Adaptive Operator Scheduling in Evolutionary Algorithms

Master's Thesis by Wouter Boomsma

Department of Computer Science, University of Aarhus
December 2003

The image on the front page shows the cities in the *World Tour Problem*, an instance of the Traveling Salesman Problem containing 1,904,711 cities. The image was downloaded from `http://www.math.princeton.edu/tsp/`

Synopsis

In the last decades, *evolutionary algorithms* have become a popular method for optimisation. They are easily implemented and understood, and can provide reasonable solutions to many problems, even when little is known about a problem in advance. Experience has however shown that in order to achieve good results, it is often necessary to incorporate problem specific knowledge into the evolutionary algorithms, for instance in the form of heuristic operators.

Much research has been done on such operators, and through the years a large number of operators have been created for specific problem domains. When faced with a certain problem, it is often no longer necessary for the evolutionary programmer to design operators herself, since many are available in the literature. Deciding which of these operators to use is however no trivial task.

Based on the literature alone, it is difficult to make an assessment of the quality of an operator. Papers presenting new operators often contain results from experiments on well-known benchmark problems, but the results on these experiments are not necessarily a good indicator for the quality of an operator. Often very different evolutionary algorithms are used, making it difficult to isolate the effects of an operator based on the presented results.

Classically, evolutionary algorithms use 2 operators, *mutation* and *crossover*, and it might therefore seem natural to try and isolate a pair of the best operators from the literature. This could be done by manually comparing performances of all pairs of operators on a benchmark problem – a rather time-consuming process. Furthermore, several studies indicate that evolutionary algorithms often benefit from having more operators to their disposal. Some studies even indicate that the optimal operator setting might change during the course of the solution process. How can the best operator selection be found in this scenario?

One solution to this problem is *adaptive operator scheduling*. It is a technique that allows an evolutionary algorithm to automatically adjust which operators are used during the course of the solution process. Several different algorithms have been proposed and tested on a variety of problems. No thorough comparison of these methods has however been done. It is not clearly investigated whether certain of these algorithms perform better than others. Neither is it known whether adaptive operator scheduling works better on some problems than others. Finally, it is not even clear whether operator scheduling in general has much effect at all. Many papers presenting adaptive operator scheduling schemes compare their performance to algorithms with only one or two operators. Good results might be presented, but this can be as much an artifact of the large set of operators as to the scheduling mechanism itself.

In this thesis I try to shed light on some of these issues. A test-suite of adaptive operator scheduling schemes is implemented, partly consisting of existing algorithms, and partly of algorithms that were written specifically for this study. The performance of these operator scheduling schemes are compared on various problem instances from three problem domains: the Traveling Salesman Problem, Multiple Sequence Alignment and numerical optimisation. In all experiments the adaptive schemes are compared to a

non-adaptive variant that also has all operators to its disposal. This isolates the effects of operator scheduling from that of using a large set of operators.

The results of these experiments show that generally, the evolutionary algorithms seem to benefit from having a large number of operators to their disposal. The positive effects of operator scheduling are less pronounced. For certain problems, significant improvements were observed while for others, there was hardly any effect. Generally, the non-adaptive algorithm that chooses operators at random performs surprisingly well and will probably suffice for most purposes. The comparison between different operator scheduling schemes shows no great difference in performance between them. If operator scheduling is to be used at all, it seems that the best advice is to choose a scheduling method which is easily implemented. I present several of such algorithms in this thesis.

During the course of the different studies, several additional observations were made. Based on poor preliminary results on the Multiple Sequence Alignment problem, an investigation was done on a well-known evolutionary algorithm that uses operator scheduling on precisely this problem. While the authors to this algorithm claim that the operator scheduling technique is vital for its performance, the conducted experiments showed the opposite. Equally good results were achieved using a random choice of operators.

Another example concerns the fundamental assumption of operator scheduling: that good operators should be used frequently. In the study of numerical optimisation problems, results indicate that this observation might not hold in general. It seems that rewarding the best operators increases the greediness of the search, which can lead the algorithm to premature convergence.

Not a great many general conclusions can be drawn based on experiments on only three problem domains. There are however indications that operator scheduling generally does not have as great an effect as intuition might suggest.

Acknowledgements

I would like to thank the following people for their support during the writing of this thesis:

René Thomsen and Jakob Svaneborg Vesterstrøm for valuable discussions and ideas helping me through the different stages of this project.

Mikkel Ricky Christensen, whose `pedantic-mode` filtered out a great deal of errors and inconsistencies.

Thomas Mailund Jensen who contributed with helpful comments on style and content of the central chapters.

Ole Caprani for accepting to be my supervisor at short notice, and his relaxed attitude that helps put things into perspective.

Jacobus J. Boomsma for acting as my non-computer-science test group, and giving valuable comments along the way.

And finally, I would like to thank Katrine Lindegaard who has managed to keep spirits high, and forced me to take my mind off the thesis once in a while.

Contents

1. Introduction

One of the primary characteristics of the human race is our ability to solve problems[1]. This is naturally done with the help of tools, and in contemporary times computers are often the tool of choice.

When solving problems with computers it quite quickly becomes apparent that some problems are fundamentally harder to solve than others. While it for some problems is possible to design ingenious algorithms that solve the problem efficiently, for others it seems considerably harder, or even impossible to come up with any algorithms at all.

The theory of computational complexity make it possible to formalise the concepts of "easy" and "hard" problems and the distinction between them. Problems can be formally classified based on their complexity, and if a problem belongs the class of **NP**-*hard* problems, we know in advance that there is little hope of finding an efficient and exact algorithms for solving it[2]. Any exact algorithm for such a problem has an execution time exploding for increasing problem sizes, and is often useless for most practical purposes.

The search for alternative algorithms is thus justified. There is a demand for faster algorithms that do not necessarily produce the exact optimal solution, but in most cases provide solutions of sufficient quality. Such methods are called *heuristic algorithms* or *approximation algorithms*.

A wide variety of such algorithms have been proposed throughout the years. Some are designed to tackle specific problems while others can be applied more generally. An example of the latter is the class of *evolutionary algorithms* (EA). Inspired by natural selection in nature, these algorithms maintain a population of solutions in a generational manner, constantly creating new solutions by means of mutation and crossover operations, and letting only the best solutions survive to subsequent generations.

Evolutionary algorithms have some attractive qualities. They are easily understood, and often require little effort to implement. The design is modular in the sense that the basic algorithm can be extended with more advanced operators or be combined with alternative heuristic algorithms for increased performance. Evolutionary algorithms are especially valuable as initial "quick and dirty" investigations of a problem. Later, as knowledge is gained about the problem, either the algorithm can be extended with additional problem-specific operators, or an alternative specialised algorithm can be designed.

