Adapting Operator Settings In Genetic Algorithms

Andrew Tuson and Peter Ross
Department of Artificial Intelligence, University of Edinburgh
80 South Bridge, Edinburgh EH1 1HN, U.K.
Email: {andrewt,peter}@aisb.ed.ac.uk
Tel: +44 (0)131 650 2715

Key Words: co-evolution, operator adaptation, COBRA, self-adaptation, operator settings.

Abstract

In the vast majority of genetic algorithm implementations, the operator settings are fixed throughout a given run. However, it has sometimes been argued that these settings 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 paper describes an empirical investigation into this question. The effect upon genetic algorithm performance of two adaptation methods upon both well-studied theoretical problems, and a hard problem from Operations Research — the flowshop sequencing problem, is examined. The results obtained indicate that the applicability of operator adaptation is problem-dependent.

Adapting Operator Settings In Genetic Algorithms

Andrew Tuson and Peter Ross
Department of Artificial Intelligence, University of Edinburgh
80 South Bridge, Edinburgh EH1 1HN, U.K.
Email: {andrewt,peter}@aisb.ed.ac.uk
Tel: +44 (0)131 650 2717

Key Words: co-evolution, operator adaptation, COBRA, self-adaptation, operator settings.

Abstract

In the vast majority of genetic algorithm implementations, the operator settings are fixed throughout a given run. However, it has sometimes been argued that these settings 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 paper describes an empirical investigation into this question. The effect upon genetic algorithm performance of two adaptation methods upon both well-studied theoretical problems, and a hard problem from Operations Research — the flowshop sequencing problem, is examined. The results obtained indicate that the applicability of operator adaptation is problem-dependent.

1 Introduction

It has long been acknowledged that the choice of operator settings has a significant impact upon genetic algorithm performance. However, finding a good choice is somewhat of a black art. The appropriate settings depend upon the other components of the genetic algorithm, such as the population model, the problem to be solved, its representation, and the operators used. The large number of possibilities precludes an exhaustive search of the space of operator probabilities.

The above also ignores the case for varying operator settings. There is evidence, both empirical and theoretical, that the most effective operator settings do vary during the course of a genetic algorithm run. For instance, Davis

[Davis, 1991] advocates the use of a time-varying schedule of operator probabilities, and finds that performance is improved — especially when a large number of operators are used.

Theoretical work [Mühlenbein, 1992] 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 genetic algorithm is. Time dependency was also discovered for mutation parameters by Hesser and Männer [Hesser and Männer, 1991].

The problem lies in devising such a schedule — this is harder than finding a good set of static operator settings. It may be advantageous, therefore, to employ a method that dynamically adjusts the operator probabilities according to a measure of the performance of each operator.

So by what criterion should we judge the performance of an operator at a given point? The ability of an operator to produce new, preferably fitter, children may be what is required — this has been suggested before by [Spears and DeJong, 1991], 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 a 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, 1994] 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.

This paper describes an investigation of some of these methods.

2 An Overview of Operator Adaptation

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

Other methods do exist that adjust operator setting based on other criteria, such as the diversity of the population (for example [Coyne and Paton, 1994]), but these will not be considered in this paper.

Adaptation methods can be divided into two classes (for a review of work on adapting operator settings see [Tuson, 1995]):

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

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

The following terminology will be used. Each operator available to the genetic algorithm has a probability of being fired — an *operator probability*. This study makes a distinction between this and any parameters associated with a given operator (henceforth an *operator parameter*). For example, a genetic algorithm could use uniform crossover 70% (operator probability) of the time, along with mutation 30% of the time, with the mutation operator possessing a bitwise mutation rate of 0.02 (operator parameter). The term *operator setting* will be taken to mean both of the terms above.

2.1 Adaptation by Co-evolution

Operator adaptation methods based on the co-evolutionary metaphor (also referred to as 'Self-adaptation') encode the operator settings onto each member of the population and allow them to evolve. The rationale behind this is as follows: solutions which have encoded operator settings that tend to produce fitter children will survive longer and so the useful operator settings will spread through the population. The original work in this area originated from the 'Evolution Strategy' community (see [Bäck, 1996] for a review). 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, 1991][Bäck, 1992], in the sense that the mutation parameter was seen to adapt to the theoretical optimum for the theoretically tractable (simple) problem being considered.

In the study in this paper, the operator probabilities are encoded as floating-point numbers on the range 0 to 1.0, with the constraint that the sum of the operator probabilities must be equal to one. An example is given in Figure 1.

Operator parameters were encoded in a similar fashion, but without the constraint. The meaning of a particular operator parameter depends 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). The value of $5/l$ was selected as it is appreciably larger than the $1/l$ which several authors agree is the optimal value for some problems (e.g. [Mühlenbein and Schlierkamp-Voosen, 1995]).

The operation process is then as follows:

1. Select a parent on the basis of fitness.

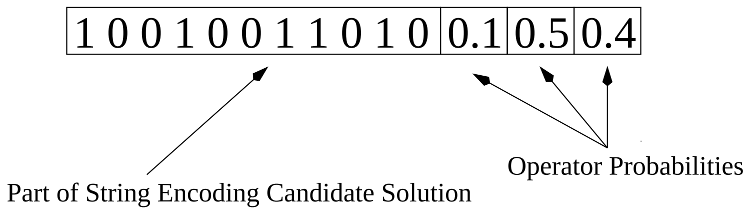

Figure 1: Representing Operator Probabilities

2. Extract the encoded operator probabilities from the parent.

3. Use these operator probabilities to determine (stochastically) which operator is used on the solution part of the string.

