (7440.69)	12317-49179	0.80	46050.00 (6143.79)	20909-49179
0.30	44086.54 (8544.17)	18247-49179	0.85	42796.36 (9422.58)	14671-49179
0.35	44047.78 (7895.98)	22924-49179	0.90	43275.96 (10050.17)	5785-49179
0.40	44757.82 (8270.68)	20088-49179	0.95	43754.88 (9104.67)	15295-49179
0.45	44439.18 (9017.82)	15173-49179	1.00	45607.92 (7352.13)	15676-49179
0.50	46649.14 (6831.68)	13851-49179	-	-	-

Evalutions Reqd. For Coevolution Method #1

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	5523.72 (4161.71)	612-9974	0.55	5343.64 (4067.99)	552-9980
0.05	5950.54 (4016.22)	562-9954	0.60	4622.90 (4165.00)	509-9978
0.10	5590.60 (4293.22)	361-9998	0.65	7295.66 (3738.88)	802-9976
0.15	5994.98 (4013.52)	506-9992	0.70	6206.30 (3883.31)	528-9936
0.20	5551.70 (4023.92)	509-9995	0.75	5911.08 (4103.82)	467-9992
0.25	5934.86 (3977.28)	604-9996	0.80	5692.06 (3979.43)	467-9991
0.30	5905.12 (3999.66)	620-9969	0.85	6378.68 (3977.99)	565-9998
0.35	6314.50 (4089.65)	446-9993	0.90	6184.28 (3972.40)	592-9991
0.40	5504.00 (4246.02)	598-9989	0.95	5741.70 (4095.75)	545-9999
0.45	5457.98 (4247.25)	498-9983	1.00	5168.10 (4039.38)	594-9991
0.50	5375.02 (4016.93)	615-9932	-	-	-

Results For Co-evolution Method #2

Measure	Mean (Std Dev)	Range
0.00	45826.16 (6625.29)	23986-49179
0.00	6540.34 (3980.00)	430-9990

Fitnesses For Co-evolution Method #3

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	41923.44 (11868.07)	7705-49179	0.55	40087.10 (14422.04)	4279-49179
0.05	42128.18 (10642.45)	16913-49179	0.60	43055.36 (12598.74)	5127-49179
0.10	39446.66 (13907.26)	503-49179	0.65	42157.24 (10094.68)	9248-49179
0.15	39926.72 (13495.74)	987-49179	0.70	43107.94 (9744.55)	9459-49179
0.20	42632.12 (11019.25)	7541-49179	0.75	43950.42 (8682.63)	15307-49179
0.25	43253.44 (9961.91)	3101-49179	0.80	43491.00 (9736.10)	11549-49179
0.30	40574.50 (12266.69)	2516-49179	0.85	41548.22 (12670.10)	5980-49179
0.35	41760.48 (11276.77)	6195-49179	0.90	43368.12 (10494.68)	12420-49179
0.40	43401.76 (9313.35)	13071-49179	0.95	44509.04 (9753.09)	7320-49179
0.45	38116.12 (16382.16)	606-49179	1.00	41632.80 (11779.56)	4638-49179
0.50	43084.50 (11788.87)	4637-49179	-	-	-

Evaluations Reqd. For Co-evolution Method #3

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	5879.02 (4055.41)	563-9948	0.55	5651.24 (4220.10)	521-9990
0.05	6654.96 (3869.90)	674-9959	0.60	6190.00 (4041.70)	438-9992
0.10	5790.72 (4170.76)	682-9999	0.65	6238.48 (4185.38)	542-9989
0.15	6657.06 (4000.14)	512-9999	0.70	5868.64 (4139.94)	358-9993
0.20	5835.14 (4053.30)	523-9996	0.75	6340.36 (4038.81)	541-9952
0.25	6969.70 (3667.87)	493-9974	0.80	6172.60 (4244.71)	460-9990
0.30	6640.74 (3926.19)	465-9998	0.85	5687.18 (4132.05)	501-9999
0.35	6689.12 (3861.15)	441-9990	0.90	6315.42 (3801.82)	470-9992
0.40	5352.24 (4196.81)	474-9996	0.95	5545.16 (4077.56)	557-9942
0.45	6730.76 (3746.07)	527-9986	1.00	5964.22 (3930.98)	656-9996
0.50	4933.04 (4066.59)	388-9983	-	-	-

Results For Co-evolution Method #4

Measure	Mean (Std Dev)	Range
Fitness	38143.76 (13878.98)	5303-49179
Evals Reqd.	6743.08 (3758.38)	539-9977

Fitnesses For Co-evolution Method #5

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	45982.50 (7026.62)	7908-49179	0.55	43630.76 (9600.41)	15399-49179
0.05	42200.50 (11321.89)	6438-49179	0.60	44168.72 (9193.73)	11895-49179
0.10	46375.48 (5523.65)	23744-49179	0.65	43550.74 (10803.19)	11885-49179
0.15	43369.00 (10445.89)	13848-49179	0.70	41788.14 (11167.68)	4258-49179
0.20	45074.96 (6882.60)	18172-49179	0.75	43959.84 (10025.18)	727-49179
0.25	44773.06 (8844.18)	10971-49179	0.80	43940.72 (10632.86)	1559-49179
0.30	46016.74 (6158.14)	20119-49179	0.85	41499.60 (12152.64)	3848-49179
0.35	45803.74 (6885.35)	24578-49179	0.90	41026.78 (12643.56)	3062-49179
0.40	44627.34 (8383.10)	11908-49179	0.95	40983.12 (14023.05)	115-49179
0.45	45766.52 (5977.22)	19362-49179	1.00	44429.66 (10807.61)	44-49179
0.50	43929.40 (8931.53)	10779-49179	-	-	-

Evaluations Reqd. For Co-evolution Method #5

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	6645.32 (3869.49)	608-9983	0.55	6853.86 (3474.22)	795-9991
0.05	7535.16 (3171.76)	1158-9993	0.60	7366.66 (3569.72)	761-9989
0.10	6187.78 (3900.62)	946-9988	0.65	6342.24 (3839.06)	699-9982
0.15	6629.26 (3763.19)	880-9987	0.70	6707.10 (3664.26)	1014-9997
0.20	6419.78 (3746.68)	832-9995	0.75	6566.46 (3769.34)	643-9999
0.25	7068.62 (3523.87)	1037-9967	0.80	6612.32 (3632.82)	1169-9982
0.30	7162.60 (3567.94)	745-9988	0.85	7242.06 (3344.12)	1114-9969
0.35	5930.06 (3730.73)	832-9970	0.90	7520.32 (3316.01)	1038-9987
0.40	6448.76 (3680.93)	653-9991	0.95	6918.32 (3602.77)	990-9981
0.45	6870.02 (3536.88)	924-9980	1.00	5670.82 (3767.66)	801-9976
0.50	7459.78 (3440.11)	772-9989	-	-	-

Fitnesses For Co-evolution Method #6

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	36464.24 (14513.50)	412-49179	0.55	40500.20 (13525.64)	4643-49179
0.05	39642.62 (13346.06)	3878-49179	0.60	38510.36 (13011.63)	5405-49179
0.10	43663.98 (11508.68)	1374-49179	0.65	36010.24 (15747.76)	3090-49179
0.15	39004.64 (13350.32)	6144-49179	0.70	37120.96 (15938.25)	3292-49179
0.20	43431.62 (9746.51)	12361-49179	0.75	38068.46 (15268.64)	47-49179
0.25	39503.52 (15371.09)	2736-49179	0.80	36007.84 (15897.95)	408-49179
0.30	36770.08 (16337.48)	411-49179	0.85	37829.30 (14174.98)	1767-49179
0.35	34428.08 (14751.44)	1553-49179	0.90	37234.66 (14717.01)	1569-49179
0.40	41744.60 (12344.53)	1754-49179	0.95	37659.04 (14997.05)	219-49179
0.45	41013.26 (12546.478)	6220-49179	1.00	40639.18 (14453.46)	43-49179
0.50	41794.76 (12691.95)	4252-49179	-	-	-

Evaluations Reqd. For Co-evolution Method #6

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	8058.44 (2985.32)	883-9956	0.55	6315.64 (3939.79)	631-9989
0.05	5996.28 (3996.31)	576-9999	0.60	6922.30 (3650.29)	546-9999
0.10	5366.32 (4086.50)	465-9984	0.65	7621.84 (3390.84)	790-9994
0.15	6547.90 (3871.07)	619-9988	0.70	6969.20 (3602.72)	384-9960
0.20	6730.16 (3751.93)	419-9980	0.75	6267.78 (3772.98)	642-9963
0.25	6664.58 (3747.29)	732-9998	0.80	6914.68 (3779.58)	503-9991
0.30	6619.40 (3669.23)	771-9982	0.85	6745.36 (3673.85)	609-9989
0.35	8014.96 (3095.60)	718-9997	0.90	6744.70 (3881.20)	726-9987
0.40	5644.54 (3962.39)	594-9931	0.95	7141.00 (3711.76)	574-9998
0.45	6594.46 (3754.27)	586-9926	1.00	5785.34 (3869.23)	635-9990
0.50	6673.42 (3947.80)	607-9997	-	-	-

Fitnesses For COBRA with p(Xover)= 0.2 and Varying Gap Size

Gap Size	Mean (Std Dev)	Range	Gap Size	Mean (Std Dev)	Range
200	45675.32 (7197.29)	16157-49179	1200	44116.56 (9770.35)	6165-49179
400	43612.54 (8082.77)	21149-49179	1400	42997.64 (10599.27)	6180-49179
600	40568.28 (12736.42)	4533-49179	1600	43320.14 (10315.47)	6220-49179
800	41867.16 (11504.45)	2241-49179	1800	42055.02 (11292.24)	6180-49179
1000	44703.63 (9373.23)	13082-49179	2000	42769.76 (10903.14)	12286-49179

Evaluations Reqd. For COBRA with p(Xover)= 0.2 and Varying Gap Size

Gap Size	Mean (Std Dev)	Range	Gap Size	Mean (Std Dev)	Range
200	5119.62 (4048.14)	480-9988	1200	5043.54 (3985.14)	328-9936
400	6377.62 (3981.05)	328-9999	1400	5305.20 (4142.60)	328-9966
600	6326.22 (3953.45)	328-9974	1600	5566.98 (4185.28)	328-9998
800	6957.82 (3954.90)	328-9998	1800	5554.64 (4198.73)	328-9998
1000	5575.30 (4031.46)	328-9928	2000	5260.20 (4133.31)	328-9966

Fitnesses For COBRA with Gap Size = 1000

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.05	43791.76 (9514.97)	15675-49179	0.55	43712.16 (9283.56)	14619-49179
0.10	41904.40 (11721.01)	583-49179	0.60	44772.96 (8238.58)	18460-49179
0.15	41478.04 (11372.02)	6941-49179	0.65	42681.38 (10313.72)	12313-49179
0.20	44703.60 (9373.23)	13082-49179	0.70	40955.46 (11559.71)	5034-49179
0.25	40239.62 (10826.69)	6007-49179	0.75	41216.48 (11642.01)	11547-49179
0.30	41783.24 (11610.39)	9246-49179	0.80	42889.42 (11292.92)	1712-49179
0.35	41500.90 (11731.79)	7657-49179	0.85	44071.90 (8016.60)	16134-49179
0.40	43683.26 (9961.56)	5454-49179	0.90	41375.58 (11388.03)	12315-49179
0.45	44618.70 (8415.61)	14619-49179	0.95	43031.56 (10779.49)	6078-49179

Evaluations Reqd. For COBRA with Gap Size = 1000

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.05	7437.26 (3125.49)	545-9994	0.55	6358.16 (3995.56)	571-9991
0.10	7339.20 (3242.40)	948-9999	0.60	5981.98 (4020.71)	589-9994
0.15	6748.62 (3762.30)	829-9990	0.65	5992.82 (4053.09)	456-9998
0.20	5575.30 (4031.46)	328-9928	0.70	6589.66 (3819.18)	482-9994
0.25	6891.16 (3753.13)	465-9941	0.75	6500.76 (4102.20)	573-9971
0.30	6368.40 (3833.25)	614-9997	0.80	5866.38 (4002.06)	413-9990
0.35	6238.30 (3953.09)	693-9989	0.85	6685.80 (4054.90)	431-9989
0.40	6424.18 (3876.75)	594-9971	0.90	6532.12 (3900.15)	315-9999
0.45	6279.32 (3987.92)	564-9998	0.95	6447.64 (3699.64)	532-9923

B.3 Results From A Generational GA

B.3.1 The $n/m/P/C_{max}$ Problem

Fitnesses For A Standard GA

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	2448.86 (71.49)	2276-2633	0.55	2450.74 (70.75)	2309-2635
0.05	2444.98 (75.16)	2302-2644	0.60	2449.76 (77.97)	2231-2658
0.10	2452.62 (71.17)	2307-2617	0.65	2443.58 (75.99)	2283-2640
0.15	2450.34 (70.66)	2304-2657	0.70	2447.18 (75.05)	2290-2624
0.20	2451.30 (71.44)	2300-2641	0.75	2454.08 (71.41)	2317-2645
0.25	2445.82 (72.01)	2309-2613	0.80	2446.98 (70.82)	2291-2623
0.30	2450.00 (77.59)	2285-2660	0.85	2446.02 (76.02)	2276-2606
0.35	2447.98 (74.96)	2292-2685	0.90	2446.94 (71.66)	2282-2609
0.40	2450.96 (72.12)	2287-2651	0.95	2441.16 (71.71)	2296-2607
0.45	2449.72 (70.27)	2284-2620	1.00	2456.34 (77.14)	2315-2619
0.50	2446.00 (70.67)	2321-2656	-	-	-

Evaluations Reqd. For A Standard GA

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	7814.00 (1752.14)	4100-9900	0.55	8240.00 (1521.84)	3700-9900
0.05	8212.00 (1485.35)	4500-9900	0.60	8308.00 (1508.75)	4100-9900
0.10	8094.00 (1782.52)	2700-9900	0.65	8236.00 (1285.42)	4300-9900
0.15	7714.00 (1573.02)	4100-9800	0.70	8214.00 (1493.86)	4000-9900
0.20	8066.00 (1550.69)	4400-9900	0.75	8134.00 (1361.12)	3700-9900
0.25	8304.00 (1546.09)	3000-9900	0.80	8364.00 (1283.71)	3500-9900
0.30	8236.00 (1278.71)	4100-9800	0.85	8518.00 (1155.45)	5300-9900
0.35	8674.00 (1302.89)	3600-9900	0.90	8770.00 (1056.46)	4500-9900
0.40	8056.00 (1964.20)	2100-9900	0.95	8914.00 (910.61)	5800-9900
0.45	8006.00 (1363.58)	4700-9900	1.00	8120.00 (1348.18)	5100-9900
0.50	7928.00 (1587.08)	3600-9900	-	-	-