With the increasing popularity of evolutionary algorithms the amount of available operators for specific problem domains has increased. For many problems, it is no longer necessary to design operators yourself, since vast libraries of them are available in the

[1] The fact that we are also notoriously good at creating them, is another story.

[2] Assuming that $\mathbf{P} \neq \mathbf{NP}$.

literature. While this of course is a positive development, it also introduces a new dilemma: Given a problem, which of the available problem-specific operators should you use?

One solution is to conduct experiments with single-operator evolutionary algorithms to manually compare the effect of different operators. Alternatively, one could create a large pool of operators, and allow the EA to randomly choose an operator every time it is about to create a new individual. This approach however seems somewhat unsatisfactory. Some operators are bound to be better than other, and choosing them all with equal probability hardly seems like an optimal strategy. Another approach could be to assign a different probability setting to each of them, for instance based on observed performance in initial experiments. An even better approach could be to let the evolutionary algorithm determine the probability setting during execution. By starting the evolutionary algorithm with a uniform probability setting, and gradually adjusting the probabilities based on the current performance of the operators, the evolutionary algorithm is able to automatically find the best probability setting. Such an approach is called *adaptive operator scheduling*.

1.1. Motivation

Adaptive operator scheduling is an intuitively convincing method. No initial selection of operators is required, and the best operators are employed at every phase of the run. Intuition is however not always the best guide.

In 1989 Davis [12] presented one of the first operator scheduling schemes in evolutionary algorithms. Since then, several alternative methods have been proposed, and test results have been presented for different problem domains. These studies are not conclusive on the effects of operator scheduling. Some present significantly improved results, while others experience no improvements at all.

Based on these mixed results, it is reasonable to suspect that the success of adaptive operator scheduling is determined either by the problem domain or by the type of adaptive operator scheduling algorithm used. The literature contains little evidence to verify this. No thorough comparisons of different schemes on the same problems have been done, and often, any newly introduced method for operator scheduling is only tried out on a single problem. Furthermore, most of the methods have only been applied using a very limited number of operators, which might also be an issue that limits the performance.

This thesis is an attempt to illuminate these issues. A test suite of different methods of adaptive operator scheduling is implemented, and experiments are done on a number of problem instances from three problem domains: the Traveling Salesman Problem, Multiple Sequence Alignment and numerical optimisation.

1.2. Methods

The test suite of adaptive operator scheduling schemes used for my experiments consists of the adaptive operator scheduling algorithms of Davis [12] and Julstrom [39] plus a

number of new algorithms based on techniques such as competing subpopulation and gene-encoded parameters. A comparison of operator scheduling schemes is not complete without a comparison with the most obvious alternative: An approach where all operators have fixed, equal probabilities of being chosen. Such an algorithm is therefore also included in the test suite.

The thesis consists of 5 different studies, and they are included in the thesis in the order that they were conducted. Initially, experiments were done on the Traveling Salesman Problem (TSP). The first study was a preliminary investigation experimenting with the adaptive operator scheduling scheme of Davis [12]. After this initial investigation all the mentioned scheduling algorithms were implemented and a complete comparison between them was done on 6 instances of the Traveling Salesman Problem.

This study was followed by an equivalent one on the Multiple Sequence Alignment (MSA) problem. Again all scheduling methods were compared, this time on 5 MSA instances.

The effects of adaptive operator scheduling on the MSA problem proved to be rather insignificant. There is a well-known evolutionary algorithm for the MSA problem by the name of SAGA [52] that uses adaptive operator scheduling and claims that this is vital to the performance of the algorithm. This apparent inconsistency was explored by a thorough investigation of the SAGA algorithm. As expected, the effect of operator scheduling was negligible.

Finally a study was done on a number of numerical benchmark functions. Since the Differential Evolution (DE) algorithm [66] has been shown to perform very well on this problem domain, an effort was done to incorporate the DE variation operator in the set of operators used.

The two studies of the TSP problem and the investigation of the SAGA algorithm, were published as 3 papers in the proceedings of different conferences. For details see appendix.

1.3. Outline

In order to give some background on the field in general, the thesis starts with an introduction on optimisation algorithms, presenting various methods and defining necessary concepts. Subsequently there is a chapter on adaptive operator scheduling, which also presents the implemented algorithms. The rest of the thesis consists of the studies described above and a conclusion wrapping it all up.

Chapter 2 introduces basic concepts and presents a number of classic search techniques. The chapter is concluded with a section on modern heuristics.

Chapter 3 describes the basic terminology of adaptive operator scheduling and adaptation in general. Furthermore, it contains a list of previous applications of adaptive operator scheduling and describes the test suite of implemented algorithms.

Chapter 4 describes the Traveling Salesman problem and various algorithms for solving

it. The chapter also contains the two studies of adaptive operator scheduling on the TSP.

Chapter 5 defines the Multiple Sequence Alignment algorithm and lists a number of algorithms for solving it. Subsequently, the two studies on the MSA problem are presented.

Chapter 6 contains the investigation of adaptive operator scheduling on a selection of numerical optimisation benchmark functions.

Chapter 7 contains of a discussion of the results obtained in the various studies of this thesis and is concluded with a description of possible future work in this area of research.

1.4. List of Abbreviations

ADOPP	Adaptive Operator Probabilities
AOSGA	Adaptive Operator Scheduling Genetic Algorithm
COFFEE	Consistency-based Objective Function for Alignment Evaluation
CS	Column Score
DE	Differential Evolution
EA	Evolutionary Algorithm
ES	Evolution Strategies
GA	Genetic Algorithm
GEP	Gap Extension Penalty
GOP	Gap Opening Penalty
GP	Genetic Programming
MSA	Multiple Sequence Alignment
MSAEA	Multiple Sequence Alignment Evolutionary Algorithm
SAGA	Sequence Alignment by Genetic Algorithm
SIDEA	Self-adaptive Individual-level Differential Evolution Algorithm
SP	Sum-of-Pairs
SPDEA	Self-adaptive Population-level Differential Evolution Algorithm
SIESA	Self-adaptive Individual-level Evolution Strategy Algorithm
TSP	Traveling Salesman Problem

2. Optimisation Algorithms

This thesis is primarily concerned with some specific issues about a certain type of optimisation algorithm, the *evolutionary algorithm*. To provide a broader perspective I will start with a short overview of some of the other methods available, thereby introducing terminology that will be of use later.