4. Apply the chosen operator to the solution part of the parent.

5. Determine (stochastically using a co-evolution crossover probability) the operator (co-evolution crossover or mutation) that will be applied to the encoded operator settings. Then apply it.

6. Renormalise so that the encoded operator probabilities sum to one (renormalisation is not required for the encoded operator parameters).

Each of the experiments below can undergo co-evolution on two levels. One approach uses an externally set co-evolution crossover probability, which sets the probability of using the co-evolution crossover operator. A higher level approach encodes this onto the string also (this is often termed *meta-learning*). The experiments performed are described below.

Two types of co-evolution operators were investigated. The first type were *strongly 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 in this case was the real-coded variant of Random Respectful Recombination (R^3) [Radcliffe, 1991]. For each real coded gene, the value of the child was simply randomly chosen from the interval bounded by the values of the two parents. The mutation operator was 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 alternative to this is to use *weakly disruptive* operators — the operations on the encoded probabilities are designed so that the child produced is quite similar to its parent. In this case, The crossover operator used is parameterised uniform crossover with parameter 0.1. The mutation operator simply changes 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.

Finally, the effect of adapting the operator parameters was also investigated. This investigation used operators of low disruption as this was the type of operator used in previous investigations. The operator probabilities, in this case, were fixed throughout the genetic algorithm run — only the operator parameters evolved.

2.2 Learning-Rule Methods

Another approach is to periodically measure the performance of the operators being used, and utilise this information to periodically adjust the operator settings accordingly. Previous work in this area has adjusted the operator probabilities according to population statistics, although there seems to be 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 give credit to an operator for producing a good child and also to the operators that produced the child's ancestors. This is in order to credit operators that, although they may not produce particularly good children, set the scene for another operator to make improvements (rather like the bucket-brigade algorithm often used in classifier systems [Goldberg, 1989]). The fourth technique (which is the subject of this investigation and will be described later) does not do this; although there is no reason why such a feature could not be added, if desired.

The first for these was devised by Davis [Davis, 1989]. The algorithm outlined appears quite complicated at first sight: each solution in the population records who its parents were, any credit it may have, and the operators that created it. A proportion of the credit for any improvements (in this case compared to the fittest member of the population) made are added to the child, a proportion of the remainder to its parents(s), and so on for a set number of generations. Davis did not use his method on-line, but instead 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.

A similar technique that requires less bookkeeping is provided by Julstrom [Julstrom, 1995]. 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 in an appropriate fashion. Experiments showed that both approaches were able to adapt operator probabilities accordingly.

One objection is that these methods require a lot of additional bookkeeping; indeed no empirical evidence has been given to justify the added complexity. This paper investigates a simpler learning-rule method: COBRA (Cost Operator-Based Rate Adaptation) [Corne et al., 1994], originally devised for adapting operator probabilities in timetabling problems.

A description of how COBRA is implemented follows. Given k operators $o_1, ..., o_k$, let $b_i(t)$ be the *benefit* (this study used *operator productivity* — the average increase in fitness when a child was produced that was fitter than its parents over a set interval), $c_i(t)$, the *cost* (a measure of the amount of computational effort used to evaluate a child), and $p_i(t)$, the probability of a given operator, i, being used at time t. We then apply the following algorithm:

1. The user decides on a set of fixed probabilities p_i. This can either be by experiment, or the user could use the probabilities that were found to work well for a genetic algorithm with fixed probabilities (there is no guarantee, however, that these will be appropriate when using COBRA).
2. After G (the gap between operator probability readjustments) evaluations, rank the operators according their values of b_i/c_i, and assign the operators their new probabilities according to their rank (i.e. the highest probability to the operator with the highest value of b_i/c_i).
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 initial operator probabilities provided by the user. The effect that these variables have upon the genetic algorithm will be investigated.

2.3 Hybrid Methods

There is no reason why the two approaches described here cannot be combined. A hybrid of the both the co-evolutionary and learning rule approaches was proposed in [Whitley, 1993]. This bears many similarities to the co-evolution of the crossover probability studied here, however only one co-evolution operator was used. This operator increased the crossover probability if the parent was fitter than the child, and decreased the crossover probability otherwise.

3 The Test Problems

In order to properly evaluate the effectiveness (or otherwise) of adaptation by co-evolution, a set of test problems needs to be chosen. 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. Each will be briefly described in turn.

- **The Flowshop Sequencing Problem**

  This is an important problem in Operations Research. In the flowshop sequencing, or $n/m/P/C_{max}$, problem jobs are dispatched to a string of machines joined in a serial fashion. The task is to find an ordering 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. One of the Taillard [Taillard, 1993] benchmark problems was used: a benchmark RNG seed of 479340445 was used to generate a completion times matrix for a flowshop with 20 jobs and 20 machines (the benchmark are available via anonymous FTP from OR-Library — mscmga.ms.ic.ac.uk).

- **The Max 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. A string length of 100 was used.

- **Goldberg's Order-3 Deceptive Problem**

  The problems that *deception* can present to a genetic algorithm has been well-studied. A classic problem in such work is the order-3 'tight' deceptive problem devised by Goldberg [Goldberg et al., 1989]. The problem used has a of length 30 bits.