Fitnesses For Co-evolution Method #1

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	2447.70 (73.18)	2261-2647	0.55	2444.30 (67.41)	2302-2660
0.05	2447.24 (66.65)	2291-2633	0.60	2449.18 (75.40)	2305-2651
0.10	2446.66 (71.89)	2296-2634	0.65	2448.64 (70.20)	2316-2665
0.15	2448.82 (77.40)	2299-2660	0.70	2451.68 (72.77)	2321-2637
0.20	2450.82 (74.82)	2275-2641	0.75	2445.18 (74.19)	2295-2629
0.25	2451.44 (72.28)	2303-2633	0.80	2448.14 (78.30)	2265-2631
0.30	2452.24 (76.97)	2298-2644	0.85	2445.20 (71.71)	2309-2616
0.35	2448.70 (73.94)	2281-2624	0.90	2453.06 (70.46)	2322-2655
0.40	2450.26 (76.00)	2309-2686	0.95	2451.06 (74.73)	2277-2639
0.45	2449.96 (69.93)	2307-2610	1.00	2446.94 (74.05)	2304-2634
0.50	2450.34 (67.55)	2303-2601	-	-	-

Evaluations Reqd. For Co-evolution Method #1

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	7900.00 (1875.84)	2600-9900	0.55	8422.00 (1611.74)	3100-9900
0.05	8060.00 (1513.27)	3800-9900	0.60	7880.00 (1780.67)	3400-9800
0.10	8104.00 (1512.60)	2100-9900	0.65	8044.00 (1341.51)	5100-9900
0.15	7988.00 (1740.76)	2600-9900	0.70	8320.00 (1400.14)	5000-9900
0.20	8384.00 (1280.83)	4700-9900	0.75	7998.00 (1420.06)	3800-9900
0.25	7758.00 (1687.84)	3900-9800	0.80	8260.00 (1284.21)	4800-9900
0.30	8348.00 (1375.24)	3300-9800	0.85	8232.00 (1501.52)	3200-9900
0.35	7958.00 (1536.50)	4300-9900	0.90	8134.00 (1584.50)	3800-9900
0.40	8330.00 (1367.36)	4300-9900	0.95	8334.00 (1487.22)	3100-9900
0.45	8208.00 (1389.65)	4800-9900	1.00	8128.00 (1509.04)	4700-9900
0.50	8050.00 (1592.89)	3600-9900	-	-	-

Results For Co-evolution Method #2

Measure	Mean (Std Dev)	Range
Fitness	2443.88 (71.98)	2282-2649
Evals. Reqd.	8244.00 (1532.33)	4300-9900

Fitnesses For Co-evolution Method #3

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	2449.34 (68.80)	2315-2637	0.55	2447.22 (68.73)	2305-2620
0.05	2445.94 (70.03)	2298-2608	0.60	2448.76 (75.64)	2281-2666
0.10	2444.68 (76.34)	2257-2642	0.65	2450.30 (74.41)	2286-2651
0.15	2445.22 (71.65)	2285-2650	0.70	2448.18 (75.49)	2278-2648
0.20	2449.40 (70.99)	2283-2642	0.75	2450.76 (68.51)	2319-2604
0.25	2449.78 (74.16)	2288-2650	0.80	2444.22 (72.56)	2298-2662
0.30	2448.86 (72.26)	2292-2631	0.85	2449.62 (76.95)	2309-2643
0.35	2448.70 (75.18)	2301-2645	0.90	2447.16 (72.36)	2317-2647
0.40	2447.72 (70.12)	2311-2643	0.95	2450.30 (75.03)	2310-2645
0.45	2446.98 (75.41)	2300-2648	1.00	2449.06 (69.67)	2314-2628
0.50	2451.10 (72.29)	2299-2621	-	-	-

Evaluations Reqd. For Co-evolution Method #3

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	7914.00 (1609.85)	3400-9900	0.55	8290.00 (1414.24)	4700-9900
0.05	8204.00 (1273.10)	4900-9900	0.60	8354.00 (1396.02)	4400-9900
0.10	8184.00 (1448.08)	2800-9900	0.65	8004.00 (1746.19)	1800-9900
0.15	8404.00 (1443.46)	3300-9900	0.70	8046.00 (1419.18)	4300-9900
0.20	8476.00 (1351.97)	4400-9900	0.75	8078.00 (1901.18)	3200-9900
0.25	8438.00 (1417.72)	3800-9900	0.80	7982.00 (1512.17)	4300-9900
0.30	8212.00 (1431.45)	4700-9800	0.85	7946.00 (1795.23)	2900-9900
0.35	8430.00 (1158.83)	4800-9900	0.90	8136.00 (1226.50)	5100-9900
0.40	8208.00 (1757.02)	3200-9900	0.95	8414.00 (1484.31)	4400-9900
0.45	8226.00 (1335.33)	4700-9900	1.00	8006.00 (1711.19)	3100-9900
0.50	8366.00 (1095.37)	5200-9900	-	-	-

Results For Co-evolution Method #4

Measure	Mean (Std Dev)	Range
Fitness	2449.32 (73.32)	2289-2638
Evals Reqd.	8272.00 (1777.53)	2900-9900

Fitnesses For COBRA with p(Xover)= 0.65 and Varying Gap Size

Gap Size	Mean (Std Dev)	Range	Gap Size	Mean (Std Dev)	Range
200	2445.36 (75.76)	2283-2640	400-2000	2443.58 (75.99)	2283-2640

Evaluations Reqd. For COBRA with p(Xover)= 0.65 and Varying Gap Size

Gap Size	Mean (Std Dev)	Range	Gap Size	Mean (Std Dev)	Range
200	8098.00 (1395.49)	4300-9900	400-2000	8236.00 (1285.42)	4300-9900

Fitnesses For COBRA with Gap Size = 1000

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.05	2437.76 (75.08)	2282-2618	0.55	2450.74 (70.75)	2309-2635
0.10	2440.58 (73.00)	2285-2597	0.60	2449.76 (77.96)	2312-2658
0.15	2440.72 (74.53)	2268-2642	0.65	2443.58 (75.99)	2283-2640
0.20	2438.40 (72.28)	2300-2620	0.70	2447.18 (75.04)	2290-2624
0.25	2447.36 (70.14)	2291-2640	0.75	2454.08 (71.41)	2317-2645
0.30	2448.64 (75.60)	2297-2637	0.80	2446.98 (70.81)	2291-2623
0.35	2449.46 (71.83)	2297-2674	0.85	2446.02 (76.01)	2276-2606
0.40	2451.40 (71.11)	2286-2626	0.90	2446.94 (71.66)	2282-2609
0.45	2446.40 (70.86)	2289-2618	0.95	2441.16 (71.70)	2296-2607

Evaluations Reqd. For COBRA with Gap Size = 1000

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.05	8766.00 (1123.67)	4400-9900	0.55	8240.00 (1521.84)	3700-9900
0.10	8802.00 (986.00)	5700-9900	0.60	8308.00 (1508.75)	4100-9900
0.15	8556.00 (1227.05)	5300-9900	0.65	8236.00 (1285.41)	4300-9900
0.20	8222.00 (1433.35)	4300-9900	0.70	8214.00 (1493.85)	4000-9900
0.25	8328.00 (1425.76)	4000-9900	0.75	8134.00 (1361.11)	3700-9900
0.30	8436.00 (1477.66)	3900-9900	0.80	8364.00 (1283.70)	3500-9900
0.35	8308.00 (1638.15)	1700-9900	0.85	8518.00 (1155.45)	5300-9900
0.40	7824.00 (1681.13)	3600-9900	0.90	8770.00 (1056.45)	4500-9900
0.45	8354.00 (1359.00)	4400-9900	0.95	8914.00 (910.61)	5800-9900

B.3.2 The Counting Ones Problem

Fitnesses For A Standard GA

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	99.12 (0.79	97-100	0.55	99.86 (0.35)	99-100
0.05	93.38 (1.51)	90-97	0.60	99.96 (0.20)	99-100
0.10	94.78 (1.60)	92-98	0.65	99.92 (0.27)	99-100
0.15	96.12 (1.53)	92-100	0.70	99.86 (0.35)	99-100
0.20	96.96 (1.08)	95-99	0.75	99.94 (0.24)	99-100
0.25	97.88 (1.21)	95-100	0.80	99.96 (0.20)	99-100
0.30	98.64 (1.00)	96-100	0.85	99.86 (0.35)	99-100
0.35	99.12 (0.79)	97-100	0.90	99.70 (0.46)	99-100
0.40	99.30 (0.64)	98-100	0.95	99.50 (0.70)	97-100
0.45	99.66 (0.59)	98-100	1.00	98.48 (1.10)	95-100
0.50	99.80 (0.40)	99-100	-	-	-

Evaluations Reqd. For A Standard GA

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	8654.00 (874.34)	7000-9900	0.55	8404.00 (860.22)	6700-9800
0.05	9146.00 (545.23)	7400-9900	0.60	8184.00 (843.17)	6300-9900
0.10	9190.00 (564.36)	7700-9900	0.65	7798.00 (780.64)	6300-9700
0.15	9142.00 (666.96)	7100-9900	0.70	7906.00 (846.50)	5800-9800
0.20	9228.00 (764.99)	5900-9900	0.75	7836.00 (883.80)	6400-9700
0.25	8944.00 (705.45)	7000-9800	0.80	7714.00 (1025.09)	5100-9800
0.30	8886.00 (592.29)	7700-9900	0.85	7770.00 (1036.38)	5800-9800
0.35	8654.00 (874.35)	7000-9900	0.90	7706.00 (927.66)	5700-9700
0.40	8882.00 (817.48)	7000-9900	0.95	7466.00 (1097.92)	5600-9700
0.45	8692.00 (818.25)	7000-9900	1.00	6848.00 (542.30)	5800-8100
0.50	8404.00 (907.07)	6400-9900	-	-	-

Fitnesses For Coevolution Method #1

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	99.74 (0.48)	98-100	0.55	99.76 (0.43)	99-100
0.05	99.62 (0.56)	98-100	0.60	99.64 (0.66)	97-100
0.10	99.74 (0.48)	98-100	0.65	99.78 (0.46)	98-100
0.15	99.80 (0.44)	98-100	0.70	99.74 (0.55)	98-100
0.20	99.80 (0.40)	99-100	0.75	99.90 (0.30)	99-100
0.25	99.74 (0.48)	98-100	0.80	99.76 (0.47)	98-100
0.30	99.56 (0.61)	98-100	0.85	99.74 (0.48)	98-100
0.35	99.64 (0.56)	98-100	0.90	99.74 (0.55)	98-100
0.40	99.80 (0.40)	99-100	0.95	99.82 (0.38)	99-100
0.45	99.74 (0.52)	98-100	1.00	99.72 (0.57)	98-100
0.50	99.76 (0.47)	98-100	-	-	-

Evaluations Reqd. For Coevolution Method #1

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	8840.00 (768.38)	6900-9900	0.55	8762.00 (721.91)	7200-9900
0.05	8638.00 (716.34)	7300-9900	0.60	8582.00 (835.15)	6200-9900
0.10	8560.00 (699.14)	7300-9900	0.65	8718.00 (745.30)	6700-9900
0.15	8460.00 (775.89)	6900-9800	0.70	8502.00 (898.77)	5100-9700
0.20	8476.00 (884.66)	6100-9900	0.75	8634.00 (875.35)	6500-9900
0.25	8760.00 (852.29)	6400-9900	0.80	8476.00 (922.29)	6600-9900
0.30	8450.00 (948.10)	5700-9900	0.85	8544.00 (810.22)	6400-9800
0.35	8680.00 (758.68)	7200-9800	0.90	8374.00 (838.76)	6500-9900
0.40	8624.00 (884.43)	6900-9900	0.95	8420.00 (833.06)	6700-9900
0.45	8502.00 (900.33)	5800-9900	1.00	8404.00 (818.04)	6500-9900
0.50	8564.00 (884.48)	6700-9900	-	-	-

Results For Co-evolution Method #2

Measure	Mean (Std Dev)	Range
Fitness	99.66 (0.59)	98-100
Evals. Reqd.	8810.00 (794.04)	7100-9900

Fitnesses For Co-evolution Method #3

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	99.74 (0.52)	98-100	0.55	99.68 (0.67)	97-100
0.05	99.74 (0.59)	97-100	0.60	99.50 (1.15)	93-100
0.10	99.78 (0.54)	97-100	0.65	99.64 (0.81)	95-100
0.15	99.72 (0.57)	98-100	0.70	99.62 (0.91)	95-100
0.20	99.80 (0.72)	96-100	0.75	99.68 (0.64)	97-100
0.25	99.60 (0.72)	97-100	0.80	99.62 (0.89)	95-100
0.30	99.68 (0.67)	97-100	0.85	99.74 (0.65)	97-100
0.35	99.76 (0.54)	98-100	0.90	99.48 (0.86)	97-100
0.40	99.78 (0.41)	99-100	0.95	99.76 (0.47)	98-100
0.45	99.62 (0.96)	95-100	1.00	99.60 (0.63)	97-100
0.50	99.82 (0.43)	98-100	-	-	-

Evaluations Reqd. For Co-evolution Method #3

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	8416.00 (885.07)	6700-9800	0.55	8552.00 (984.32)	6500-9900
0.05	8526.00 (805.43)	6500-9800	0.60	8396.00 (920.42)	6600-9900
0.10	8526.00 (884.49)	6700-9900	0.65	8596.00 (830.65)	6700-9900
0.15	8462.00 (910.14)	6400-9900	0.70	8490.00 (899.61)	6500-9900
0.20	8070.00 (910.88)	6200-9800	0.75	8520.00 (940.21)	6100-9900
0.25	8308.00 (906.16)	6300-9900	0.80	8428.00 (852.53)	6800-9900
0.30	8280.00 (828.01)	6400-9900	0.85	8244.00 (966.26)	6100-9900
0.35	8256.00 (890.65)	6200-9900	0.90	8434.00 (837.52)	6600-9900
0.40	8162.00 (977.52)	5800-9700	0.95	8542.00 (977.77)	4800-9900
0.45	8354.00 (781.33)	7000-9900	1.00	8466.00 (915.33)	5900-9900
0.50	8380.00 (1000.00)	6300-9900	-	-	-