The objective of an optimisation problem is to find the best solution of all possible solutions. Many real-world problems are naturally stated as optimisation problems. Finding the energy-optimal 3-D structure of a protein, determining the most effective time-table for the teachers of a school, or more generally, discovering the optimal setting for a given set of parameters are examples of such problems.

The collection of all possible solutions for a given problem can be regarded as defining a search space, and for this reason optimisation algorithms are often referred to as search algorithms. Some of the search spaces are more easily visualised than others. For continuous numerical problems, there is an intuitive way of illustrating a search process. Consider for instance the search space for the 2-dimensional case of the Rastrigin benchmark function (figure 2.1). The search process corresponds to movements on the fitness surface defined by the function

$$z = x^2 - 10 + 10 * \cos(2\pi x) + y^2 - 10 + 10 * \cos(2\pi y) \tag{2.1}$$

One natural way to move around in this space would be to take small steps by making adjustment to x and y, respectively. We can visualise how these small steps correspond to different values of z in figure 2.1.

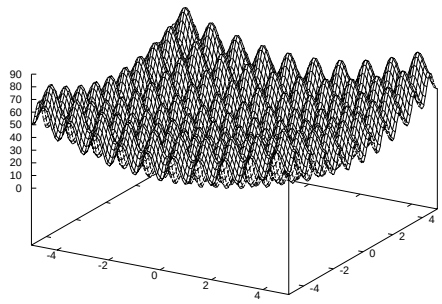


Figure 2.1.: The search space of the 2-dimensional case of the Rastrigin function.

As the number of dimensions of numerical problems increase, it becomes harder to imagine the shape of the search landscape. For combinatorial problems this effect is even more pronounced. As an example we can consider the Traveling Salesman Problem. This problem consists of visiting a list of n cities while minimising the overall travelled distance[1]. Any solution to this problem is an ordering of the cities, which can be represented as a permutation of the numbers from 1 to n, where each number represents a city. Although it difficult to imagine what the corresponding search space might look like, we can mathematically define what a small step in the search space should be. In the case of the Traveling Salesman Problem one such move could be the swapping two cities (figure 2.2). Even though such a move is not easily visualised, it seems like a basic move in the search space of this problem.

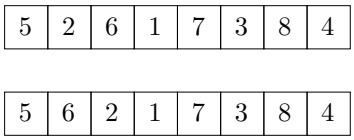


Figure 2.2.: Two permutations of the integers from 1 to 8. Solutions to the Traveling Salesman Problem are often represented as such a permutation, and a single swap of two cities constitutes a "small" step in the search space for this problem.

For a given point in the search space, one type of operation can result in many different new solutions (e.g. many different pairs of cities can be swapped). The set of all these solutions is called the *neighbourhood* of the current solution.

In the following sections I will present various schemes for solving optimisation problems. I will start with the most naive of these search algorithms, and then progress to some more advanced variants. Further information on the different schemes can be found in Michalewicz and Fogel's "How to Solve It" [50] and in Papadimitriou and Steiglitz "Combinatorial Optimization Algorithms and Complexity" [56].

2.1. Traditional Methods

2.1.1. Exhaustive Search

Given some search space, the most natural thought might be to just try all solutions and subsequently pick the best one. This is the idea behind *exhaustive search*. All this method requires is that solutions are examined in a systematic way so that we avoid solving the same problem twice and ensure that we know when we are finished. This is called an *enumeration* of the solutions, and exhaustive algorithms are correspondingly also known as *enumerative algorithms*.

For most problem instances of realistic size, it is completely unrealistic to do an exhaustive search. A problem such as the Traveling Salesman Problem with n cities has a search space containing $(n-1)!/2$ solutions. Already with instances of 50 cities, an

[1] A more precise definition of the Traveling Salesman Problem is given in section 4.

exhaustive algorithm would be enumerating a set of 10^{62} solutions. Needless to say, this is not something that one can afford waiting for.

2.1.2. Local Search

Local search is basically a version of exhaustive search that only focuses on a limited area of the search space. More precisely, it chooses one random solution from the search space, and investigates the neighbourhood of this solution for possible improvements. When an improved solution is found, it becomes the new current solution. This procedure continues as long as improvements can be made.

Hill-climbing techniques belong to the class of local search algorithms. These algorithms consistently replace the current solution with the best of its neighbours if this neighbour is a better solution. For 2-dimensional numerical optimisation problems as the one described above this strategy thus follows a trail upwards to the nearest hill-top (or valley if it is a minimisation problem). There are of course no guarantees that this is in fact the overall best solution. When looking at figure 2.1 it is obvious that such a strategy is likely to get stuck in some local optimum. Often, several runs with different initial values are therefore run in the hope of finding one of the better local optima.

2.1.3. Divide and Conquer

An obvious thought when faced with a complicated problem is to try to split it into smaller, more manageable problems. One can then hope that these smaller problems are easier to solve and that their solutions can be combined to a solution for the original problem. There are not a great many problems that can easily be partitioned and combined like this, but for the ones that can, *divide and conquer* presents an elegant solution.

Normally the subproblems are similar to the original problem, and divide and conquer is therefore often naturally implemented as a recursive algorithm[2]. One of the most simple examples of this type of algorithm is the binary search algorithm. Given a sorted list of n elements, the problem is to find the position of an element with value v. The problem can be split in two by separately considering the first and the second half of the list. Each of these lists represent a smaller version of the original problem. In the case of binary search, it turns out that we don't even have to solve both of the smaller problems. Since the lists are sorted, we can easily decide in which of the two lists our element resides, and therefore ignore the other list. This dividing strategy continues until we have a list of size 1, containing the element we were looking for (provided it is present in the list). See figure 2.3.

2.1.4. Branch-and-Bound

The *branch-and-bound* technique is basically a critical enumeration of the search space. It enumerates, but constantly tries to rule out parts of the search space that cannot

[2] An algorithm that repeatedly applies itself on smaller or simpler instances of the same problem.

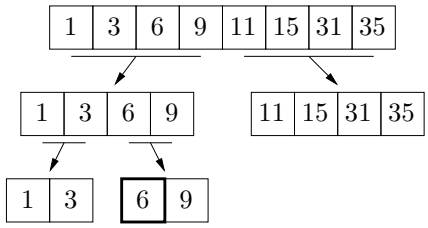


Figure 2.3.: A binary search for the element with value 6. After each division of the list, the edge-values of the new lists are checked to determine which of them contains the element of interest.

contain the best solution. Two requirements must be fulfilled for branch-and-bound to be applicable. First, it must be possible for the search space to be meaningfully partitioned into disjoint sets (recursively). Secondly, there must be a way to calculate lower bounds for the solutions of the the subsets (for minimisation problems).