- **The Royal Road Problem**

  Work by Forrest and Mitchell [Forrest and Mitchell, 1993] provides the seminal study of this problem. The Royal Road function used in this study (R1) has a of length 64 bits

- **The Long Path Problem**

  Recent work [Horn et al., 1994] 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 — because of local minima (there are none) — but due to the extreme length of the path to the optimum (the path length is proportional to $2^{l/2}$). This study examined the 'Root2Path' function, represented by a binary string of length 29.

4 Implementation

An unstructured population was used. Two population models were used as part of this study, in order to see if this has any effect upon the success (or

$$\begin{array}{c|c|c} 1\,3\,6 & 5\,4\,2 & 7\,8 \\ 6\,4\,3 & 7\,1\,5 & 2\,8 \end{array} \xrightarrow{\text{Crossover}} \begin{array}{c} 1\,3\,6\,7\,\boxed{1}\,5\,\boxed{7}\,8 \\ 6\,4\,3\,5\,\boxed{4}\,2\,\boxed{2}\,8 \end{array} \xrightarrow{\text{Legalise}} \begin{array}{c} 1\,3\,6\,7\,4\,5\,2\,8 \\ 6\,4\,3\,5\,1\,2\,7\,8 \end{array}$$

Figure 2: The Modified PMX Operator

otherwise) of operator adaptation. The first population model used was *steady-state* reproduction and a *kill-worst* replacement policy; the second model was *generational* replacement with *elitism*.

In both cases, in order to operate on the population, a parent is selected using *rank-based* selection. The operator to use is then selected (according to its probability of selection), and if the operator is crossover, a second parent is selected entirely at random. The operator is then applied to the parent(s) to produce the child.

The representation used for the flowshop sequencing problem was a permutation of the jobs to be placed into the flowline, numbered 1 to n. The crossover operator used was the 'Modified PMX' operator [Mott, 1990]. 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 2).

Based on a study of this problem by Reeves [Reeves, 1995], *shift* mutation was selected for the flowshop sequencing problem. This operator randomly picks an element and shifts it to another randomly selected position.

For the remaining problems, all binary-encoded, the representation used was a string of binary digits. The operators used were: binary 'bit-flip' mutation with bitwise mutation probability of m/l, where l was the length of the string (m has a default value of 1); and parameterised uniform crossover (default parameter value of 0.5 was used — each gene is selected from either parent with equal probability).

5 Results

A large number of experiments were performed in this study; too many, to give here. Therefore a summary of the results are given, and the reader is directed to [Tuson, 1995] for full results.

Two measures of performance were used: the quality of solution obtained after 10,000 evaluations, and the number of evaluations after which improvements in solution quality were no longer obtained (or 10,000 evaluations — whichever was smaller). This number of evaluations was chosen as preliminary investigations showed that the GA population had converged long before then.

Problem	Solution Quality	Evaluations
Flowshop	2441 (71) [0.95]	7714 (1573) [0.15]
Max Ones	99.96 (0.20) [0.8]	7714 (1025) [0.8]
Deceptive	289.68 (2.41) [0.05]	4484 (1883) [0.90]
Royal Road	35.52 (6.02) [0.35]	5626 (2254) [0.95]
Long Path	48687 (3441) [0.55]	3976 (1717) [0.95]

Table 1: Best Results for a Generational GA with Fixed Operator Probabilities

Problem	Solution Quality	Evaluations
Flowshop	2387 (71) [0.05]	5064 (1616) [0.55]
Max Ones	100.00 (0.00) [0.0-0.90]	2172 (702) [0.80]
Deceptive	289.28 (3.22) [0.1]	2095 (1552) [0.95]
Royal Road	40.64 (7.65) [0.95]	2876 (2260) [0.25]
Long Path	48024 (2907) [0.05]	5129 (4088) [0.2]

Table 2: Best Results for a Steady-State GA with Fixed Operator Probabilities

Fifty genetic algorithm runs were performed, for both population models, with a population size of 100 and a rank-based selection pressure of 1.5. A *t-test* was applied in order to ascertain if any differences were significant.

5.1 The Genetic Algorithm With Fixed Operator Settings

The effect of varying crossover probability on a genetic algorithm with fixed operator probabilities was investigated. 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. This provides a benchmark against which the performance of a genetic algorithm using co-evolution will be compared.

This exhaustive search of the crossover probability measured how sensitive this operator setting is to genetic algorithm performance. This gave some indication of how hard the genetic algorithm was to tune, and allowed later comparison of the tuning difficulty when co-evolution is used.

The best (average) results obtained for each problem/population model with crossover probability 0.05-0.95 (extreme values are rarely useful) are given in Tables 1 and 2. For each entry, the standard deviation is given in parentheses, and the crossover probability at which the sample was taken is given in *square* brackets.

When the trends in performance against crossover probability are examined, some general patterns were observed (reflected somewhat in Tables 1 and 2). The choice of operator probabilities appears to depend upon the problem to be solved, the population model (interestingly, the steady-state genetic algorithm gave consistently better performance), and the performance criterion being used.