Results For Co-evolution Method #4

Measure	Mean (Std Dev)	Range
Fitness	99.58 (0.70)	97-100
Evals Reqd.	8358.00 (977.57)	6000-9900

Fitnesses For Co-evolution Method #5

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	99.28 (0.98)	96-100	0.55	99.22 (1.06)	95-100
0.05	99.46 (0.81)	97-100	0.60	99.26 (1.09)	96-100
0.10	99.12 (1.18)	95-100	0.65	99.10 (1.15)	94-100
0.15	99.08 (0.99)	96-100	0.70	98.92 (1.53)	94-100
0.20	99.16 (1.01)	96-100	0.75	98.80 (1.52)	93-100
0.25	99.10 (1.20)	95-100	0.80	99.24 (1.12)	95-100
0.30	99.16 (0.96)	96-100	0.85	98.92 (1.67)	91-100
0.35	99.06 (1.30)	94-100	0.90	99.20 (1.37)	93-100
0.40	99.40 (0.87)	96-100	0.95	99.00 (1.24)	94-100
0.45	99.52 (0.64)	98-100	1.00	98.68 (1.97)	90-100
0.50	98.92 (1.32)	95-100	-	-	-

Evaluations Reqd. For Co-evolution Method #5

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	8750.00 (899.16)	5500-9800	0.55	8438.00 (857.17)	6900-9900
0.05	8782.00 (889.19)	6500-9900	0.60	8736.00 (797.94)	6900-9800
0.10	8866.00 (805.13)	7000-9900	0.65	8714.00 (926.50)	6300-9900
0.15	8842.00 (709.67)	6300-9900	0.70	8790.00 (815.16)	6800-9900
0.20	8850.00 (763.48)	6900-9900	0.75	8778.00 (692.03)	6700-9900
0.25	8852.00 (866.55)	6900-9900	0.80	8518.00 (932.02)	6800-9900
0.30	8940.00 (670.82)	7200-9900	0.85	8646.00 (904.03)	6500-9900
0.35	8692.00 (759.43)	6700-9900	0.90	8602.00 (959.06)	6300-9800
0.40	8720.00 (832.82)	6200-9900	0.95	8630.00 (992.82)	6100-9900
0.45	8548.00 (781.08)	7000-9900	1.00	8472.00 (1187.27)	5500-9900
0.50	8936.00 (740.47)	6900-9900	-	-	-

Fitnesses For Co-evolution Method #6

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	98.66 (1.50)	95-100	0.55	98.46 (1.85)	93-100
0.05	98.54 (1.49)	94-100	0.60	98.54 (2.14)	90-100
0.10	98.62 (1.77)	93-100	0.65	98.62 (1.99)	91-100
0.15	98.72 (1.49)	94-100	0.70	98.30 (2.16)	89-100
0.20	98.80 (1.60)	94-100	0.75	98.66 (1.50)	94-100
0.25	98.94 (1.50)	93-100	0.80	98.26 (1.84)	94-100
0.30	98.68 (1.58)	93-100	0.85	98.44 (1.79)	92-100
0.35	98.44 (1.49)	95-100	0.90	98.22 (2.65)	87-100
0.40	98.72 (1.92)	92-100	0.95	97.72 (3.00)	87-100
0.45	98.94 (1.75)	92-100	1.00	98.10 (2.79)	87-100
0.50	98.52 (2.24)	90-100	-	-	-

Evaluations Reqd. For Co-evolution Method #6

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	8824.00 (979.09)	6700-9900	0.55	8986.00 (744.31)	7400-9900
0.05	9072.00 (787.41)	6300-9900	0.60	8888.00 (914.90)	5800-9900
0.10	9018.00 (745.30)	6800-9900	0.65	8690.00 (730.82)	6800-9800
0.15	8622.00 (796.31)	6000-9900	0.70	8954.00 (713.92)	7000-9800
0.20	9038.00 (714.95)	6900-9900	0.75	9050.00 (674.16)	6900-9900
0.25	8944.00 (633.77)	6600-9900	0.80	8836.00 (918.42)	6100-9900
0.30	8986.00 (635.92)	7400-9900	0.85	9102.00 (736.61)	6300-9900
0.35	9004.00 (651.75)	6700-9900	0.90	8702.00 (928.98)	6600-9900
0.40	8940.00 (789.18)	7000-9900	0.95	8956.00 (781.83)	6500-9900
0.45	8918.00 (660.81)	7200-9900	1.00	8776.00 (1061.05)	6000-9900
0.50	8958.00 (762.12)	6200-9900	-	-	-

Fitnesses For COBRA with p(Xover)= 0.8 and Varying Gap Size

Gap Size	Mean (Std Dev)	Range
ALL	99.96 (0.19)	99-100

Evaluations Reqd. For COBRA with p(Xover)= 0.8 and Varying Gap Size

Gap Size	Mean (Std Dev)	Range
ALL	7714.00 (1025.08)	1050804-7714

Fitnesses For COBRA with Gap Size = 1000

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.05	98.36 (1.43)	95-100	0.55	99.86 (0.34)	99-100
0.10	99.04 (0.99)	96-100	0.60	99.96 (0.19)	99-100
0.15	99.78 (0.41)	99-100	0.65	99.92 (0.27)	99-100
0.20	99.76 (0.47)	98-100	0.70	99.86 (0.34)	99-100
0.25	99.92 (0.27)	99-100	0.75	99.94 (0.23)	99-100
0.30	99.90 (0.30)	99-100	0.80	99.96 (0.19)	99-100
0.35	99.90 (0.30)	99-100	0.85	99.86 (0.34)	99-100
0.40	99.94 (0.23)	99-100	0.90	99.70 (0.45)	99-100
0.45	99.82 (0.43)	98-100	0.95	99.50 (0.70)	97-100

Evaluations Reqd. For COBRA with Gap Size = 1000

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.05	8486.00 (1031.89)	6200-9900	0.55	8404.00 (860.22)	6700-9800
0.10	8382.00 (1035.89)	5500-9900	0.60	8184.00 (843.17)	6300-9900
0.15	8464.00 (941.64)	6500-9900	0.65	7798.00 (780.63)	6300-9700
0.20	7808.00 (859.49)	6100-9700	0.70	7906.00 (846.50)	5800-9800
0.25	8022.00 (933.65)	6100-9600	0.75	7836.00 (883.80)	6400-9700
0.30	7896.00 (934.44)	5700-9800	0.80	7714.00 (1025.08)	5100-9800
0.35	8050.00 (749.46)	6700-9800	0.85	7770.00 (1036.38)	5800-9800
0.40	7982.00 (832.75)	6000-9800	0.90	7706.00 (927.66)	5700-9700
0.45	8446.00 (914.15)	6500-9900	0.95	7466.00 (1097.92)	5600-9700

B.3.3 The Order-3 Deceptive Problem

Fitnesses For A Standard GA

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	288.96 (2.918)	284-294	0.55	287.44 (2.940)	282-294
0.05	289.68 (2.411)	282-296	0.60	286.36 (2.066)	282-292
0.10	289.36 (2.840)	284-294	0.65	287.12 (2.503)	282-294
0.15	289.60 (2.465)	284-296	0.70	286.76 (2.557)	282-294
0.20	289.04 (2.049)	284-294	0.75	286.24 (2.702)	282-292
0.25	288.40 (2.800)	282-294	0.80	286.00 (3.046)	282-296
0.30	288.44 (3.053)	282-294	0.85	286.04 (2.697)	280-292
0.35	288.76 (3.069)	284-296	0.90	285.72 (2.653)	282-292
0.40	288.36 (2.762)	284-296	0.95	284.92 (2.504)	280-292
0.45	288.32 (2.444)	284-294	1.00	284.48 (2.699)	278-290
0.50	286.88 (2.566)	284-294	-	-	-

Evaluations Reqd. For A Standard GA

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	5670.00 (2305.14)	1700-9900	0.55	5928.00 (2083.94)	1200-9300
0.05	5960.00 (1964.68)	2100-9700	0.60	6378.00 (1820.25)	1900-9800
0.10	6038.00 (2330.99)	1800-9600	0.65	5446.00 (1671.31)	2600-9200
0.15	6212.00 (2234.34)	1300-9700	0.70	5584.00 (1814.53)	1900-9900
0.20	6188.00 (2309.17)	1400-9800	0.75	5484.00 (2114.65)	400-9900
0.25	5848.00 (1810.21)	1900-9900	0.80	5600.00 (2097.33)	1800-9700
0.30	5680.00 (1706.58)	2400-9600	0.85	5050.00 (1688.69)	1600-9200
0.35	6166.00 (2172.79)	1000-9900	0.90	4484.00 (1883.86)	1300-9100
0.40	6490.00 (1984.87)	2900-9900	0.95	4598.00 (1537.07)	1900-9700
0.45	6100.00 (2125.93)	2300-9900	1.00	4032.00 (1430.16)	1200-9400
0.50	5856.00 (1906.95)	1100-9600	-	-	-

Fitnesses For Coevolution Method #1

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	287.16 (2.831)	282-294	0.55	288.04 (2.638)	284-294
0.05	287.96 (2.449)	284-294	0.60	288.28 (2.800)	282-294
0.10	287.28 (2.254)	282-292	0.65	287.32 (2.641)	282-294
0.15	287.76 (3.010)	282-296	0.70	287.96 (2.668)	282-296
0.20	287.32 (2.846)	282-294	0.75	287.92 (2.910)	282-296
0.25	287.28 (2.735)	282-292	0.80	288.28 (2.498)	282-294
0.30	287.40 (2.272)	284-292	0.85	287.96 (3.006)	282-296
0.35	287.76 (2.387)	282-294	0.90	287.24 (2.680)	282-292
0.40	287.48 (2.879)	282-294	0.95	287.48 (2.737)	282-294
0.45	287.84 (3.120)	282-296	1.00	287.60 (2.993)	282-294
0.50	287.96 (2.638)	282-294	-	-	-

Evaluations Reqd. For Coevolution Method #1

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	6494.00 (1899.30)	2300-9800	0.55	6928.00 (1916.77)	2200-9900
0.05	6634.00 (1977.93)	2400-9900	0.60	6912.00 (1988.23)	1900-9900
0.10	6026.00 (2298.93)	1900-9800	0.65	6248.00 (2048.63)	2100-9700
0.15	6182.00 (2023.72)	1500-9700	0.70	6668.00 (2246.72)	1800-9900
0.20	6730.00 (2045.99)	2100-9900	0.75	6402.00 (1906.46)	2400-9900
0.25	6672.00 (2124.33)	1600-9900	0.80	6452.00 (2294.10)	2600-9900
0.30	6458.00 (2327.67)	2200-9800	0.85	6538.00 (2187.68)	1500-9600
0.35	6200.00 (2271.12)	1500-9900	0.90	6624.00 (2047.88)	2300-9800
0.40	5692.00 (2158.50)	1200-9800	0.95	5708.00 (2041.74)	2100-9000
0.45	6386.00 (2177.43)	2100-9900	1.00	6586.00 (1766.81)	3200-9900
0.50	6428.00 (2219.82)	1900-9800	-	-	-

Results For Co-evolution Method #2

Measure	Mean (Std Dev)	Range
Fitness	287.76 (3.063)	282-29
Evals. Reqd.	6536.00 (2372.74)	1100-9900

Fitnesses For Co-evolution Method #3

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	287.12 (2.688)	282-294	0.55	287.72 (3.072)	282-294
0.05	288.08 (3.199)	282-296	0.60	287.88 (2.923)	282-298
0.10	288.32 (2.839)	282-298	0.65	287.88 (2.277)	282-292
0.15	286.88 (3.128)	282-294	0.70	288.20 (2.778)	280-294
0.20	287.84 (2.648)	282-294	0.75	288.12 (2.604)	284-294
0.25	288.12 (2.840)	282-294	0.80	288.24 (3.241)	282-296
0.30	287.84 (2.880)	282-294	0.85	287.76 (2.518)	282-292
0.35	287.80 (3.079)	282-294	0.90	287.88 (2.169)	282-294
0.40	288.76 (3.368)	282-296	0.95	287.32 (2.731)	282-294
0.45	287.56 (2.721)	282-292	1.00	287.80 (2.891)	282-294
0.50	288.56 (2.913)	282-294	-	-	-

Evaluations Reqd. For Co-evolution Method #3

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	5874.00 (2330.64)	1800-9800	0.55	6450.00 (2006.31)	1300-9800
0.05	6622.00 (2047.36)	1800-9900	0.60	6184.00 (2171.12)	1500-9900
0.10	6248.00 (1911.56)	2300-9600	0.65	6464.00 (2269.08)	1600-9700
0.15	5814.00 (2274.81)	1000-9900	0.70	6440.00 (2343.33)	1700-9600
0.20	6792.00 (1852.66)	3000-9900	0.75	6356.00 (2026.34)	2800-9900
0.25	5894.00 (2083.30)	1400-9900	0.80	6070.00 (2037.77)	1600-9900
0.30	6196.00 (2186.77)	1400-9900	0.85	6258.00 (2384.45)	1300-9900
0.35	5992.00 (2433.58)	700-9700	0.90	6562.00 (2150.80)	1100-9800
0.40	5868.00 (2116.93)	1600-9600	0.95	6046.00 (2095.15)	1600-9800
0.45	6668.00 (1989.21)	2500-9700	1.00	6568.00 (1978.93)	2100-9800
0.50	6456.00 (2148.40)	1300-9900	-	-	-

Results For Co-evolution Method #4

Measure	Mean (Std Dev)	Range
Fitness	287.72 (2.367)	284-292
Evals Reqd.	5918.00 (2161.08)	1500-9400

Fitnesses For Co-evolution Method #5

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	288.88 (2.971)	282-296	0.55	288.64 (2.605)	284-294
0.05	288.52 (2.906)	284-296	0.60	288.08 (3.708)	278-296
0.10	288.12 (3.380)	282-298	0.65	288.52 (3.093)	280-294
0.15	289.28 (2.960)	282-296	0.70	288.80 (2.653)	282-294
0.20	288.88 (2.916)	282-294	0.75	289.00 (3.079)	282-296
0.25	288.76 (3.095)	282-296	0.80	288.08 (3.346)	282-294
0.30	289.08 (3.230)	280-296	0.85	288.88 (3.128)	282-296
0.35	289.12 (2.832)	282-294	0.90	288.44 (3.281)	282-294
0.40	288.56 (2.913)	284-294	0.95	288.76 (2.990)	284-296
0.45	288.92 (3.051)	284-296	1.00	288.24 (2.902)	282-296
0.50	288.52 (2.647)	282-294	-	-	-