The search proceeds through as a series of branch operations, where the problem is divided into a number of subproblems. This process can be described with a search tree where the root corresponds to the original problem and each node corresponds to some subproblem (figure 2.4). As the tree is filled out, we will gradually learn something about the search space. In particular, some subproblems will be sufficiently small that they need not be divided further and can be solved directly. Little by little we thus produce solutions to the problem, and the best of these solutions can be considered as the current best candidate for the global solution. Such candidates make it possible to rule out subtrees of our search process: If we at some point calculate a lower bound for a subproblem, and this lower bound turns out to be larger than our current candidate solution, it has little meaning to further pursue this subtree. Trees are thus pruned, and in many cases, this gives a substantial increase in performance.

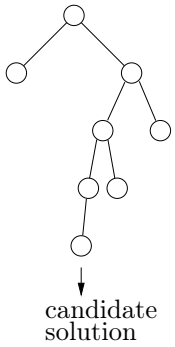


Figure 2.4.: A binary tree describing the solution process of the branch-and-bound technique.

2.1.5. Dynamic Programming

Dynamic programming is another attempt at a more intelligent enumeration of the search space. Again, we consider problems which are naturally composed into smaller instances. The key observation is that for certain problems, different global solutions often involve solving the same subproblems. The enumeration of all solution thus leads to a massive amount of re-computation. If solutions to subproblems are instead stored, significant amounts of computation time can be saved. This can be achieved by formulating the solution process as a recursion, and then storing the solutions for different stages of this recursion in a table. The table is often filled out as an initial phase to the algorithm, such that the solution can simply be found by traversing backwards through the table.

Dynamic programming is not easily explained or understood without an example, and I therefore refer to section 5.2.1, where a dynamic programming approach to the sequence alignment problem is presented.

2.2. Modern Heuristics

Despite the fact that divide and conquer, branch-and-bound and dynamic programming are significant improvements over exhaustive search, their time-complexity if often still too high. Consider for instance the Traveling Salesman Problem with n cities. A simple exhaustive search has to consider $(n-1)!/2$ solutions. This can be reduced by a dynamic programming approach [56] to an algorithm with time complexity $O(n^2 2^n)$, which is a great improvement, but still hardly satisfactory for problems of realistic size. The hill-climbing algorithm is much faster, but often ends in a local optimum of low quality, even if it is repeated several times. It is this issue of *premature convergence* that is the basis for a class of algorithms that often go under the name of *modern heuristics*. Basically, they are advanced local search strategies, for which the avoidance of premature convergence plays a cental role.

2.2.1. Simulated Annealing

The hill-climbing strategy described in section 2.1.2 picked a single solution in the search space and then continuously tried to improve it by picking better solutions in its neighbourhood. It is the very greedy nature of this algorithm that leads it to local optima. It can never go downhill, and will always go to the nearest top (for maximisation problems).

The *simulated annealing* algorithm uses a similar approach, but occasionally accepts solutions that are worse than the current. The probability that it takes such a step decreases with time. More precisely, the probability p for acceptance is

$$p = e^{\frac{\text{eval}(s_c)-\text{eval}(s_n)}{T}} \tag{2.2}$$

where s_c is the current solution and s_n is the new one. The value T decreases according to some function $g(T)$, which is often chosen to be exponentially decreasing with the number of iterations. The initial value of T can be determined empirically once the algorithm is implemented.

2.2.2. Tabu-search

The *Tabu-search algorithm* is another attempt to escape the local optima in the search space. The rationale is again that the search should primarily improve the solution, but occasionally be allowed to pass through regions of inferior solutions. Tabu-search is constructed around the idea that this strategy alone is often not sufficient to escape from local optima. If the current solution is at a local optimum, and the search is allowed to make moves that reduce the quality of a solution, it will often make only a few such steps, and then find some path back to the local optimum it came from. It thus risks oscillating around this peak endlessly, while never managing to escape to a better optimum.

The remedy to this problem is to prohibit the search to repeat moves that have recently been made. More precisely, the tabu-search is often designed to prevent repetition of the basic actions of a variation operator. An example is the permutation representation and the basic swap operation mentioned above (figure 2.2). It is on the swap operation between specific cities than a restriction can be imposed. Every time a certain swap between to cities is executed, it can be remembered and prohibited for some fixed number of future moves. This means that the search cannot retrace its own steps and is forced to consider other options. Often, this algorithm is implemented so that it in fact considers all neighbours, including the prohibited ones. Under normal circumstances it chooses a non-tabu move, but if one of the candidate solutions is particularly good, an exception can be made.

The general description of Tabu-search is accredited to various papers by Fred Glover in the late seventies and eighties [19, 20, 21, 22]. However, already in 1973 Lin and Kernighan used a similar approach in their well-known Lin-Kernighan algorithm [63] for the Traveling Salesman Problem. This algorithm will be presented in detail in section 4.1.

2.2.3. Evolutionary Algorithms

All the methods considered so far work with one current solution, which is then repeatedly changed, often in the direction that gives the highest gain. The main idea of *evolutionary algorithms* (EAs) is to generalise this basic local search scheme to consider a number of solutions simultaneously. In isolation, it is not clear that this idea will provide better solutions. Considering n points simultaneously instead of one is essentially the same as repeating a local search n times. There is, however, one advantage: By considering n solutions simultaneously we potentially have more knowledge of the search space available to the algorithm. This knowledge can be exploited by combining the information of several different solutions. Operators that create a new solution based on several of the current solutions are called *recombination* operators or *crossover* operators. In the field of evolutionary algorithms, operators that use only a single solution are often called *mutation* operators.

Evolutionary algorithms also introduce other new ideas and terminology. Instead of regarding the solution process as n individual local search processes, it is viewed as a single process where a *population* of *individuals* is maintained. Each of these individuals

represents a solution to the problem. These solutions are sometimes referred to as the *genotype* of an individual, and the components of the solution (for instance elements in a vector) are sometimes called *genes*. With certain intervals a Darwinian inspired *selection* process weeds out the worst of the individuals and lets the best survive. These intervals are called *generations*, and the quality of an individual is often referred to as its *fitness*. The terms *parent* and *child* are used to denote the relationship between input and output to a variation operator. Normally, each generation consist of a mutation phase, a crossover phase and a selection phase, but not necessarily in this order. Each phase is traditionally associated with a single operator.