Problem	Solution Quality	Evaluations Reqd.
Flowshop	2444 (67)	7758 (1688)
Max Ones	99.90 (0.30)	8374 (839)
Deceptive	288.28 (2.80)	5692 (2159)
Royal Road	35.36 (6.01)	6794 (1805)
Long Path	49164 (107)	5150 (1797)

Table 3: Co-evolution with Strongly Disruptive Operators & Generational GA

Problem	Solution Quality	Evaluations Reqd.
Flowshop	2394 (72)	5063 (1628)
Max Ones	100.00 (0.00)	2478 (418)
Deceptive	288.60 (2.31)	2518 (2124)
Royal Road	32.80 (9.23)	2759 (1961)
Long Path	46649 (6831)	4622 (4165)

Table 4: Co-evolution with Strongly Disruptive Operators & Steady-State GA

The final point is illustrated by the deceptive problem: a low crossover probability gives high quality results, a high crossover probability exchanges solution quality for a higher speed of search.

5.2 Co-evolution With Strongly Disruptive Operators

For each of the problems, an exhaustive search was made of the operator probabilities: a genetic algorithm was run for co-evolution crossover probabilities 0.0 to 1.0 with steps of 0.05. Experiments were also performed using meta-learning.

The best average performance attained for each problem/population model are given in Tables 3 and 4 with standard deviations given in parentheses. Throughout the rest of this paper, underlined table entries indicate a significant difference in performance (by t-test) when compared against a tuned genetic algorithm with fixed operator probabilities.

The performance of the GA was found, in all cases, to be insensitive to the choice of co-evolution crossover probability. The effect of using meta-learning was found to be insignificant.

Comparison with a genetic algorithm with a fixed crossover probability indicates a significant drop in performance when co-evolution is used, in most cases. The cases in which performance remained comparable were those that had suitable operator probabilities of around 0.5; or were not particularly sensitive to operator probability anyway.

Why the drop in performance? A possible reason could be that the genetic algorithm was not able to evolve a suitable crossover probability. To see if this was the case plots of the evolved crossover probability against the number of

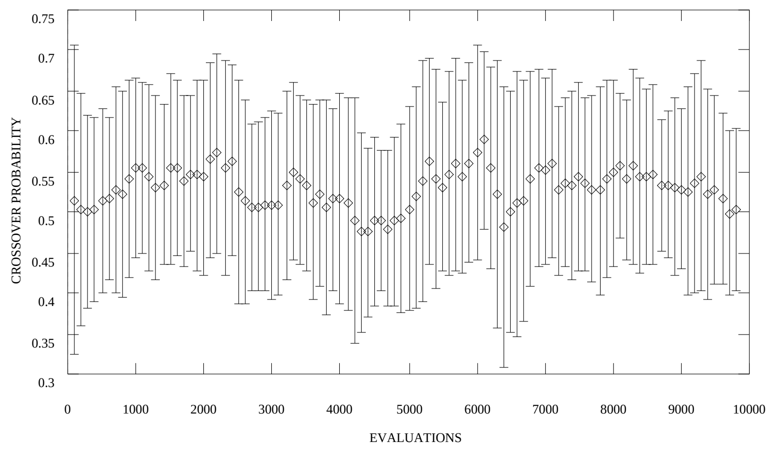

Figure 3: A Typical Plot When Strongly Disruptive Operators Were Used

evaluations made so far were obtained (the standard deviation of the genetic algorithm population at that time is given by the error bars). The plot shown in Figure 3 (for a generational GA attempting the max ones problem) is typical for all of the problems.

It can be clearly seen that the crossover probability remains around or near to 0.5, This is despite the fact that, for many problems, the suitable choice of crossover probability lies away from 0.5. This explains why performance is often degraded. The question is: why is no adaptation observed?

Plots of the *operator productivities* - the average improvement in fitness from parent to child that a given operator produces - were generated for a conventional genetic algorithm with crossover probability 0.5; so to see which operator was providing the most improvements at a given stage of a genetic algorithm run. Example plots are given later in this paper (Figures 6 and 7). In all cases crossover was consistently producing the greater improvements from parent to child.

Two possible reasons for non-adaptation can be proposed. First, the differences in productivity between operators are not great enough to result in sufficient selection pressure upon the encoded crossover probability. Alternatively, the operators being used are too disruptive — destroying any useful information that the genetic algorithm has found so far.

5.3 Co-evolution With Weakly Disruptive Operators

The hypothesis that adaptation was prevented from occurring because of the disruption caused by the operators can be tested by simply using less disruptive

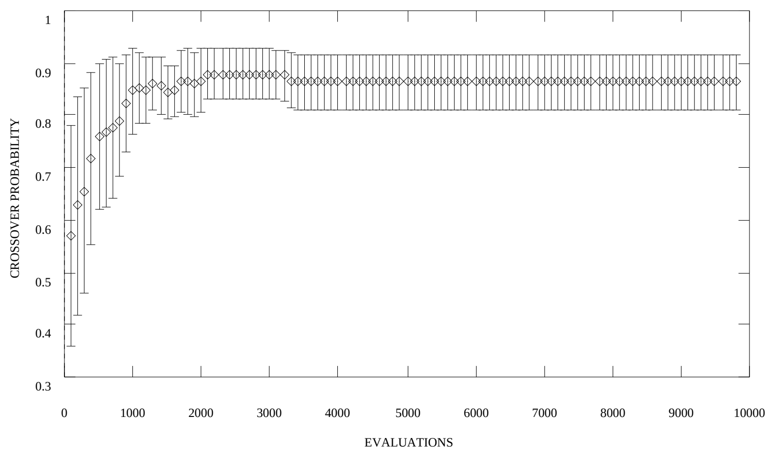

Figure 4: A Typical Plot When Weakly Disruptive Operators Were Used

operators. In some cases, adaptation was seen to occur — an example being for the counting ones problem when a steady state model was used (Figure 4).