Evaluations Reqd. For Co-evolution Method #5

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	5714.00 (2129.41)	1400-9800	0.55	5784.00 (2160.49)	1400-9700
0.05	6064.00 (2362.09)	300-9500	0.60	5904.00 (2049.77)	1800-9900
0.10	5820.00 (2635.68)	700-9800	0.65	5726.00 (2262.99)	1800-9900
0.15	6272.00 (1741.03)	2400-9800	0.70	5746.00 (1990.59)	2300-9500
0.20	5826.00 (2026.01)	2400-9900	0.75	5932.00 (1706.86)	2600-9800
0.25	6118.00 (2358.45)	1500-9900	0.80	5628.00 (1885.74)	800-9900
0.30	6560.00 (2044.40)	2200-9800	0.85	5318.00 (1824.36)	2700-9500
0.35	6026.00 (2325.66)	1300-9600	0.90	5542.00 (1893.04)	1400-9500
0.40	5372.00 (2173.20)	500-9900	0.95	5468.00 (2073.49)	1300-9600
0.45	5874.00 (2287.34)	1900-9800	1.00	5306.00 (1917.95)	2100-9900
0.50	5852.00 (2321.39)	800-9900	-	-	-

Fitnesses For Co-evolution Method #6

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	287.64 (3.480)	280-296	0.55	287.72 (3.175)	282-296
0.05	287.20 (2.857)	282-292	0.60	287.00 (2.864)	282-294
0.10	287.60 (3.019)	282-294	0.65	286.80 (2.939)	282-292
0.15	288.20 (3.376)	280-294	0.70	287.84 (3.270)	282-298
0.20	287.76 (3.063)	282-296	0.75	287.44 (2.968)	282-294
0.25	287.36 (3.160)	280-296	0.80	287.04 (2.660)	282-294
0.30	288.16 (3.120)	282-296	0.85	287.20 (3.274)	278-294
0.35	286.92 (2.972)	282-294	0.90	287.72 (3.441)	280-296
0.40	288.04 (3.452)	282-296	0.95	287.04 (3.423)	280-296
0.45	286.96 (3.180)	280-296	1.00	286.92 (2.568)	282-292
0.50	286.64 (3.084)	280-294	-	-	-

Evaluations Reqd. For Co-evolution Method #6

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	5722.00 (1984.87)	2100-9600	0.55	6138.00 (2249.88)	1900-9900
0.05	6036.00 (2243.45)	800-9800	0.60	5410.00 (2296.71)	1000-9800
0.10	5666.00 (2036.92)	1100-9900	0.65	5228.00 (2233.20)	1500-9500
0.15	6150.00 (2460.42)	1100-9900	0.70	6048.00 (2330.77)	1100-9900
0.20	5040.00 (1805.99)	1800-9200	0.75	5230.00 (2290.96)	1300-9700
0.25	5850.00 (1981.43)	2200-9900	0.80	5600.00 (2099.90)	1100-9700
0.30	5876.00 (2213.10)	1200-9900	0.85	5588.00 (2250.75)	1600-9900
0.35	5078.00 (2116.62)	600-8800	0.90	5540.00 (1823.62)	2500-9900
0.40	5882.00 (2421.13)	1400-9900	0.95	5266.00 (2152.35)	800-9400
0.45	6156.00 (2485.49)	1100-9900	1.00	5582.00 (2124.02)	2000-9900
0.50	5692.00 (2125.82)	2500-9500	-	-	-

Fitnesses For COBRA with p(Xover)= 0.05 and Varying Gap Size

Gap Size	Mean (Std Dev)	Range	Gap Size	Mean (Std Dev)	Range
200	285.44 (2.714)	280-292	1200	286.56 (2.913)	282-292
400	285.16 (2.402)	280-290	1400	286.84 (3.100)	282-292
600	286.08 (2.591)	280-294	1600	286.88 (2.997)	280-294
800	286.24 (2.102)	282-292	1800	287.24 (2.881)	282-292
1000	286.24 (2.549)	280-294	2000	287.44 (3.125)	280-294

Evaluations Reqd. For COBRA with p(Xover)= 0.05 and Varying Gap Size

Gap Size	Mean (Std Dev)	Range	Gap Size	Mean (Std Dev)	Range
200	4322.00 (1659.19)	900-8800	1200	4314.00 (1661.56)	1400-9400
400	4522.00 (1586.47)	1700-8900	1400	4160.00 (1588.45)	1300-9200
600	4194.00 (1500.18)	800-8200	1600	4534.00 (1809.26)	1300-9700
800	4140.00 (1636.94)	1400-9000	1800	4204.00 (1344.76)	1800-7900
1000	4244.00 (1956.74)	900-9300	2000	4488.00 (1743.40)	1800-9800

Fitnesses For COBRA with Gap Size = 1000

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.05	286.24 (2.549)	280-294	0.55	287.44 (2.940)	282-294
0.10	286.28 (3.098)	280-294	0.60	286.36 (2.066)	282-292
0.15	286.76 (2.619)	282-292	0.65	287.12 (2.503)	282-294
0.20	287.08 (2.862)	282-292	0.70	286.76 (2.557)	282-294
0.25	286.52 (2.906)	280-292	0.75	286.24 (2.702)	282-292
0.30	286.84 (2.859)	282-294	0.80	286.00 (3.046)	282-296
0.35	287.12 (2.861)	282-294	0.85	286.04 (2.697)	280-292
0.40	287.00 (2.505)	280-292	0.90	285.72 (2.653)	282-292
0.45	288.04 (3.310)	282-296	0.95	284.92 (2.504)	280-292

Evaluations Reqd. For COBRA with Gap Size = 1000

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.05	4244.00 (1956.74)	900-9300	0.55	5928.00 (2083.94)	1200-9300
0.10	4568.00 (2005.23)	900-9500	0.60	6378.00 (1820.25)	1900-9800
0.15	4948.00 (2279.84)	1000-9600	0.65	5446.00 (1671.31)	2600-9200
0.20	5398.00 (2056.06)	1200-9300	0.70	5584.00 (1814.53)	1900-9900
0.25	5100.00 (1740.22)	1600-9000	0.75	5484.00 (2114.64)	400-9900
0.30	5972.00 (1809.20)	2100-9500	0.80	5600.00 (2097.33)	1800-9700
0.35	5796.00 (2206.71)	1400-9600	0.85	5050.00 (1688.69)	1600-9200
0.40	6294.00 (2411.50)	1700-9800	0.90	4484.00 (1883.86)	1300-9100
0.45	6412.00 (1973.99)	2600-9600	0.95	4598.00 (1537.07)	1900-9700

B.3.4 The Royal Road Problem

Fitnesses For A Standard GA

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	30.56 (5.920)	16-48	0.55	34.88 (7.112)	24-56
0.05	30.72 (5.625)	16-40	0.60	34.72 (6.714)	16-48
0.10	32.96 (5.909)	24-48	0.65	34.24 (7.680)	16-56
0.15	33.60 (5.306)	24-40	0.70	34.08 (5.499)	24-48
0.20	32.96 (5.219)	24-48	0.75	33.44 (6.536)	24-48
0.25	34.24 (5.316)	24-48	0.80	31.04 (6.327)	16-48
0.30	33.28 (5.393)	24-48	0.85	28.80 (5.986)	16-40
0.35	35.52 (6.020)	24-48	0.90	29.28 (7.937)	8-48
0.40	34.56 (6.681)	16-48	0.95	25.28 (7.219)	16-48
0.45	34.24 (5.551)	24-48	1.00	22.72 (6.471)	8-40
0.50	34.40 (4.866)	24-40	-	-	-

Evaluations Reqd. For A Standard GA

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	6800.00 (2261.95)	800-9900	0.55	7620.00 (1761.59)	1500-9900
0.05	6784.00 (2342.42)	1000-9900	0.60	7578.00 (1632.82)	3600-9800
0.10	6880.00 (1974.23)	3300-9800	0.65	7264.00 (1701.50)	2900-9900
0.15	7250.00 (1770.11)	3900-9900	0.70	7636.00 (1571.21)	3800-9900
0.20	6916.00 (1781.94)	3100-9900	0.75	7520.00 (1419.15)	4100-9900
0.25	7426.00 (1674.61)	2400-9900	0.80	7216.00 (1718.18)	2900-9800
0.30	7274.00 (1687.34)	3400-9900	0.85	6608.00 (1937.92)	1400-9900
0.35	7804.00 (1389.81)	4700-9900	0.90	7088.00 (2059.77)	500-9900
0.40	7298.00 (1641.03)	2600-9900	0.95	5626.00 (2253.51)	600-9700
0.45	7650.00 (1704.96)	3200-9900	1.00	4836.00 (2375.69)	500-9900
0.50	7642.00 (1680.96)	3700-9700	-	-	-

Fitnesses For Coevolution Method #1

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	33.12 (6.790)	24-48	0.55	32.80 (5.824)	24-48
0.05	33.44 (4.716)	24-40	0.60	32.96 (5.219)	24-40
0.10	33.92 (6.311)	24-48	0.65	34.72 (6.116)	24-48
0.15	34.24 (5.995)	24-48	0.70	33.28 (5.848)	24-48
0.20	34.08 (5.945)	16-48	0.75	33.28 (5.149)	24-48
0.25	33.92 (5.440)	24-48	0.80	34.08 (7.298)	16-48
0.30	33.12 (5.771)	24-48	0.85	34.24 (5.551)	24-48
0.35	33.12 (5.309)	24-48	0.90	32.80 (6.645)	16-48
0.40	33.76 (5.832)	24-48	0.95	33.76 (5.832)	16-48
0.45	33.12 (5.544)	24-40	1.00	35.36 (6.005)	24-48
0.50	35.04 (5.744)	24-48	-	-	-

Evaluations Reqd. For Coevolution Method #1

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	7212.00 (1746.72)	2700-9900	0.55	7386.00 (1634.99)	2900-9700
0.05	7554.00 (1506.67)	4000-9900	0.60	7636.00 (1590.57)	3200-9900
0.10	7506.00 (1555.43)	4200-9900	0.65	7324.00 (1764.49)	3600-9900
0.15	7362.00 (1781.34)	2100-9700	0.70	7290.00 (1656.29)	3100-9900
0.20	7296.00 (1607.35)	3700-9900	0.75	7374.00 (1576.42)	4100-9900
0.25	7496.00 (1394.12)	4800-9900	0.80	7204.00 (1875.94)	2000-9900
0.30	7220.00 (1524.99)	2000-9500	0.85	7380.00 (1653.24)	3300-9900
0.35	7392.00 (1456.13)	4400-9900	0.90	7228.00 (1702.71)	3500-9800
0.40	6856.00 (1833.05)	2600-9900	0.95	7706.00 (1628.55)	3100-9900
0.45	6794.00 (1805.37)	2900-9800	1.00	7732.00 (1325.66)	4700-9900
0.50	7458.00 (1596.50)	2800-9900	-	-	-

Results For Co-evolution Method #2

Measure	Mean (Std Dev)	Range
Fitness	34.72 (6.714)	16-48
Evals Reqd.	7430.00 (1672.39)	1600-9700

Fitnesses For Co-evolution Method #3

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	32.32 (6.589)	24-48	0.55	32.96 (6.121)	16-48
0.05	32.80 (7.714)	8-56	0.60	33.44 (5.231)	24-40
0.10	33.12 (5.544)	24-48	0.65	33.44 (6.729)	16-48
0.15	33.60 (5.306)	24-48	0.70	32.16 (6.090)	16-48
0.20	32.96 (5.219)	24-48	0.75	32.80 (6.039)	16-48
0.25	33.76 (6.256)	24-48	0.80	32.80 (6.248)	24-48
0.30	32.48 (6.073)	24-48	0.85	32.32 (7.148)	16-48
0.35	32.48 (5.405)	16-40	0.90	34.08 (6.361)	16-48
0.40	32.00 (6.788)	24-48	0.95	35.20 (5.768)	24-48
0.45	33.76 (6.652)	16-48	1.00	32.96 (5.688)	24-40
0.50	32.32 (6.589)	24-48	-	-	

Evaluations Reqd. For Co-evolution Method #3

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	6964.00 (2026.50)	2500-9900	0.55	7188.00 (1793.05)	1300-9600
0.05	7206.00 (1947.75)	2300-9900	0.60	7384.00 (1715.97)	4100-9900
0.10	7356.00 (1841.64)	1600-9800	0.65	7196.00 (1814.49)	1800-9800
0.15	7642.00 (1461.10)	4600-9900	0.70	7228.00 (1921.77)	2000-9800
0.20	7138.00 (1580.87)	1800-9200	0.75	7172.00 (1873.50)	2700-9900
0.25	7600.00 (1553.31)	3000-9900	0.80	7378.00 (1619.29)	2400-9700
0.30	7208.00 (1719.28)	2200-9700	0.85	7354.00 (1948.46)	2600-9900
0.35	7128.00 (1924.99)	2200-9900	0.90	7298.00 (1841.24)	200-9900
0.40	6898.00 (1907.40)	2200-9900	0.95	7364.00 (1849.51)	1500-9900
0.45	7258.00 (2109.79)	300-9900	1.00	6978.00 (2039.44)	1400-9800
0.50	7000.00 (1905.15)	1700-9900	-	-	-

Results For Co-evolution Method #4

Measure	Mean (Std Dev)	Range
Fitness	33.60 (6.400)	16-48
Evals Reqd.	7656.00 (1500.02)	3100-9900

Fitnesses For Co-evolution Method #5

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	29.12 (7.633)	16-48	0.55	30.24 (7.721)	16-48
0.05	28.48 (5.579)	16-40	0.60	30.40 (8.158)	16-48
0.10	30.08 (6.311)	16-48	0.65	29.44 (7.233)	16-48
0.15	29.92 (6.560)	16-40	0.70	29.12 (7.290)	8-40
0.20	29.44 (6.080)	16-40	0.75	29.76 (7.680)	16-40
0.25	28.64 (5.788)	16-40	0.80	29.44 (6.870)	16-48
0.30	30.56 (6.917)	16-48	0.85	29.60 (6.645)	16-40
0.35	29.76 (6.796)	16-48	0.90	31.20 (6.835)	16-48
0.40	31.36 (6.163)	16-48	0.95	28.80 (6.400)	16-40
0.45	30.72 (7.394)	16-40	1.00	30.72 (7.733)	16-56
0.50	30.56 (6.536)	16-48	-	-	-