Several techniques exist for the exact design of an evolutionary algorithm. Often, evolutionary algorithms differ in their selection phase. Especially the amounts of new individuals that are created, and their incorporation into the population give rise to different implementations. The selection operation is often described by a pair of variables μ and λ, where μ denotes the number of parents used in each generation, and λ denotes the amount of created offspring. A $(\mu + \lambda)$ strategy creates λ new individuals from μ parents and then selects among both parents and offspring to determine the individuals for the next generation. A (μ, λ) selection strategy means that only the offspring are considered as possible candidates for the next generation. A technique that is sometimes used in selection is *elitism*, meaning that some of the best solutions in a population are guaranteed to survive the selection phase. A special case is the *steady-state* evolutionary algorithm, where only a single individual is replaced in every generation. This individual is compared to the worst individual in the population and the best of the two survives to the next generation.

Many of the above ideas have been developed independently several times in different parts of the world. This resulted in different flavours of evolutionary algorithms. The most well-known of these are Fogel, Owens and Walsh' *Evolutionary Programming* [18] from 1966, Rechenberg's *Evolution Strategies* [57, 58] from 1965 and Holland's *Genetic Algorithms* [32] from 1975. While the first two were largely focused on solving a particular type of problem, Holland pursued a more general effort to model adaptation methods from nature. Holland was also the first to use a crossover operation. The two other methods had relied solely on mutation to provide the necessary variation.

The last decade has blurred the distinctions between these different approaches. Most of the characteristic ideas from one method have been adopted by others and today, it is often hard to label algorithms as belonging to one of these classic families. Most people now refer to these approaches under the collective name of *evolutionary algorithms* or *evolutionary computation*.

Operators

As mentioned in the previous sections, all local search algorithms require certain variation operators in order to create a neighbourhood of new candidate solutions.

For numerical problems of two dimensions we can visualise the search space, and intuitively design operators that can subsequently be generalised to higher dimensions. If we, for instance, want to design an operator that takes small steps in the search space,

this can be done by adding a random value to all entries in the vector representing this individual's solution. Various possibilities exist for the distribution of this random value. Many choose a normal distribution (Gaussian mutation), but other schemes have been proposed.

For the crossover operator we need some method of combining two solutions. If these are both vectors of length n, one way of achieving this would be to choose some *cut-point* i in the vector, and construct the new solution by taking all elements up to i from the first parent, and all elements beyond i from the second parent. This method is called *1-point* crossover. It can easily be generalised to *k-point* crossover, where k cut-points are chosen (figure 2.5). These operators can actually produce two children by interchanging the parents. In some implementations both these children are considered. In others, only the best child is kept for the next generation.

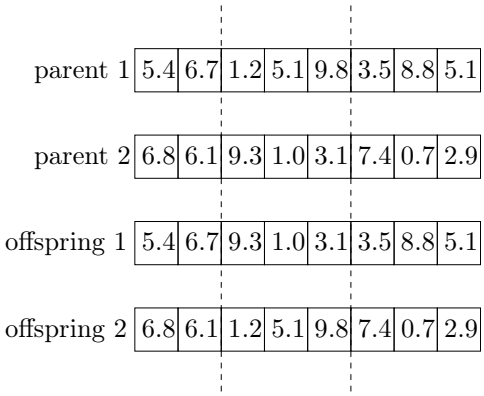


Figure 2.5.: A 2-point crossover operation. Note that the k-point operator naturally produces two children.

An alternative to the k-point strategy is the so-called *uniform* crossover. For each element in the child vector, the operator chooses randomly between the elements at the this position in the two parents (figure 2.6).

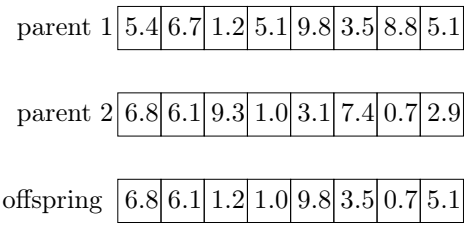


Figure 2.6.: An operation of the uniform crossover.

When entering the domain of combinatorial problems, the issue of operator design becomes more complicated. Operators depend heavily on the representation used. If we instead of the vector of n real-valued elements consider a permutation of n elements, none of the above variation operators would work, and other variation operators must be

designed. As mentioned, for the permutation representation (figure 2.2) a mutation move could be to swap two elements. Experience however shows that often more sophisticated operations are needed to achieve satisfactory results.

Much research is done on evolutionary algorithms on all kinds of problem domains and with variety of different representations. Accordingly, also the design of operators has received much attention. This effect is so pronounced, that for many problems a long list of potential operators exists. For the evolutionary programmer, this is a positive development. Instead of being forced to design operators herself, she can choose among the operators that are already available for the problem at hand. But how should she make her selection?

Often the literature contains no clear examples of the performance of a specific operator, but only of the absolute performance of an evolutionary algorithm using it. Since so many variations in the design of evolutionary algorithms exist, it is therefore hard to compare operators based on the results in their original descriptions. A one-to-one manual comparison between the operators is time consuming, and furthermore, certain studies indicate that it might be beneficial to use more than just one mutation and one crossover operator. There are even some indications that the optimal choice of operators is not fixed throughout the solution process. Some operators primarily focus on *exploration*, taking large steps in the search space and could thus be good for an initial location of the area of interest. Other operators are of a more *exploitative* nature, using smaller steps that are appropriate for fine-tuning the solutions in the final stages of the solution process. But how can one determine which is which, and when to change strategy during the course of a solution process?

One way of solving this problem is by letting the evolutionary algorithm automatically determine the optimal choice of operators throughout the course of the solution process. Methods to do this will be investigated in detail in chapter 3.

2.2.4. Differential Evolution

As a last remark in this section, I would like to mention the Differential Evolution algorithm by Storn and Price [66, 67]. Presented in 1995, this algorithm is one of the more recent developments in the field of evolutionary computation. Despite its name it bears little resemblance to any evolutionary process found in nature. It is, however, a remarkably simple algorithm that gives impressive results on numerical optimisation problems. It maintains a population of n individuals, and uses only a single variance operation which contains elements from the mutation, crossover and selection operations described in the previous section.

In each generation we create a new, empty population p_n, which we will subsequently fill out one individual at a time. Each individual is represented as a vector with real-valued entries. For each individual $p_n[i]$, we randomly choose three parents $\vec{x}_1$, $\vec{x}_2$ and $\vec{x}_3$ in the old population p_o. From these three parents, a temporary individual $\vec{x}'$ is created

$$\vec{x}' = \vec{x}_1 + F * (\vec{x}_3 - \vec{x}_2) \tag{2.3}$$

where the value F is a parameter of the algorithm typically set to 0.5.

The temporary individual x' is now combined with the individual $p_o[i]$ to create the individual $p_n[i]$. Different strategies for this recombination exist and I will only present one of them here. For more information see one the original papers on the algorithm [66, 67] or visit the Internet page (`http://www.icsi.berkeley.edu/~storn/code.html`).