Adaptation was found to occur in the following cases: flowshop sequencing and counting ones with a steady-state model only — albeit not reliably. In the other cases adaptation was not seen to occur. This lends support to the hypothesis that disruptive operators hinder adaptation. However, it is not clear why adaptation was observed for those two problems, but not the others; especially as for all the problems considered here, crossover was the operator with the highest productivity.

What effect does this have upon performance? As before, an exhaustive search was made: a genetic algorithm was run for co-evolution crossover probabilities 0.0 to 1.0 with steps of 0.05. Experiments were also performed using meta-learning. The results obtained are given in Tables 5 and 6. The performance of the genetic algorithm was found, in all cases, to be insensitive to the choice of co-evolution crossover probability. As before, the effect of using meta-learning was found to be insignificant.

Performance was still found to be degraded when compared to a genetic algorithm with fixed operator probabilities — even for the cases for which adaptation was observed. Examination of Figure 4 suggests a possible reason. It takes time to evolve a suitable crossover probability — too long in fact. By the time that the genetic algorithm has found a 'good' crossover probability, the population is quite close to the optimum anyway, allowing little time for any positive impact on performance to be made. This requires further investigation. Observing the effect upon genetic algorithm performance of initialising

Problem	Solution Quality	Evaluations Reqd.
Flowshop	2444 (73)	7946 (1795)
Max Ones	99.82 (0.43)	8070 (911)
Deceptive	288.56 (2.91)	5814 (2275)
Royal Road	35.20 (5.77)	6898 (1907)
Long Path	48687 (3441)	4768 (1870)

Table 5: Co-evolution With Weakly Disruptive Operators & Generational GA

Problem	Solution Quality	Evaluations Reqd.
Flowshop	2397 (71)	5572 (2000)
Max Ones	100.00 (0.00)	2475 (842)
Deceptive	289.00 (3.18)	2436 (1592)
Royal Road	31.68 (8.758)	2663 (2202)
Long Path	44509 (9753)	4933 (4067)

Table 6: Co-evolution with Weakly Disruptive Operators & Steady-State GA

the crossover probability closer to a known suitable value may provide useful information.

5.4 Co-evolution of Operator Parameters

In the literature thus far, co-evolution has been used to adapt operator parameters — specifically the mutation parameter. To this end, an investigation of the effect of evolving the operator parameters whilst the crossover probability remained fixed was performed. As before, an exhaustive search was made: a genetic algorithm was run for co-evolution crossover probability 0.0 to 1.0 with steps of 0.05, allowing the operator parameters to evolve. The results obtained for a genetic algorithm with fixed crossover probability and default operator parameters are given in Tables 7 and 8 (the crossover probabilities used are given in square brackets); these will be used as a basis for comparison.

Problem [p(Xover)]	Solution Quality	Evaluations
Max Ones [0.80]	99.96 (0.20)	7714 (1025)
Deceptive [0.05]	289.68 (2.41)	5960 (1965)
Royal Road [0.35]	35.52 (6.02)	7804 (1390)
Long Path [0.55]	48686 (3441)	5370 (2138)

Table 7: The Fixed Operator Probabilities (Generational) Used for Comparison

The results discussed earlier caution towards examining whether adaptation

Problem [p(Xover)]	Solution Quality	Evaluations
Max Ones [0.80]	100.00 (0.00)	2172 (702)
Deceptive [0.20]	289.12 (3.08)	3506 (2523)
Royal Road [0.95]	40.64 (7.65)	3786 (1741)
Long Path [0.20]	47312 (4765)	5129 (4088)

Table 8: The Fixed Operator Probabilities (Steady-State) Used for Comparison

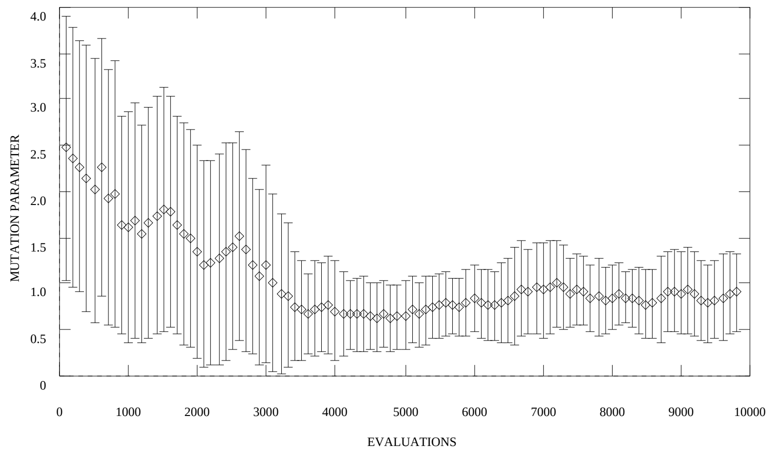

Figure 5: A Plot of Mutation Parameter For The Counting Ones Problem

does take place. Therefore plots of the evolved crossover probability against the number of evaluations made so far were obtained (Figure 5).

Adaptation does take place for both the counting ones problem (to $1/l$ — the theoretical optimum [Bäck, 1993]), and the long path problem. However this was only observed for a generational genetic algorithm. It is not apparent why — further investigation is needed.