Evaluations Reqd. For Co-evolution Method #5

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	6118.00 (2882.75)	400-9900	0.55	6736.00 (1988.34)	1100-9700
0.05	6796.00 (2167.94)	500-9800	0.60	6794.00 (2468.39)	300-9900
0.10	7058.00 (2228.46)	1900-9900	0.65	6260.00 (2258.05)	1000-9900
0.15	6636.00 (2149.39)	1000-9900	0.70	7124.00 (2494.35)	800-9800
0.20	6886.00 (2350.83)	700-9900	0.75	6384.00 (2503.47)	900-9800
0.25	6352.00 (2536.70)	400-9900	0.80	6702.00 (2266.49)	1400-9900
0.30	6710.00 (2634.25)	500-9900	0.85	6566.00 (2386.68)	1100-9800
0.35	6028.00 (2350.74)	1300-9900	0.90	6734.00 (2139.30)	1400-9900
0.40	6638.00 (2404.90)	900-9900	0.95	6086.00 (2505.03)	300-9900
0.45	6638.00 (2482.17)	500-9900	1.00	6398.00 (2504.19)	400-9900
0.50	7090.00 (1707.54)	3200-9600	-	-	-

Fitnesses For Co-evolution Method #6

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	28.64 (6.614)	16-48	0.55	27.84 (7.198)	16-40
0.05	28.32 (7.012)	16-40	0.60	28.48 (8.025)	8-48
0.10	29.92 (6.939)	16-40	0.65	29.92 (7.807)	16-48
0.15	27.36 (7.520)	16-40	0.70	28.00 (7.019)	16-40
0.20	30.72 (6.063)	24-40	0.75	28.64 (6.614)	16-48
0.25	30.40 (7.155)	16-48	0.80	27.20 (6.197)	16-40
0.30	26.88 (5.935)	16-40	0.85	28.00 (7.376)	8-48
0.35	27.20 (7.155)	16-40	0.90	25.60 (7.155)	16-40
0.40	27.20 (5.543)	16-40	0.95	29.60 (7.200)	16-48
0.45	28.00 (6.039)	16-40	1.00	27.84 (6.448)	16-40
0.50	28.80 (8.466)	16-48	-	-	-

Evaluations Reqd. For Co-evolution Method #6

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	6432.00 (2590.48)	300-9900	0.55	5818.00 (2734.79)	900-9900
0.05	5992.00 (2370.47)	400-9600	0.60	6468.00 (2681.97)	1100-9900
0.10	5660.00 (2462.76)	200-9900	0.65	7178.00 (2151.39)	1300-9900
0.15	6148.00 (2617.11)	700-9900	0.70	5694.00 (2458.24)	200-9900
0.20	6890.00 (2083.38)	1000-9900	0.75	5642.00 (2562.58)	1200-9800
0.25	6760.00 (2324.22)	1200-9800	0.80	5718.00 (2445.70)	500-9900
0.30	5936.00 (2341.34)	600-9500	0.85	6086.00 (2457.07)	700-9900
0.35	6054.00 (2685.83)	500-9700	0.90	5754.00 (2793.43)	200-9700
0.40	5920.00 (2539.68)	500-9500	0.95	5892.00 (2525.06)	300-9700
0.45	6474.00 (2302.76)	1400-9900	1.00	5702.00 (2821.09)	300-9800
0.50	6482.00 (2440.46)	1300-9800	-	-	-

Fitnesses For COBRA with p(Xover)= 0.35 and Varying Gap Size

Gap Size	Mean (Std Dev)	Range	Gap Size	Mean (Std Dev)	Range
200	32.80 (6.449)	16-40	1200	33.28 (7.040)	16-48
400	33.60 (6.974)	24-48	1400	32.96 (7.443)	16-48
600	34.08 (6.361)	24-48	1600	33.92 (6.105)	16-48
800	34.08 (5.945)	24-48	1800	33.60 (7.332)	16-48
1000	33.44 (5.920)	24-48	2000	35.04 (6.174)	24-56

Evaluations Reqd. For COBRA with p(Xover)= 0.35 and Varying Gap Size

Gap Size	Mean (Std Dev)	Range	Gap Size	Mean (Std Dev)	Range
200	7664.00 (1723.22)	1200-9900	1200	7152.00 (1823.86)	1000-9900
400	7580.00 (1598.87)	3400-9800	1400	7026.00 (2079.01)	600-9900
600	7728.00 (1583.16)	3400-9900	1600	7670.00 (1622.12)	3200-9900
800	7446.00 (1638.43)	4100-9900	1800	7410.00 (1709.64)	3000-9700
1000	7616.00 (1781.61)	3200-9900	2000	7810.00 (1626.31)	2100-9900

Fitnesses For COBRA with Gap Size = 1000

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.05	28.16 (7.877)	8-40	0.55	34.88 (7.112)	24-56
0.10	29.92 (7.807)	16-48	0.60	34.72 (6.714)	16-48
0.15	30.08 (7.254)	16-48	0.65	34.24 (7.680)	16-56
0.20	32.00 (8.158)	16-48	0.70	34.08 (5.498)	24-48
0.25	33.44 (6.917)	24-56	0.75	33.44 (6.536)	24-48
0.30	33.12 (7.674)	24-56	0.80	31.04 (6.327)	16-48
0.35	33.44 (5.920)	24-48	0.85	29.12 (6.146)	16-40
0.40	33.92 (5.891)	24-40	0.90	29.60 (7.547)	16-48
0.45	35.84 (6.037)	24-48	0.95	25.28 (7.219)	16-48

Evaluations Reqd. For COBRA with Gap Size = 1000

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.05	5604.00 (2605.75)	600-9500	0.55	7620.00 (1761.59)	1500-9900
0.10	5966.00 (2286.09)	1000-9700	0.60	7578.00 (1632.82)	3600-9800
0.15	6712.00 (2230.93)	1300-9600	0.65	7292.00 (1696.91)	2900-9900
0.20	6806.00 (1795.37)	2700-9900	0.70	7636.00 (1571.21)	3800-9900
0.25	7086.00 (1989.37)	2000-9800	0.75	7520.00 (1419.15)	4100-9900
0.30	7466.00 (1436.88)	4900-9900	0.80	7216.00 (1718.18)	2900-9800
0.35	7616.00 (1781.61)	3200-9900	0.85	6648.00 (1937.34)	1400-9900
0.40	7478.00 (1546.38)	4400-9900	0.90	7016.00 (2049.52)	500-9900
0.45	7620.00 (1610.21)	3900-9900	0.95	5596.00 (2224.85)	600-9700

B.3.5 The Long Path Problem

Fitnesses For A Standard GA

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	35823.60 (16035.06)	134-49179	0.55	48686.72 (3440.53)	24603-49179
0.05	35855.28 (14854.81)	3040-49179	0.60	48612.54 (3963.93)	20865-49179
0.10	36550.54 (14799.13)	6127-49179	0.65	47213.02 (6666.96)	24604-49179
0.15	39987.68 (13176.81)	2235-49179	0.70	47090.14 (6686.35)	24603-49179
0.20	39440.08 (12403.98)	2718-49179	0.75	47212.94 (6667.23)	24603-49179
0.25	41595.70 (11483.50)	12221-49179	0.80	48161.38 (4814.72)	24604-49179
0.30	41288.76 (13626.49)	6172-49179	0.85	45984.20 (9771.60)	12316-49179
0.35	46863.18 (7686.79)	12269-49179	0.90	41806.42 (12287.67)	12315-49179
0.40	44721.68 (9428.15)	24603-49179	0.95	39258.50 (16587.45)	124-49179
0.45	46359.74 (7492.58)	24603-49179	1.00	31970.14 (18953.33)	123-49179
0.50	47685.56 (5832.60)	24603-49179	-	-	-

Evalutions Reqd. For A Standard GA

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	7836.00 (2045.16)	1700-9900	0.55	5370.00 (2137.96)	1500-9900
0.05	7848.00 (2171.47)	1700-9900	0.60	5516.00 (1953.90)	2700-9500
0.10	7864.00 (2053.55)	1800-9900	0.65	5144.00 (1895.16)	1700-9900
0.15	7368.00 (2326.58)	600-9900	0.70	4852.00 (2162.24)	1400-9400
0.20	7178.00 (2262.32)	300-9900	0.75	4920.00 (1894.20)	1600-9000
0.25	7040.00 (2328.60)	300-9900	0.80	4940.00 (2272.00)	800-9600
0.30	6986.00 (2121.50)	2300-9900	0.85	4826.00 (1974.51)	2100-9100
0.35	6634.00 (2278.82)	1600-9900	0.90	5026.00 (2514.74)	1900-9800
0.40	6606.00 (2444.37)	1600-9900	0.95	3976.00 (1717.73)	1600-8500
0.45	5254.00 (2440.09)	1300-9900	1.00	3040.00 (1011.92)	1300-6700
0.50	5924.00 (1938.30)	1900-9500	-	-	-

Fitnesses For Coevolution Method #1

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	47029.74 (6679.51)	24603-49179	0.55	49163.64 (107.23)	48413-49179
0.05	47702.82 (5836.81)	23848-49179	0.60	47643.76 (5293.39)	24603-49179
0.10	48196.02 (4815.59)	24603-49179	0.65	47643.00 (6089.82)	21533-49179
0.15	47703.46 (5835.97)	24604-49179	0.70	46306.18 (7494.15)	23835-49179
0.20	47518.14 (6760.78)	12317-49179	0.75	48318.76 (4258.45)	24603-49179
0.25	46967.30 (6814.67)	24603-49179	0.80	47212.08 (6663.23)	24604-49179
0.30	46473.48 (8257.61)	12306-49179	0.85	46220.98 (7982.76)	24604-49179
0.35	47212.04 (6663.29)	24604-49179	0.90	47927.56 (4971.63)	24605-49179
0.40	48424.92 (3813.08)	24604-49179	0.95	47061.18 (6532.28)	24603-49179
0.45	45738.44 (8526.08)	24601-49179	1.00	47227.00 (6148.02)	24593-49179
0.50	48502.06 (3543.97)	24604-49179	-	-	-

Evalutions Reqd. For Coevolution Method #1

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	5568.00 (2232.97)	2000-9900	0.55	6018.00 (2152.09)	1600-9700
0.05	5320.00 (2224.32)	1400-9700	0.60	5720.00 (2297.21)	1200-9900
0.10	5150.00 (1796.69)	2000-9800	0.65	5442.00 (1735.06)	2400-9700
0.15	6028.00 (2018.52)	2700-9900	0.70	6202.00 (2069.25)	1900-9900
0.20	5582.00 (2079.68)	1900-9800	0.75	5272.00 (2026.33)	1600-9400
0.25	5980.00 (2441.72)	1300-9600	0.80	5544.00 (2459.36)	1000-9700
0.30	5388.00 (2223.11)	1700-9900	0.85	5896.00 (2198.99)	1700-9900
0.35	6108.00 (2204.07)	1900-9800	0.90	5250.00 (2161.50)	1900-9300
0.40	5886.00 (2376.38)	1900-9900	0.95	6362.00 (1997.68)	2000-9500
0.45	5966.00 (1938.92)	1800-9900	1.00	5952.00 (2289.73)	2100-9800
0.50	5856.00 (1920.54)	1700-9600	-	-	-

Results For Co-evolution Method #2

Measure	Mean (Std Dev)	Range
Fitness	47704.44 (5836.47)	24603-49179
Evals Reqd.	5680.00 (2084.41)	1500-9900

Fitnesses For Co-evolution Method #3

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	47444.30 (6952.55)	12316-49179	0.55	47542.76 (5862.19)	24603-49179
0.05	46660.06 (7566.79)	21533-49179	0.60	46722.14 (7367.85)	24603-49179
0.10	48687.48 (3440.64)	24603-49179	0.65	45983.32 (8795.33)	12317-49179
0.15	45984.18 (8795.83)	12316-49179	0.70	46042.68 (8700.18)	12315-49179
0.20	47704.50 (6332.53)	12316-49179	0.75	46659.62 (7280.97)	24603-49179
0.25	47704.50 (5836.23)	24604-49179	0.80	47704.30 (5293.65)	24603-49179
0.30	47950.28 (5066.24)	24603-49179	0.85	47916.22 (6111.06)	12315-49179
0.35	46967.28 (6814.81)	24603-49179	0.90	47581.56 (5867.90)	24604-49179
0.40	46134.00 (8475.08)	11571-49179	0.95	47595.56 (5828.85)	24603-49179
0.45	48195.98 (4815.79)	24603-49179	1.00	46652.36 (7645.07)	18077-49179
0.50	47527.78 (5922.05)	24603-49179	-	-	-

Evaluations Reqd. For Co-evolution Method #3

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	5624.00 (2241.38)	1600-9900	0.55	5888.00 (2347.64)	1700-9900
0.05	5048.00 (2307.14)	1500-9900	0.60	5562.00 (2270.32)	1200-9900
0.10	4768.00 (1870.66)	1600-9400	0.65	5626.00 (2213.13)	1300-9600
0.15	5332.00 (1911.27)	2500-9500	0.70	5922.00 (2416.71)	1100-9900
0.20	5980.00 (2235.62)	2600-9900	0.75	5912.00 (2155.05)	1500-9700
0.25	5380.00 (1975.96)	1100-9900	0.80	5512.00 (2460.05)	1500-9700
0.30	5324.00 (1847.65)	1900-9500	0.85	5644.00 (2173.21)	2300-9600
0.35	4968.00 (2264.37)	1800-9600	0.90	5814.00 (2211.24)	1700-9900
0.40	5374.00 (2431.11)	1300-9900	0.95	5876.00 (2300.48)	1800-9800
0.45	5002.00 (2147.23)	1700-9400	1.00	5762.00 (2115.65)	1300-9800
0.50	5694.00 (2034.83)	2000-9800	-	-	-

Results For Co-evolution Method #4

Measure	Mean (Std Dev)	Range
Fitness	46719.52 (7372.11)	24603-49179
Evals Reqd.	5024.00 (2039.66)	2300-9700

Fitnesses For Co-evolution Method #5

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	43504.54 (10957.35)	3093-49179	0.55	46406.74 (7401.51)	24601-49179
0.05	41696.92 (12981.57)	411-49179	0.60	45224.06 (10794.86)	3099-49179
0.10	46905.62 (7342.62)	12317-49179	0.65	42694.32 (11644.82)	5790-49179
0.15	41615.84 (14466.34)	412-49179	0.70	42520.38 (11178.65)	12316-49179
0.20	41713.40 (13714.67)	6175-49179	0.75	40517.16 (13796.00)	6170-49179
0.25	39970.50 (14701.62)	5015-49179	0.80	42709.94 (12314.63)	3029-49179
0.30	40024.00 (13203.58)	3097-49179	0.85	37919.76 (15922.62)	126-49179
0.35	45684.00 (9622.51)	11933-49179	0.90	39304.98 (13469.22)	6171-49179
0.40	41503.06 (12324.32)	6172-49179	0.95	42883.82 (11949.69)	12315-49179
0.45	46593.18 (8686.91)	6172-49179	1.00	37911.40 (16568.95)	30-49179
0.50	43055.42 (12186.01)	5404-49179	-	-	-