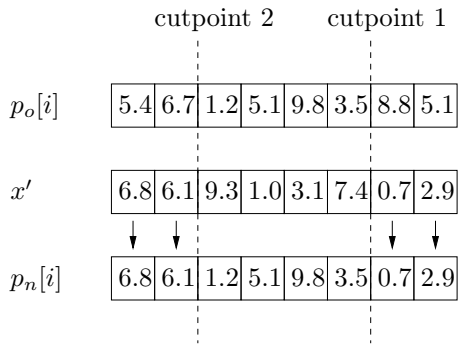


Figure 2.7.: The crossover operator for Differential Evolution. The index for cut-point 2 is determined as the first index for which a randomly chosen value is below the parameter CR. Notice that the cut-points 1 and 2 do not necessarily appear in that order.

The recombination operation is basically a 2-point crossover. Initially, $p_n[i]$ is set to $p_o[i]$ and the first cut-point is chosen at some random index. The second crosspoint is however not determined at this point. Instead, prior to any transfer from x' to $p_n[i]$, we choose a uniformly distributed random value between 0 and 1. Only as long as this value is smaller than a parameter CR do we make the transfer. The index where the random value exceeds CR for the first time thus defines the second cut-point. This crossover strategy is called the *exponentialcrossover* (see figure 2.7).

When $p_n[i]$ is successfully completed, its fitness value is compared to that of $p_o[i]$. If this new candidate for the i'th position in the population is worse than the old candidate, it is deleted and $p_o[i]$ is allowed to survive to the next generation.

The Differential Evolution algorithm lies somewhere in between the normal evolutionary algorithms and the classic local search strategies. A population of individuals is used, but the population is basically treated as a set of individual local search strategies. There is no selection phase, and each new solution only has to compete with the previous best solution at that position in the population. The population structure is however exploited by the somewhat mysterious variation operator, where three different parents are involved. The motivation for this design can be found in the original description [66]. For our purposes, an observation on the operator's behaviour will suffice: By taking the vectorial difference between two solutions and adding it to a third, the step size of the algorithm is automatically adjusted. In the initial phase the average distances between individuals are large, and the variation operator thus makes large adjustments. Gradually as the search progresses, differences will become smaller, and the step size is reduced correspondingly.

Personal experience has shown this algorithm to outperform any other evolutionary algorithm on all continuous optimisation problems that I have tried. It is robust and

seems to be remarkably good at avoiding local optima. I can only recommend that this algorithm is given a chance whenever a numerical optimisation problem needs to be solved. In section 6 I will show an application of this algorithm on a small selection of problems.

3. Adaptive Operator Scheduling

3.1. Introduction

The design of an Evolutionary Algorithm naturally induces a number of parameters. In a way, these parameters represent the uncertainty present within the design. When solving a specific problem, a setting for these parameters must be found, and it is vital to the performance of the algorithm that these settings are well-chosen.

In the 1970's and 1980's systematic investigations were done to determine the general best setting for EA's over a large amount of problems [14, 25]. These efforts must be seen in the light that in those days, a certain type of EA dominated the field to such degree, that an investigation of the general best parameter settings seemed meaningful. Today, it is generally accepted that best settings vary for different problems. Furthermore, many different evolutionary schemes are currently being used, and knowledge about good parameter settings cannot be transferred between them. Parameters therefore need to be tuned for each new algorithm on each new problem. This is often done in a rather ad hoc manner, perhaps tuning one parameter at a time. In most cases, evolutionary algorithms are thus run with suboptimal parameter settings. Even if a thorough investigation of alternative parameter settings is done, it is assumed that the optimal parameter setting remains the same during the course of the evolutionary process. No studies have however indicated that this is generally the case.

3.1.1. Methods of adaptation

Adaptive parameter control provides a compelling solution to this problem. The idea is that parameters are tuned while the algorithm solves the problem. Various methods of adaptation have been proposed and applied to different types of parameters. The terminology in the field can, however, be somewhat confusing. Especially the concepts of *adaptation* and *self-adaptation* are used rather indiscriminately in the literature. Several survey papers have tried to restore some order in the field by introducing classifications of adaptation schemes, and thus clear definitions of the different concepts. Although there is a significant overlap among the different classification schemes, some disagreement about the terminology has remained. I will shortly summarise the different schemes below.

The first general classification was proposed by Angeline in 1995 [1] and was based on two criteria: (1) the level at which adaptation is applied and (2) the nature of the update rules. The possible values for each of these criteria are given in table 3.1.

The *level of adaptation* specifies the scope of the parameters. For *population-level* adaptation, the same parameters apply to all individuals in the population. *Individual-level*

Criteria	Values
Level of adaptation	Population-level
	Individual-level
	Component-level
Nature of update-rule	Absolute
	Empirical (self-adaptive)

Table 3.1.: Classification scheme (Angeline) [1].

adaptation adjusts parameters "locally", and thus allows differences between individuals. Finally, *component-level* adaption specifies parameters for each component in the genotype of an individual. As an example, consider a mutation-rate parameter, i.e. a parameter determining the probability that mutations will take place. This parameter can actually exist on all three levels. At the population-level, the parameter specifies one global mutation probability. At the individual-level, it implies that each individual has an independent mutation rate, while at the component-level we can have mutation rates specifying the probability of mutation for each gene in the genotype.

Update-rules are either *absolute* or *empirical*. Algorithms with absolute update-rules use fixed guidelines to update parameter settings based on the current state of the solution process. These guidelines may be based on the current diversity of the population or the number of times that certain operators have provided good solutions. Self-adaptive schemes do not use any such explicit measure. Instead, the selection pressure already present in the evolutionary process is used to let better parameter settings evolve. This is often done by incorporating parameter settings in the genome of the individuals. The underlying assumption is that well-performing individuals often have a good parameter setting, and since the selection pressure favours individuals with high fitness, it will therefore automatically favour good parameter settings.

Some years later, Smith and Fogarty [64] introduced a classification scheme that can be regarded as an extension to Angeline's. It incorporates the *target of adaptation* as a class, distinguishing for instance whether adaptation is done on parameters or on operators. This produces a classification consisting of three classes: The target of adaptation, the scope of adaptation (i.e. the adaptation-level of table 3.1) and the basis for adaptation (i.e. the Nature of update-rule of table 3.1).