No adaptation was observed at all for either the deceptive or royal road problems. In the case of the deceptive problem, this may be due to the fact that the theoretically optimum value for the mutation parameter ($3/l$) lies in the middle of the range of the encoded mutation parameter. An experiment where the mutation parameter is initialised towards one end of the range should resolve this question. For the royal road problem adaptation may be made difficult by the stepwise fitness function, a result of which would be to make information on mutation parameter performance intermittent.

The performance of the genetic algorithm was found, in all cases, to be insensitive to the choice of co-evolution crossover probability. When the quality

Problem	Solution Quality	Evaluations Reqd.
Max Ones	99.52 (0.64)	8438 (857)
Deceptive	289.12 (2.83)	5306 (1918)
Royal Road	31.36 (6.16)	6086 (2505)
Long Path	46906 (7342)	5892 (2461)

Table 9: Co-evolution of Operator Parameters with a Generational GA

Problem	Solution Quality	Evaluations Reqd.
Max Ones	100.00 (0.00)	2791 (1011)
Deceptive	289.32 (2.61)	2555 (2124)
Royal Road	29.76 (8.32)	2428 (1937)
Long Path	46375 (5523)	5671 (3768)

Table 10: Co-evolution of Operator Parameters with a Steady-State GA

of results (Tables 9 and 10) is examined for the cases for which adaptation took place, genetic algorithm performance was found to be degraded. As suggested earlier, this may be a result of the time it takes for the genetic algorithm to adapt — good choices of the operator parameters are more important earlier in the genetic algorithm run than later.

The speed to solution for the deceptive problem was improved in this case, presumably due to the mutation parameter lying initially close to the theoretically optimal value ($3/l$). In the case of the royal road problem, an exchange of decreased solution quality in favour of increased speed of search was observed, an effect of the increased mutation parameter.

In any case, a thorough study of the effect of the operator parameters upon genetic algorithm performance may resolve many of the questions raised here.

5.5 The Effectiveness of COBRA as a Function of The Probability Re-Ranking Interval

The previous study of COBRA [Corne et al., 1994] did not investigate the effect of varying G (the gap between re-ranking the operator probabilities). This investigation examined a range of values of G (from 200 to 2000 evaluations in steps of 200). The mean performance for each of these problems is given in Tables 11 and 12.

For three of the problems considered here, the choice of G had no significant effect upon genetic algorithm performance. The exceptions were the deceptive and long path problems.

To interpret these results it is necessary for us to know what COBRA is doing during the course of a genetic algorithm run. Therefore plots of the

	Mean Solution Quality With Re-Ranking Interval (G)							
Steady-State	200	400	600	800	1000	1200	1400	1600
Flowshop	2398	2397	2398	2401	2399	2397	2398	2392
Max Ones	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
Deceptive	287.7	287.5	287.4	288.0	288.0	287.9	288.1	288.3
Royal Road	40.00	39.36	40.80	40.48	40.32	40.48	40.80	40.64
Long Path	45675	43613	40568	41867	44704	44117	42998	43320
Generational	200	400	600	800	1000	1200	1400	1600
Flowshop	2445	2444	2444	2444	2444	2444	2444	2444
Max Ones	99.9	99.9	99.9	99.9	99.9	99.9	99.9	99.9
Deceptive	285.4	285.2	286.1	286.2	286.2	286.6	286.8	286.9
Royal Road	32.80	33.60	34.08	34.08	33.44	33.28	32.96	33.92
Long Path	47459	47843	48686	47887	47950	47696	48195	48687

Table 11: Solution Quality With Re-Ranking Interval (G)

operator productivities were then obtained for a conventional genetic algorithm with crossover probability 0.5, to see which operator was providing the most improvements at a given stage of a genetic algorithm run. The plots shown Figures 6 and 7 tend to be typical for the problems considered here — crossover is consistently the more productive operator.

	Mean Evaluations Reqd. With Re-Ranking Interval (G)							
Steady-State	200	400	600	800	1000	1200	1400	1600
Flowshop	5767	5979	5437	6081	5271	5862	5100	5993
Max Ones	2118	2139	2172	2172	2172	2172	2172	2172
Royal Road	3888	3741	3922	3910	4012	3845	3826	3800
Long Path	5120	6378	6326	6958	5575	5044	5305	5567
Generational	200	400	600	800	1000	1200	1400	1600
Flowshop	8098	8236	8236	8236	8236	8236	8236	8236
Max Ones	7714	7714	7714	7714	7714	7714	7714	7714
Deceptive	4322	4422	4194	4140	4244	4314	4160	4534
Royal Road	7664	7580	7728	7446	7616	7152	7026	7670
Long Path	5306	5344	5534	5418	5516	5536	5632	5434

Table 12: Evaluations Required With Re-Ranking Interval (G)

In the case of the deceptive problem, a trade-off was observed — low values of G corresponding to an increased speed of search, although at the expense of solution quality: high values of G favouring a higher quality of solution. The reason for this behaviour is closely linked to the effect of the crossover probability for the static genetic algorithm — high crossover probabilities lead the genetic algorithm to the deceptive optimum more quickly, leading to a faster speed of search whilst sacrificing quality. As crossover was consistently the more productive operator the genetic algorithm will assume a high crossover probability. For low G this will occur earlier in a genetic algorithm run and favour speed over quality.