Evaluations Reqd. For Co-evolution Method #5

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	7058.00 (2102.10)	1800-9800	0.55	7020.00 (1720.34)	3000-9800
0.05	6578.00 (2284.84)	1300-9900	0.60	6090.00 (2144.87)	1700-9800
0.10	6286.00 (2157.31)	2400-9600	0.65	7318.00 (2432.42)	300-9900
0.15	6490.00 (2256.66)	2600-9800	0.70	6058.00 (2176.42)	900-9600
0.20	6268.00 (2193.39)	2300-9900	0.75	7084.00 (2123.52)	2400-9800
0.25	6782.00 (2085.82)	3300-9800	0.80	6648.00 (2505.37)	2200-9900
0.30	6850.00 (2522.16)	1500-9900	0.85	7170.00 (2077.52)	2000-9800
0.35	6710.00 (2099.16)	1400-9900	0.90	7184.00 (2255.96)	1400-9900
0.40	6842.00 (2122.27)	2200-9900	0.95	5892.00 (2460.71)	1700-9900
0.45	6594.00 (2043.07)	2800-9900	1.00	6596.00 (2079.52)	2800-9900
0.50	6182.00 (2232.63)	1900-9900	-	-	-

Fitnesses For Co-evolution Method #6

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	41837.66 (12017.93)	6171-49179	0.55	41204.88 (13698.35)	3099-49179
0.05	37774.06 (15119.36)	6172-49179	0.60	41413.56 (14244.30)	3390-49179
0.10	43651.96 (10131.74)	24603-49179	0.65	38326.38 (14419.70)	1564-49179
0.15	41398.04 (12790.30)	6171-49179	0.70	40492.58 (13539.53)	3100-49179
0.20	40835.58 (13070.12)	12315-49179	0.75	37806.78 (15675.28)	1383-49179
0.25	42620.36 (11765.09)	12315-49179	0.80	40068.82 (14823.87)	27-49179
0.30	42812.66 (12317.93)	6170-49179	0.85	37602.92 (16086.92)	520-49179
0.35	44198.72 (11408.40)	3100-49179	0.90	39631.00 (14270.97)	27-49179
0.40	36117.36 (17137.04)	1563-49179	0.95	39023.66 (13884.65)	12310-49179
0.45	41341.28 (12477.21)	10780-49179	1.00	39359.26 (15393.47)	29-49179
0.50	42095.70 (11716.69)	12318-49179	-	-	-

Evaluations Reqd. For Co-evolution Method #6

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.00	6518.00 (2292.30)	2200-9900	0.55	6758.00 (2223.42)	2100-9800
0.05	6146.00 (2134.77)	2100-9900	0.60	6398.00 (2254.01)	1400-9900
0.10	5870.00 (2300.89)	1800-9900	0.65	6646.00 (2133.47)	1900-9700
0.15	5962.00 (2418.83)	900-9900	0.70	6696.00 (2367.35)	1900-9900
0.20	5750.00 (2256.21)	1600-9700	0.75	6768.00 (2333.36)	500-9900
0.25	6706.00 (2091.83)	2200-9900	0.80	7196.00 (2238.92)	700-9900
0.30	6456.00 (2404.34)	1300-9900	0.85	6966.00 (2339.36)	2100-9900
0.35	6508.00 (2180.44)	1800-9800	0.90	6850.00 (2532.21)	1700-9900
0.40	6330.00 (2368.06)	1400-9900	0.95	6310.00 (2173.31)	2700-9800
0.45	6102.00 (2471.07)	500-9900	1.00	6392.00 (2182.09)	2000-9900
0.50	6298.00 (2412.76)	1800-9900	-	-	-

Fitnesses For COBRA with p(Xover)= 0.55 and Varying Gap Size

Gap Size	Mean (Std Dev)	Range	Gap Size	Mean (Std Dev)	Range
200	47458.68 (6955.13)	12317-49179	1200	47696.02 (5834.59)	24603-49179
400	47842.60 (6168.59)	12315-49179	1400	48195.20 (4815.63)	24603-49179
600	48686.12 (3440.45)	24603-49179	1600	48687.44 (3440.77)	24602-49179
800	47887.34 (6147.10)	12315-49179	1800	48687.24 (3440.60)	24603-49179
1000	47950.06 (6143.73)	12315-49179	2000	48686.72 (3440.53)	24603-49179

Evaluations Reqd. For COBRA with p(Xover)= 0.55 and Varying Gap Size

Gap Size	Mean (Std Dev)	Range	Gap Size	Mean (Std Dev)	Range
200	5306.00 (2257.55)	1500-9900	1200	5536.00 (2245.68)	1500-9900
400	5344.00 (2233.30)	1500-9800	1400	5632.00 (2261.54)	1500-9900
600	5534.00 (2193.86)	1500-9900	1600	5434.00 (2221.76)	1500-9900
800	5418.00 (2054.52)	1500-9700	1800	5312.00 (2039.96)	1500-9900
1000	5516.00 (2258.43)	1500-9900	2000	5280.00 (2182.93)	1500-9900

Fitnesses For COBRA with Gap Size = 1000

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.05	41407.06 (14494.93)	1563-49179	0.55	47950.06 (6143.73)	12315-49179
0.10	44141.00 (11106.03)	6172-49179	0.60	47875.72 (6441.92)	12315-49179
0.15	47150.10 (8239.43)	6172-49179	0.65	45477.68 (9271.48)	12317-49179
0.20	45001.02 (9709.72)	12315-49179	0.70	46721.42 (7372.74)	24603-49179
0.25	48687.44 (3440.63)	24603-49179	0.75	46721.00 (7372.53)	24603-49179
0.30	48196.00 (4815.69)	24604-49179	0.80	46932.62 (7643.73)	12317-49179
0.35	49056.16 (859.88)	43037-49179	0.85	46352.80 (9769.42)	6171-49179
0.40	49177.08 (13.44)	49083-49179	0.90	42046.36 (11817.65)	12315-49179
0.45	48134.72 (4821.75)	24604-49179	0.95	41610.52 (14174.58)	124-49179

Evaluations Reqd. For COBRA with Gap Size = 1000

p(Xover)	Mean (Std Dev)	Range	p(Xover)	Mean (Std Dev)	Range
0.05	4286.00 (2338.20)	1700-9800	0.55	5516.00 (2258.43)	1500-9900
0.10	4792.00 (2396.31)	1700-9900	0.60	5702.00 (1928.78)	2800-9900
0.15	4352.00 (2043.25)	1900-9500	0.65	5540.00 (2095.51)	1700-9900
0.20	5576.00 (2221.67)	2200-9900	0.70	4728.00 (1929.25)	1400-9800
0.25	4876.00 (2230.11)	1600-9900	0.75	5302.00 (1984.58)	1600-9100
0.30	5086.00 (2350.49)	1600-9900	0.80	4958.00 (2236.07)	800-9600
0.35	5208.00 (1850.49)	1900-8900	0.85	4874.00 (1983.41)	2100-9000
0.40	4756.00 (2057.00)	2000-9700	0.90	4842.00 (2325.77)	1900-9800
0.45	5200.00 (2012.26)	1300-9900	0.95	4480.00 (2154.25)	1600-9400

Appendix C

Graphs

C.1 Overview

This appendix provides all of the graphs used, or referred to in the main body of this dissertation. The graphs are typical of the behaviour shown by the GA. To ensure this, at least 10 runs were inspected in each case.

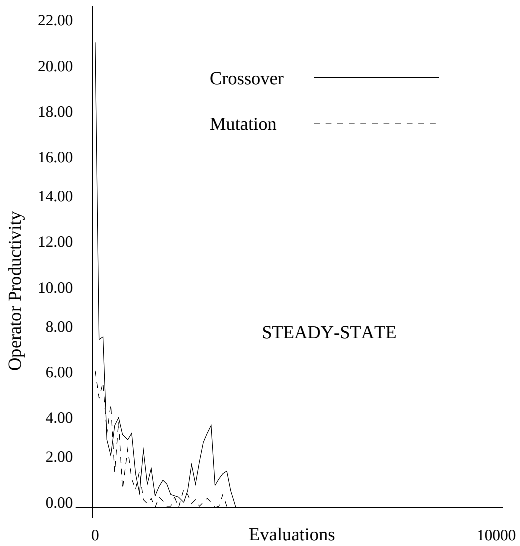
The first set of graphs depict the operator productivities for both crossover and mutation during the course of a GA run. Operator productivity is defined as the average *improvement* produced from parent to child by the application of a given operator over the last 100 operations. Error bars were not included on these graphs for reasons of clarity; in fact the operator productivity information was found to be *very* noisy.

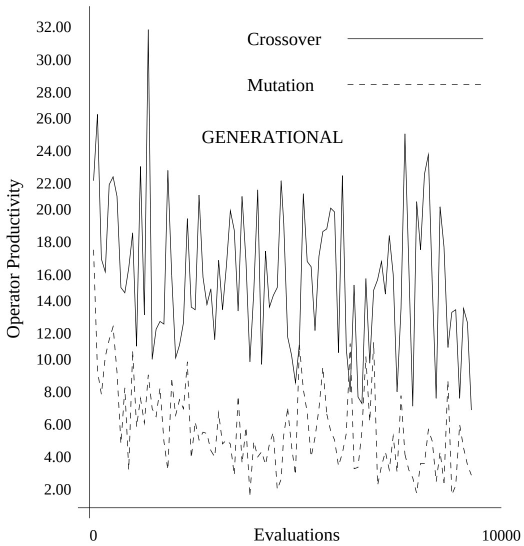
The second set of graphs plot the evolved crossover probability against the number of evaluations made so far, for co-evolution methods 1 to 4. The error bars show the standard deviation of the crossover probability in the GA population.

The fourth set of graphs shown here plot the evolved mutation parameter (which can take the range of 0 to $1/l$) during the course of a GA run. The error bars, in this case, show the standard deviation of the mutation parameter in the GA population.