Eiben, Hinterding and Michalewicz have produced two papers on the classification of adaptive methods [30, 24]. The first, from 1997, is another extension of Angeline, where an environment-level is introduced as a specific level of adaptation, and which contains a finer devision of the update-rule class (deterministic, adaptive and self-adaptive). In their definition, deterministic update-rules use fixed guidelines to adapt a parameter independent of the current state of the solution process. An example would be a mutation operator that decreases its step size over time. The Adaptive and self-adaptive classes are defined as Angeline's absolute and empirical classes, respectively.

Apparently not quite satisfied with this framework, Eiben, Hinterding and Michalewicz presented a new classification scheme in 2000. Here the scope of adaptation is abandoned

Criteria	Values
Scope of adaptation	Population
	Individual
	Component
Basis of adaptation	Absolute
	Empirical (self-adaptive)
Target of adaptation	Parameters
	Operators
	...

Table 3.2.: Classification scheme (Smith and Fogarty) [64].

as a class for itself. This decision is based on the observation that for certain algorithms, the adaptation scope is cannot be well-defined, since adaptation works simultaneously on several different levels.

Criteria	Values
Scope of adaptation	Environment
	Population
	Individual
	Component
Types of adaptation	Deterministic
	Adaptive
	Self-adaptive

Table 3.3.: Classification scheme (Eiben, Hinterding and Michalewicz) [24].

In this thesis I focus on the adaptation of operator probabilities. With the target of adaptation thus being fixed, I found it natural to describe the different methods in the light of a classification based on two criteria: (1) the scope of adaptation and (2) the types of adaptation. This classification is reminiscent of Angeline's original classification. I will however use some of the terminology of the later classification schemes.

3.2. Adaptive Operator Scheduling

In recent years the field of evolutionary computation has grown rapidly. A massive amount of papers are produced studying a variety of different applications. Many of these papers present improved EAs for certain specific problem domains. One aspect of this research is the search for new and better operators for specific problem domains. With time, the number of available problem-specific operators increases steadily.

Especially for combinatorial problems, it seems that this is an important direction of research. Even though EAs are often promoted as general problem solving methods that require little knowledge about the problem, this hardly seems to hold for combinatorial

problems. Often, the design of the representation alone rules out the use of standard EA operators, and forces the programmer to consider alternatives. In order to achieve satisfactory results, experience shows that it is often necessary to incorporate significant amounts of problem specific knowledge into these operators. The possibilities for creating such heuristic operators are endless.

For an increasing amount of problem domains, the evolutionary programmer is thus faced with a new problem. Given a problem to solve, a selection among the set of available operators must be made. Given their heuristic nature, the performance of these operators often varies across different problems in the domain. For instance, certain operators might fail to scale up gracefully, becoming computationally infeasible for larger problem instances. Although the literature might provide comparisons between operators on specific problem instances, this generally does not establish which operator is best for the new problem instance at hand.

An elaborate manual comparison of all operators would provide an assessment of the quality of the operators, but does not exploit the fact that the usefulness of the operators often changes during the course of a single run. Furthermore, it does not take into account that dependencies between operators can exist and that through interaction multiple operators might provide better results than single operators.

Adaptive operator scheduling is a method to avoid the initial process of selecting operators. Instead, all operators are made available to the EA, which then adjusts the frequency of use of each of them depending on their performance.

3.2.1. Previous studies

Below is a selection of studies on adaptive operator scheduling that have been conducted since Lawrence Davis' paper in 1989, which is by many considered to be a pioneering paper on the subject. The list is not complete, but I feel that it is representative for the field in general and gives an impression of the different approaches that have been tried so far.

1989 Lawrence Davis [12] presented an evolutionary algorithm with five operators that were adapted during the course of the evolutionary process (for details see section 3.2.2). It was applied to a simple hill-climbing problem, and shown to have performance slightly interior to an algorithm where operator probabilities had been manually tuned. The algorithm was also applied to the training of neural networks, where it produced good results.

1994 Schlierkamp-Voosen and Mühlenbein [61] introduced an evolutionary algorithm that uses several populations instead of one. Each represents a different strategy and individuals are gradually migrated to the population that is most successful. They introduce three operators but only present results on the scheduling of two of them. Experiments were done on unimodal and multimodal numerical benchmark functions. Their results indicate increased performance and robustness.

1994 William Spears [65] designed an adaptive operator scheduling approach that used an extra bit in the genome of an individuals to determine whether two-point or uniform crossover should be used. If the two parents disagreed on the value of this bit, the choice would be made at random. An alternative version used the average setting of the whole population to make this decision. Experiments were done on the N-peak problems by De Jong and Spears [38]. In his study, a comparison is made to an algorithm choosing randomly between the two operators. Results indicate that performance improvements are primarily caused by the fact that more operators are available to the evolutionary algorithm.

1995 Julstrom [39] presents a scheme similar to that of Davis (details are found in section 3.2.2). The algorithm performs scheduling between one mutation operator and one crossover operator. Experiments were conducted on a numerical benchmark function and one small instance of the Traveling Salesman Problem (30 cities). No comparisons were done to non-adaptive alternatives.

1996 Notredame and Higgins [52] used Davis' operator scheduling method to schedule between 22 operators for the Multiple Sequence Alignment problem. Good results were reported, but no comparisons were made to non-adaptive alternatives.

1996 Tuson and Ross [76] also used a scheme reminiscent of Davis' and also in their study scheduling is limited to two operators – a mutation and a crossover operator. They applied it to a flowshop sequencing problem and some well-known optimisation benchmarks. Comparisons were made to a manually tuned EA. Results indicated that the adaptive operator scheduling algorithm was less sensitive to bad initial probability settings, but did not improve overall results. It was suggested that operator performance might not be the only factor to consider when adjusting operator probability settings. However, it was also mentioned that their algorithm does provide performance improvements on timetable problems.

1998 Eiben, Sprinkheuizen-Kuyper and Thijsen [15] used competing subpopulations to schedule between 10 variants of the same crossover operator, each using a different number of crossover points. Experiments were done on seven standard optimisation benchmark functions. They conclude that their adaptive EA performed as well as an EA using the best single crossover operator. Operator scheduling was concluded to be a safe tool to find optimal operators, but did not improve overall results.

2002 Thomsen and Krink [74] present another approach based on competing subpopulation. A more sophisticated scheme for migration between populations was however used. Scheduling was done on three different mutation operators, and experiments were conducted on six well-known benchmark problems. Good results were reported. However, the approach was only compared to an alternative evolutionary algorithm with a single operator.

To summerise, different studies of operator scheduling have been conducted, but it is hard to draw any general conclusions from them. Most studies only use very few

operators or only test their approach on a few problems. Most importantly, I have found no studies that compare different operating scheduling schemes on the same problems. This thesis is an attempt at such a comparison for three different problem domains: the Traveling Salesman Problem, Multiple Sequence Alignment and numerical optimisation benchmarks.