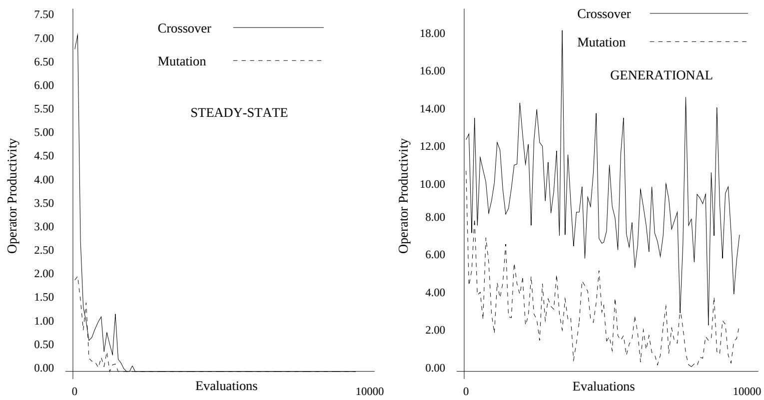

Figure 6: Operator Productivities For The Deceptive Problem

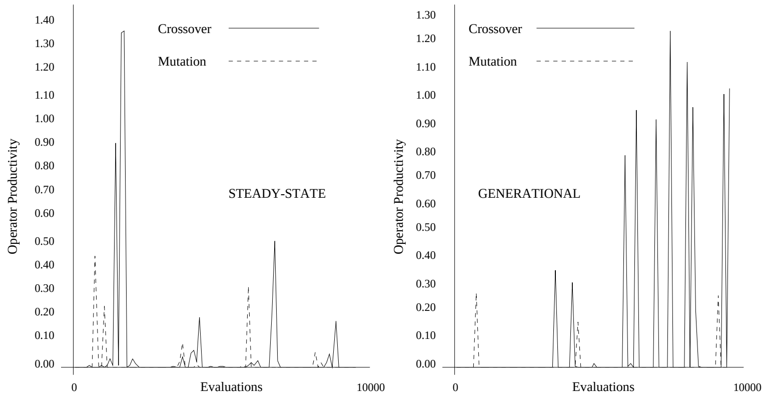

Figure 7: Operator Productivities For The Long Path Problem

For the long path problem, behaviour varied according to the population model used. When a generational model was used, solution quality was affected, rising with increased G. Presumably the larger samples prevent spurious re-ranking due to noise in the operator productivity information.

The trends observed with a steady-state genetic algorithm were quite the opposite: solution quality was degraded when compared to a conventional ge-

	Generational GA		Steady-State GA	
Problem	Static GA	GA w/COBRA	Static GA	GA w/COBRA
Flowshop	2441-2453	2438-2454	2387-2420	2396-2411
Max Ones	93.38-99.96	98.36-99.96	99.90-100.00	99.84-100.00
Deceptive	284.20-289.68	284.92-288.04	286.28-289.28	286.28-288.48
Royal Road	22.72-35.52	25.28-35.84	25.76-40.64	26.88-40.32
Long Path	35824-48687	41407-49177	28287-48025	40955-44723

Table 13: Fitness Obtained with Crossover Probabilities in Range 0.05-0.95

	Generational GA		Steady-State GA	
Problem	Static GA	GA w/COBRA	Static GA	GA w/COBRA
Flowshop	7714-8914	7824-8914	5064-7717	5045-6641
Max Ones	7714-9228	7714-8486	2172-5645	2172-6526
Deceptive	4484-6212	4244-6412	2192-4210	1979-3409
Royal Road	5626-7804	5596-7620	2876-3952	2607-4012
Long Path	3976-7848	4286-5702	5129-6736	5575-7437

Table 14: Evaluations Reqd. with Crossover Probabilities in Range 0.05-0.95

netic algorithm, declining further with increasing G. This is a result of crossover being the more productive operator, most of the time (Figure 7). Unfortunately the preferred crossover probability is low (0.2). Therefore COBRA will mostly adopt a high crossover probability with the effect of degrading solution quality. However, when G is small, there is a greater chance that 'spurious' re-rankings to a lower (ie. better) crossover probability will occur which reduces the adverse effect of COBRA somewhat.

5.6 The Effectiveness of COBRA as a Function of The Initial Operator Probabilities

The relationship between the performance of COBRA and the initial crossover probability was investigated. 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) was examined. Tables 13 and 14 display the range of performance attained, for all problems and population models, with and without the use of COBRA.

It was apparent, in most cases, that COBRA was less sensitive to the initial crossover probability, than a conventional genetic algorithm. COBRA appears to mitigate the effect of bad choices somewhat.

But does this affect performance? The best performance attainable appears, in most cases, to be unaffected by COBRA. The exceptions are: the deceptive problem (for both population models), and the long path problem. Each case will be discussed in turn.

In the case of the deceptive problem, the best solution quality attained is reduced, but with a corresponding increase in speed. This trade off was found to be controllable by the initial crossover probability. The reason for this is similar to the reason for the trends for G — the higher productivity of crossover means that a high crossover probability is adopted, which favours speed of search at the expense of quality.

The results for the long path problem, however, are disappointing. When a steady-state model was used, solution quality was significantly degraded. The reason for this is, as for the trend in G, largely due to the predominantly higher crossover productivity (Figure 7). This results in COBRA adopting a high crossover probability, hence effecting a reduction in solution quality.

6 Conclusions

In general it was established, by examining the effect of the crossover probability on a conventional genetic algorithm, that the choice of crossover probability is dependant upon the problem to be solved, the population model, and the performance measure used.

Conclusions will be discussed for each approach separately, with some final remarks to place them in a wider context.

6.1 The Co-evolutionary Approach

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.