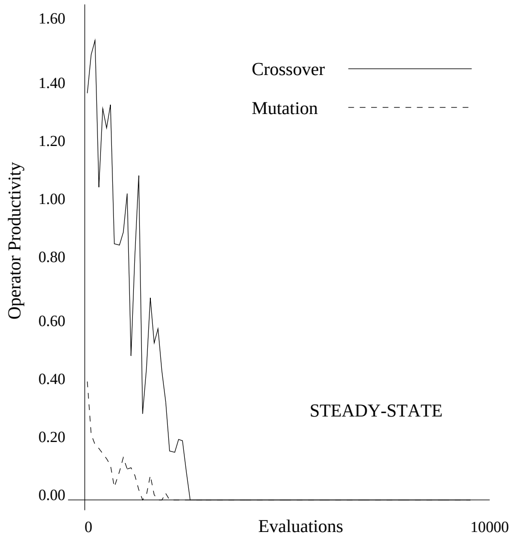
C.2 Plots of Operator Productivities

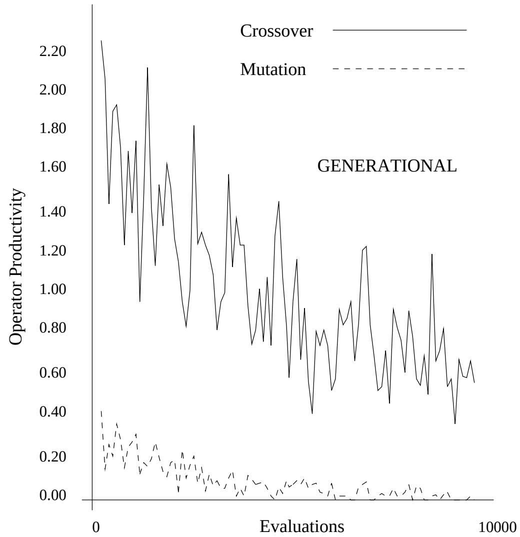
C.2.1 The $n/m/P/C_{max}$ Problem



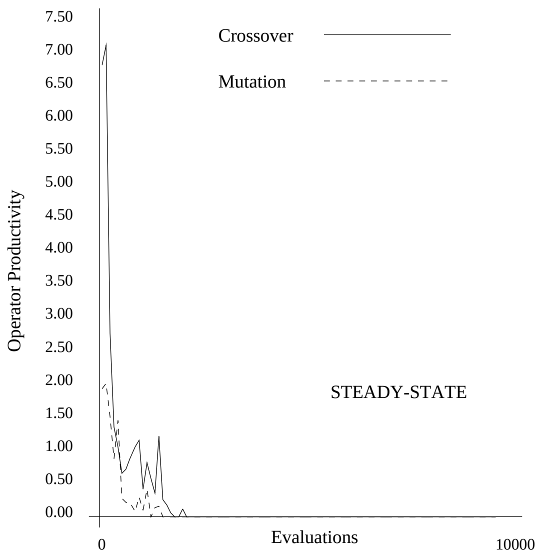


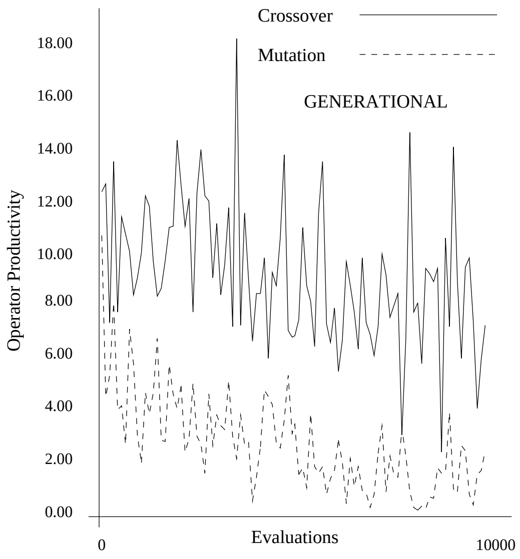
C.2.2 The Counting Ones Problem



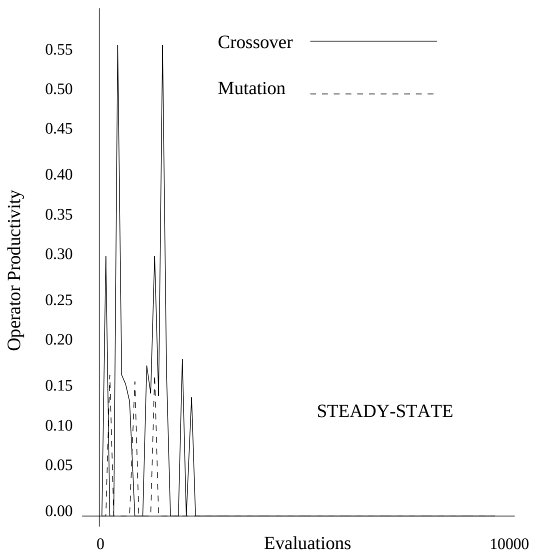


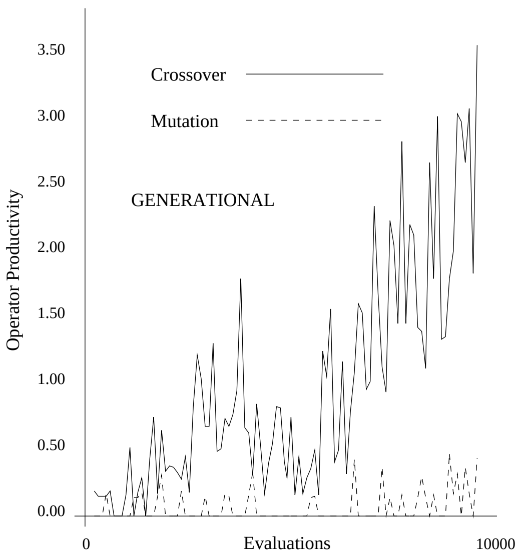
C.2.3 The Order-3 Deceptive Problem



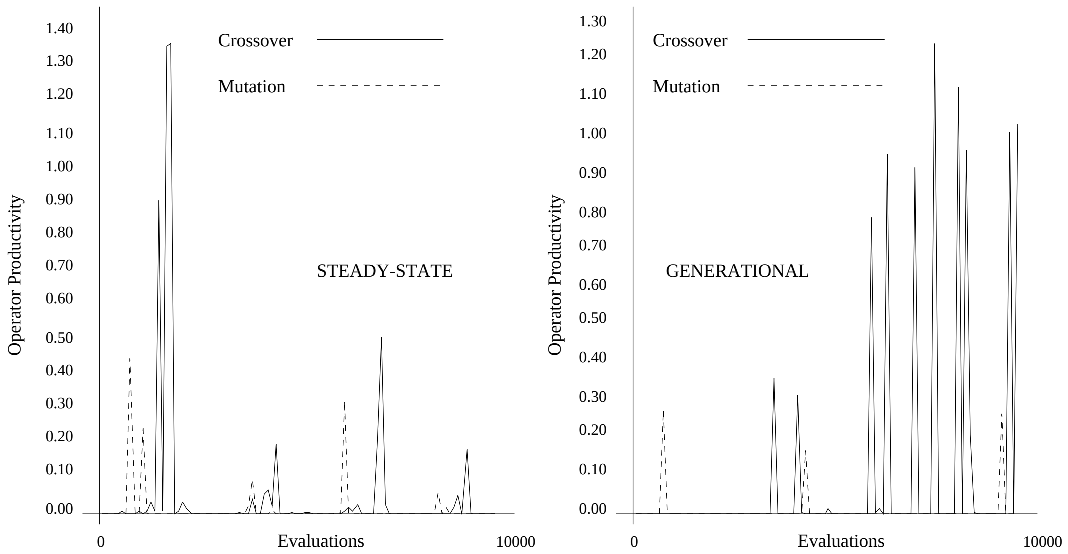


C.2.4 The Royal Road Problem





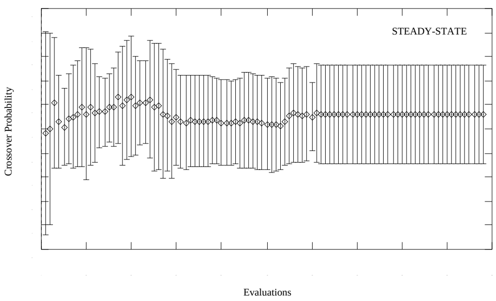
C.2.5 The Long Path Problem

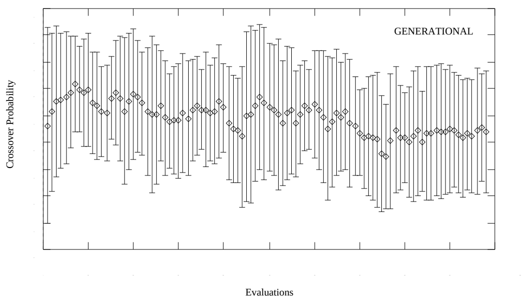


C.3 Plots of Evolved Crossover Probability

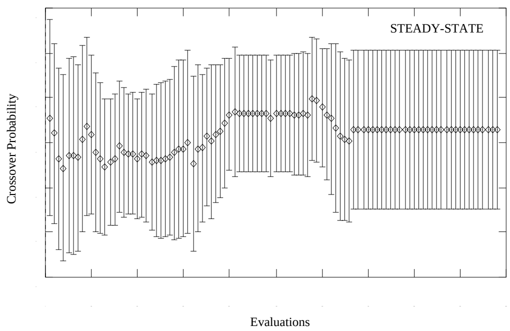
C.3.1 The $n/m/P/C_{max}$ Problem

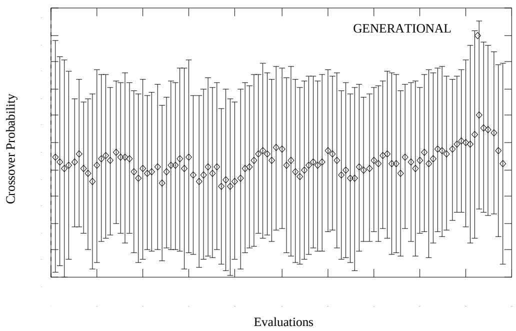
Co-evolution Method #1



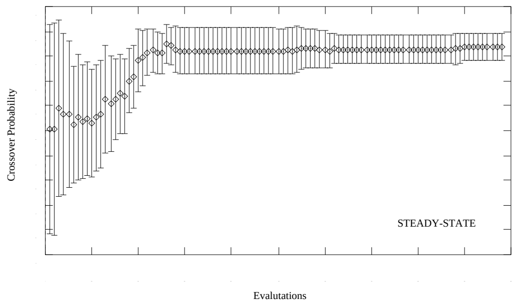


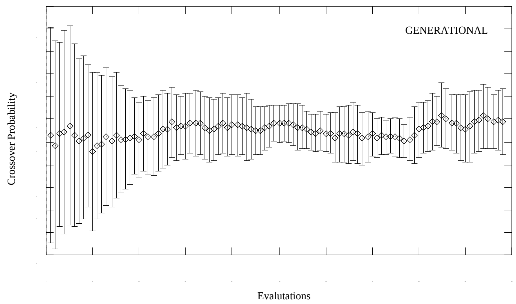
Co-evolution Method #2



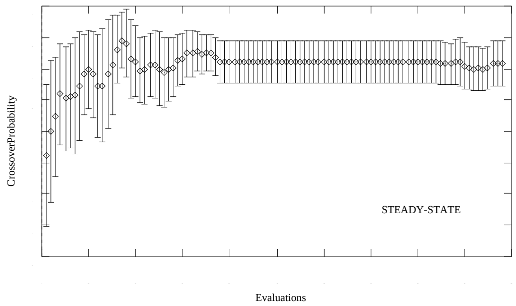


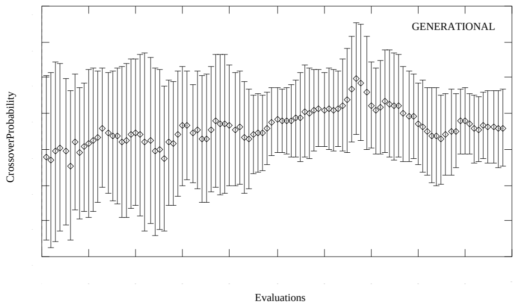
Co-evolution Method #3





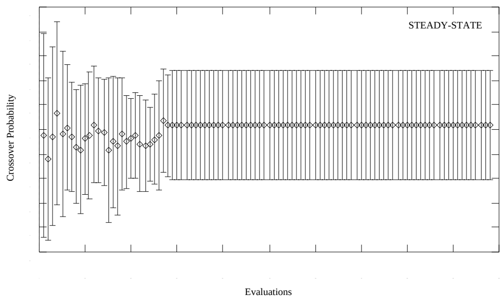
Co-evolution Method #4

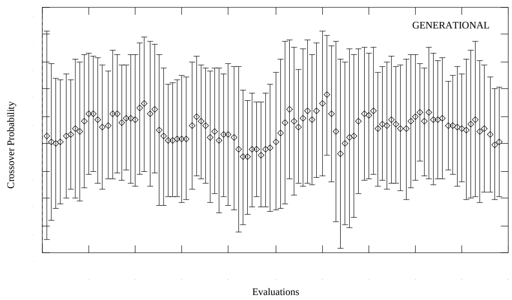




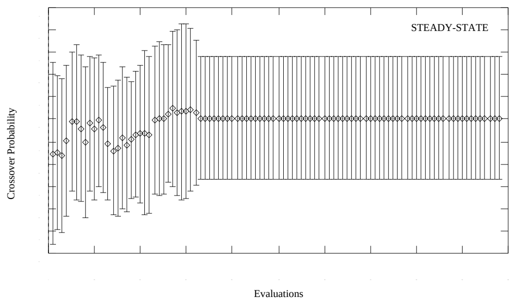
C.3.2 The Counting Ones Problem

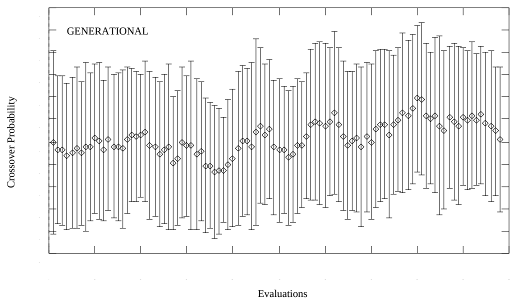
Co-evolution Method #1



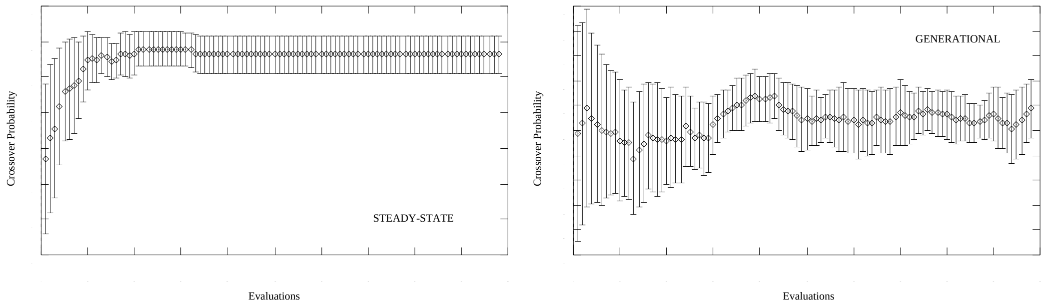


Co-evolution Method #2

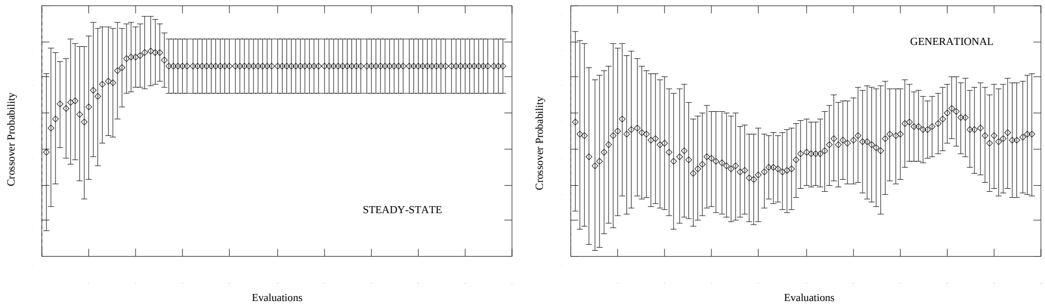




Co-evolution Method #3

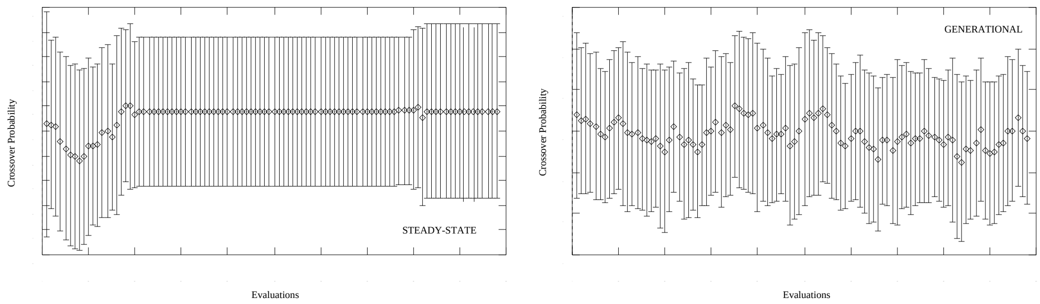


Co-evolution Method #4

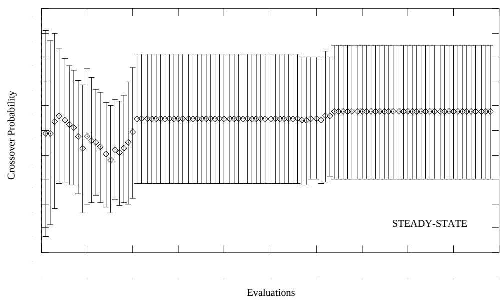