Unsurprisingly, 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 genetic algorithm run appears to be important. 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 genetic algorithm run. Genetic algorithm performance was found to be sensitive to the operator parameters used.

It would appear that, at least for the problems used here, adaptation by co-evolution of operator settings is of little practical use. However, much of the previous work (e.g. [Bäck et al., 1991]) is more positive. Why is there a difference?

First, virtually all of the work performed previously examined the adaptation of operator parameters. The results obtained for operator parameters in this are more optimistic than for probabilities. Second, part of the reason also lies in the emphasis that some of the work has placed upon adaptation being an end in itself — with the implicit assumption that adaptation is a 'good thing'. This work strongly questions this assumption.

Finally, work by [Hinterding, 1995] looked at adapting operator parameters by co-evolution. The mutation parameter he adapted was the standard deviation of a Gaussian mutation, for problems that have a search space of real numbered parameters (in a similar fashion to the 'Evolution Strategy' community). The paper concluded that for many problems, the adaptation of the mutation parameter lead to improved genetic algorithm performance. This suggests that binary-encoded problems have less potential for this approach than real-coded problems; further investigation is required to confirm this.

6.2 Adaptation Using COBRA

It was established, by examining the effect of the crossover probability on a conventional genetic algorithm, that the choice of crossover probability is dependent upon the problem to be solved, the population model, and the performance measure used.

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 factors affecting performance.

No improvement in performance was found to occur when COBRA was being used, however the genetic algorithm 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 genetic algorithm runs to obtain equivalent performance. Also, the performance of COBRA (on the basis of the results for the deceptive problem) may well be a technique that favours speed of search, at the possible expense of quality.

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.

This suggests that operator productivity can be a poor basis with which to assign operator probabilities for a static genetic algorithm, and possibly misleading for COBRA. Operator productivity is but one factor to consider; others include the maintenance of diversity lost by processes such as drift and premature convergence by the mutation operator.

One class of problems are known to exhibit improved performance when COBRA is used: timetabling problems. In addition, recent work [Ross and Corne, 1995] has shed some light upon what makes timetabling problems easy or hard for a genetic algorithm. 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.

To summarise, COBRA has promise as a means to remove some of the parameter tuning that can bedevil genetic algorithm applications. But like many other such devices, there is the danger of being misled into a poor choice of operator probabilities by some problems.

6.3 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 genetic algorithm, and what is trying to be achieved.
- For adaptation to occur reliably, the adaptation mechanism should be separated from the main genetic algorithm 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 may be useful to view the assignment of operator probabilities as, in effect, providing knowledge (albeit in an implicit form) to the genetic algorithm 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.

7 Acknowledgements

Thanks to Dave Corne for his advice on COBRA. Thanks also to the Engineering and Physical Sciences Research Council for their support of Andrew Tuson via a studentship with reference 95306458.

References

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

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

[Bäck, 1992] Bäck, T. (1992). The Interaction of Mutation Rate, Selection and Self-Adaptation Within a Genetic Algorithm. In [Manner and Manderick, 1992].

[Bäck, 1993] Bäck, T. (1993). Optimal Mutation Rates In Genetic Search. In Forrest, S., editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*. San Mateo: Morgan Kaufmann.

[Bäck, 1996] Bäck, T. (1996). *Evolutionary Algorithms in Theory and Practice*. Oxford University Press.

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

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

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

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

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

[Forrest and Mitchell, 1993] Forrest, S. and Mitchell, M. (1993). Relative Building Block Fitness and the Building Block Hypothesis. In Whitely, L. D., editor, *Foundations of Genetic Algorithms 2*. San Mateo: Morgan Kaufmann.

[Goldberg, 1989] Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization & Machine Learning*. Reading: Addison Wesley.

[Goldberg et al., 1989] Goldberg, D., Korb, B., and Deb, K. (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3:493–530.

[Hesser and Männer, 1991] Hesser, J. and Männer, R. (1991). Towards an optimal mutation probability for genetic algorithms. In Schwefel, H. P. and Männer, R., 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. Springer-Verlag, Berlin, Germany.

[Hinterding, 1995] Hinterding, R. (1995). Representation and Self-adaption in Genetic Algorithms. In *Proceedings of the First Korea-Australia Joint Workshop on Evolutionary Computation*.

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

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

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

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

[Mühlenbein, 1992] Mühlenbein, H. (1992). How Genetic Algorithms Really Work: Mutation & Hillclimbing. In [Manner and Manderick, 1992], pages 15–26.

[Mühlenbein and Schlierkamp-Voosen, 1995] Mühlenbein, H. and Schlierkamp-Voosen, D. (1995). Anaylsis of Selection. Mutation and Recombination in Genetic Algorithms. In Banzhaf, W. and Eeckman, F. H., editors, *Evolution as a Computational Process*, pages 188–214. Springer Verlag.

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

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

[Ross and Corne, 1995] Ross, P. and Corne, D. (1995). The phase transition niche for evolutionary algorithms in timetabling. In *Proceedings of the First International Conference on the Theory and Practice of Automated Timetabling*, Edinburgh. Napier University.

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

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

[Tuson, 1995] Tuson, A. L. (1995). Adapting Operator Probabilities In Genetic Algorithms. Master's thesis, Department of Artificial Intelligence, Univeristy of Edinburgh.

[Whitley, 1993] Whitley, D. (1993). Genetic Reinforcement Learning for Neurocontrol Problems. *Machine Learning*, 13:259–284.