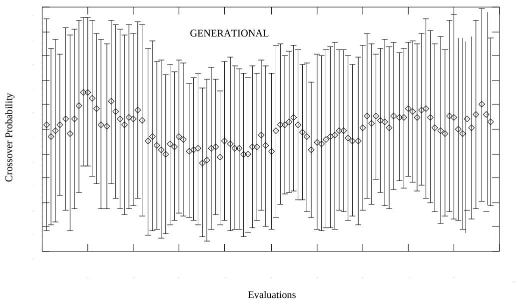
C.3.3 The Order-3 Deceptive Problem

Co-evolution Method #1

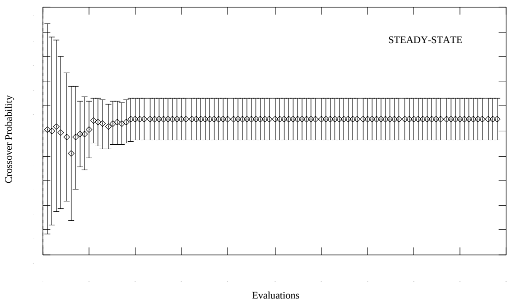


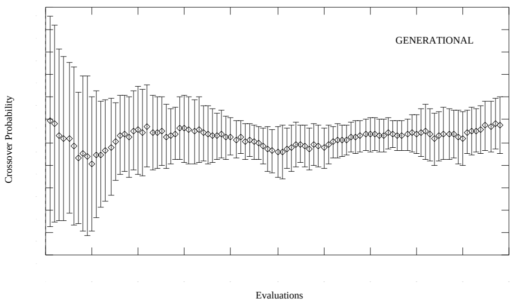
Co-evolution Method #2



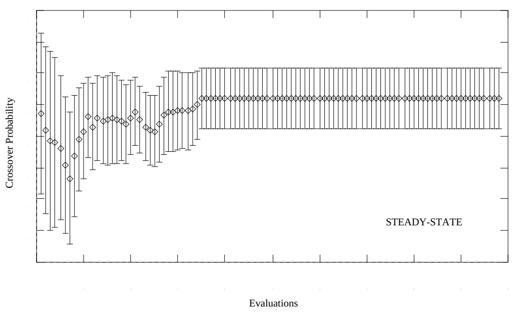


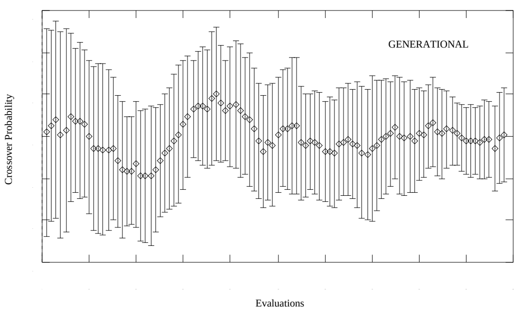
Co-evolution Method #3





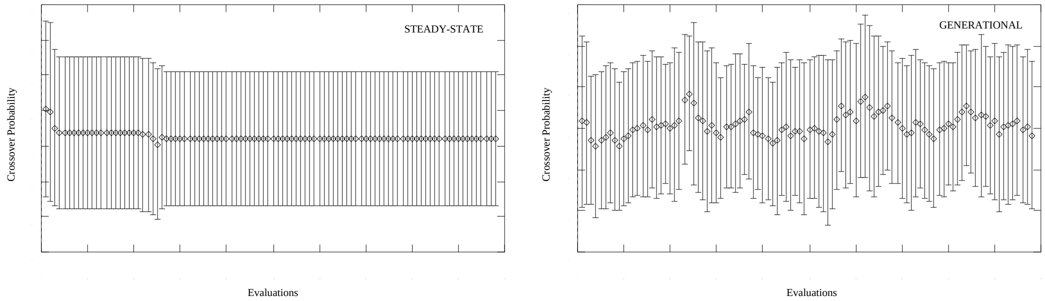
Co-evolution Method #4



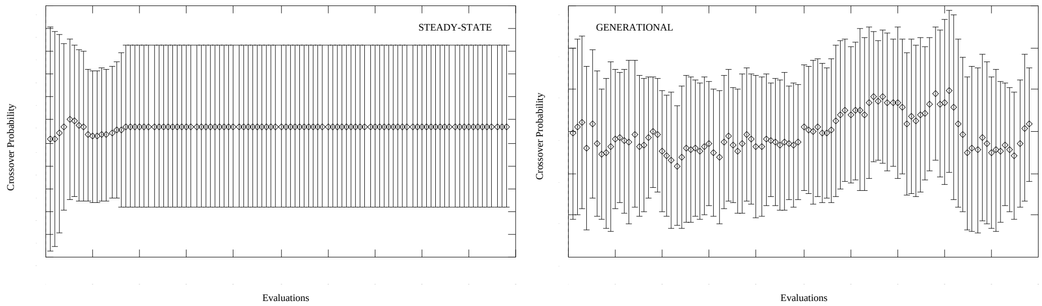


C.3.4 The Royal Road Problem

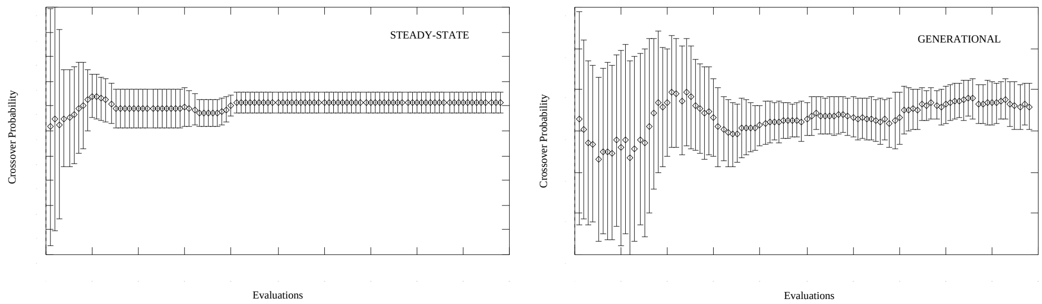
Co-evolution Method #1



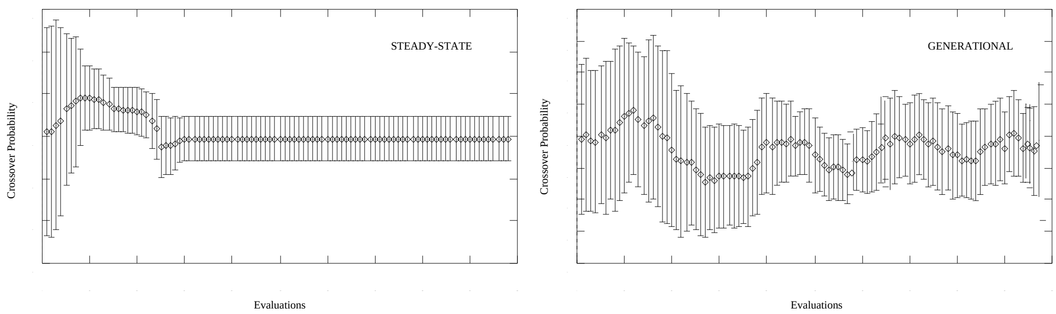
Co-evolution Method #2



Co-evolution Method #3

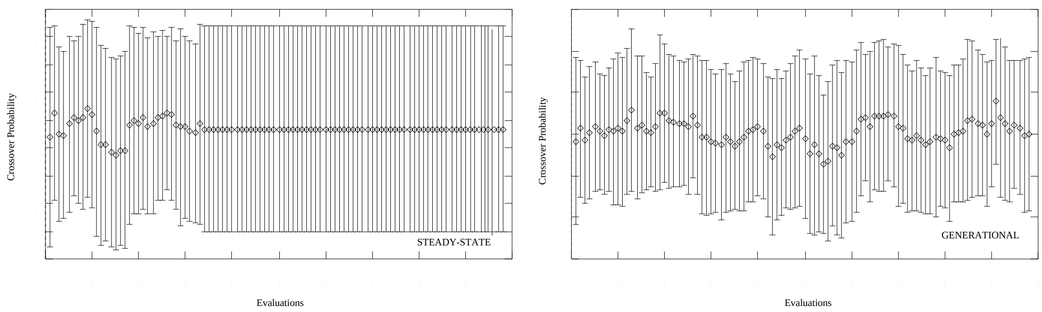


Co-evolution Method #4

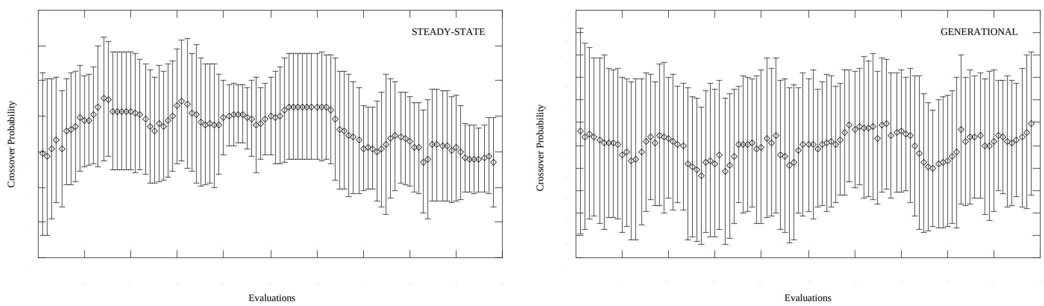


C.3.5 The Long Path Problem

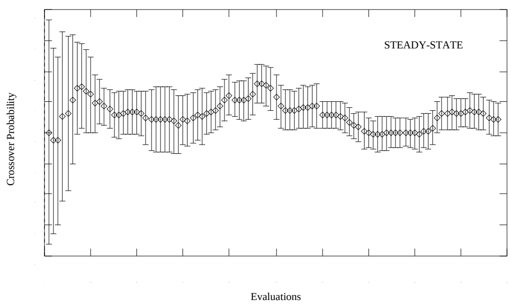
Co-evolution Method #1

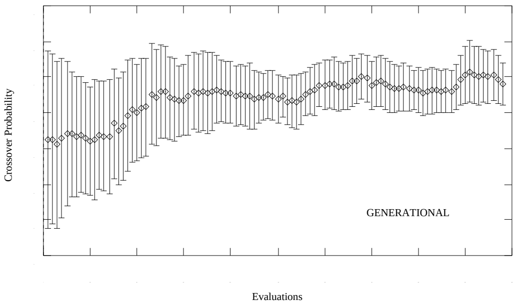


Co-evolution Method #2

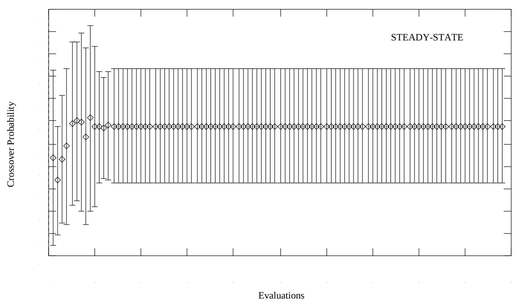


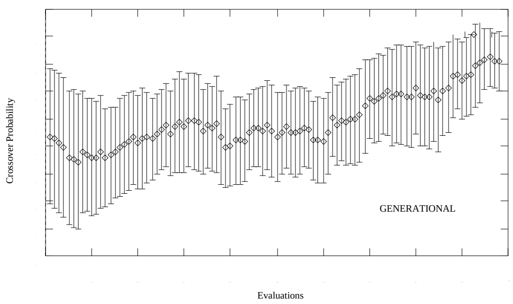
Co-evolution Method #3





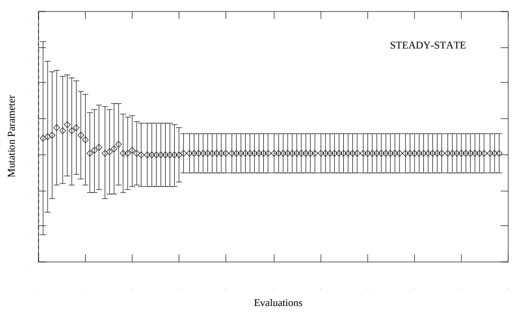
Co-evolution Method #4

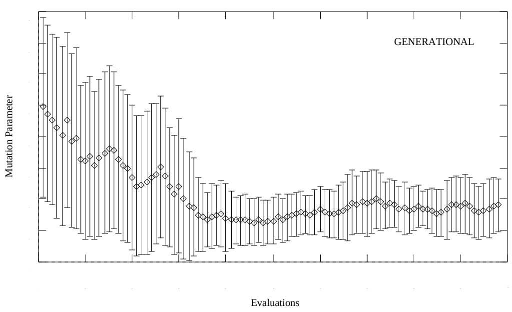




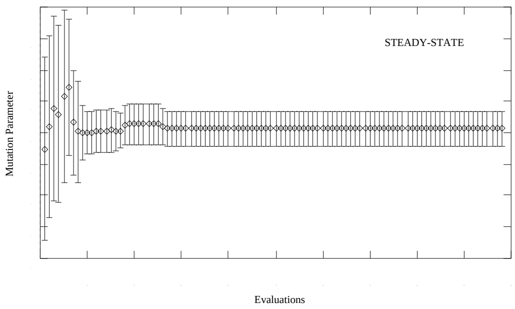
C.4 Plots of The Evolved Mutation Parameter

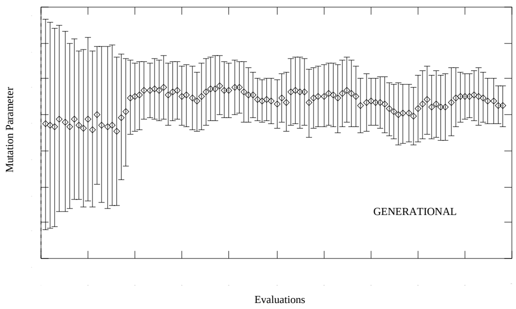
C.4.1 The Counting Ones Problem



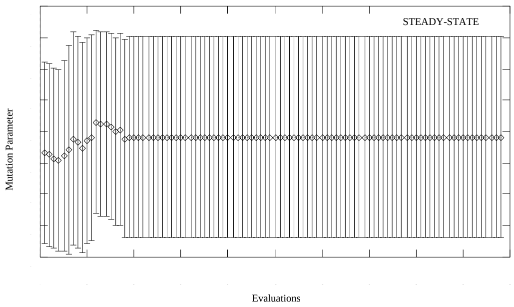


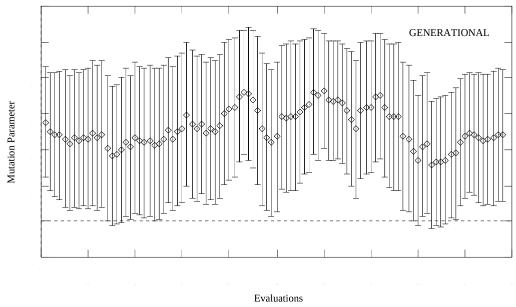
C.4.2 The Order-3 Deceptive Problem





C.4.3 The Royal Road Problem





C.4.4 The Long Path Problem