3.2.2. Implemented Algorithms

The following set of adaptive operator scheduling algorithms was composed partly of existing methods and partly of methods that were designed specifically for this study. For this selection of methods implementation-simplicity was a priority. If adaptive operator scheduling is to be a practical method replacing elaborate manual comparisons, it is vital that the implementation efforts are kept within reasonable limits. The following list contains a description of each method. Each description is concluded with an overview of the parameter settings that were used for this algorithm. These settings were manually tuned based on preliminary experiments.

The basic structure of the evolutionary algorithms used with these different schemes is shown in Algorithm 1 and 5. These basic loops both consider the case where a the same operator pool is used for both mutation and crossover operators. An alternative is to use separate pools that are adapted independently. Both of these designs will be used in the experiments.

Notice that there is no explicit selection phase in the algorithms. Instead, for each position in the population, a candidate solution is computed which only replaces the old individual in that position if is has higher fitness value. This scheme is successfully used in Differential Evolution [67] and gave good results in initial experiments (results not shown).

The Operator Scheduling Algorithm by Davis

Davis' algorithm from 1989 [12] represents one of the first efforts at operator adaptation in EAs. It uses population-level adaptation and absolute update rules. More specifically, a global set of operator probabilities is adapted based on the performance of the operators in the last generations (size of adaptation window W). The performance of operators is measured by the quality of the individuals that they produce. Newly created individuals are rewarded if their fitness surpasses the fitness of all other individuals in the population. The size of the reward is determined by the amount of improvement. Furthermore, a certain percentage of the reward is recursively passed on to the individual's ancestors (to a certain maximum depth M). This reward strategy is motivated by the fact that a series of suboptimal solutions is often necessary in order to ultimately reach a better solution, and that the corresponding operators should thus be rewarded (algorithm 2).

With certain intervals (I), the rewards are used to update the probability setting. For each operator i:

$$p'_i = (1 - S) * p_i + S * \frac{\text{reward}_i}{\text{totalReward}} \tag{3.1}$$

Algorithm 1 Basic loop for the Davis and ADOPP algorithms.

```
1  INITIALISE(population)
2  while ¬ done
3        do
4           newPop ← population
5           for i ← eliteSize to population.SIZE()
6                 do
7                    CHOOSE-OPERATOR()
8                    newIndividual ← CROSSOVER-OR-MUTATION(population)
9                    if newIndividual .FITNESS() > population[i] .FITNESS()
10                     then
11                          newPop[i] ← newIndividual
12          population ← newPop
13          if generations mod I = 0
14            then
15                 ADAPT()
```

Algorithm 2 Adaptation procedure for the Davis algorithm.

```
ADAPT(operatorPool)
1  for W individuals created most recently
2        do PASS-REWARD-TO-PARENTS()
3  for W individuals created most recently
4        do ASSIGN-REWARDS-TO-OPERATORS(operatorPool)
5  UPDATE-OPERATOR-PROBABILITIES(operatorPool)
```

where p'_i is the new probability for operator i and S is the shift percentage parameter, determining the influence that the current update has on the total probability setting.

In the present study a slightly modified version of Davis' algorithm is used. Lower bounds are set on the probabilities to avoid extinction of operators, and in adaptation phases where no improvement is made the probabilities are shifted slightly toward their initial positions. Furthermore, the steady state evolutionary approach is replaced by an EA using elitism. Preliminary experiments showed that these modifications gave better results and faster convergence (results not shown).

Parameter settings: Window of adaptation (W): 100 individuals, Interval of adaptation (I): 20–50 evaluations, Shift percentage (S): 15%, Percentage of reward to pass back (P): 90%, Number of generations to pass back (M): 10.

The ADOPP Algorithm

Julstrom's Adaptive Operator Probabilities algorithm (ADOPP) [39] from 1995 is very similar to Davis' approach. The main difference between the two is the way in which ancestral information is represented. While Davis provides each individual with pointers to their parents, Julstrom explicitly provides each individual with a tree specifying which operators were used to create its ancestors. Davis' approach is more effective since the tree structure is implicitly present in the individuals and does not have to be copied every time new individuals are created. However, Davis' algorithm has some disadvantages that Julstrom avoids. Since the ancestral information in Davis' algorithm is stored in the individuals, information is lost when individuals die. The depth of the implicit trees are therefore somewhat unreliable and, in general, Davis' algorithm is only able to maintain a moderate amount of ancestral information. Furthermore, the individuals in Davis' algorithm require a great deal of bookkeeping to sustain pointers only to living individuals, an inconvenience which is avoided in ADOPP. Another difference between the two approaches is that Davis rewards individuals that improve the overall best individual in the population, whereas Julstrom only requires individuals to exceed the median individual.

The reward scheme is an integrated part of the ADOPP. With every creation of a new individual that exceeds the population's current median all operators in the individual's tree are instantly rewarded. The only role of the adaptation phase is to convert the rewards to new probability settings (algorithm 3):.

Algorithm 3 Adaptation procedure for the ADOPP algorithm.

```
ADAPT(operatorPool)
1  UPDATE-OPERATOR-PROBABILITIES(operatorPool)
```

When converting operator rewards to a new probability setting, ADOPP uses a greedy variant of the update rule by Davis (corresponding to $S = 100\%$). For each operator i:

$$p_i' = \frac{\text{reward}_i}{\text{totalReward}} \tag{3.2}$$

where p_i' is the new probability for operator i.

Parameter settings: Window of adaptation(W): 100 individuals, Interval of adaptation(I): 1 generation, Percentage of reward to pass back(P): 80%, Height of trees(M): 4.

Adaptation Using Subpopulation (Subpop)

This approach was inspired by the work of Schlierkamp-Voosen and Mühlenbein [61] on competing subpopulations, in which different subpopulations represent different operator strategies. This idea can be used as an adaptive operator scheduling method by letting each subpopulation represent the use of a single operator. The EA thus maintains a number of subpopulations equal to the number of operators. For each of these subpopulations, only one designated operator can be applied to the individuals currently residing

there. During the course of a run, the relative sizes of the subpopulations are altered depending on their fitness. The fitness of a subpopulation is calculated based on where the currently best individuals reside. In order to avoid random oscillations in population sizes this decision is based on the last 10 generations (as proposed by Schlierkamp-Voosen and Mühlenbein [61]). For group i, the quality measure is defined as

$$quality(i) = \sum_{k=0}^{9}((1 - \frac{k}{10}) * best\_ind_k(i)) \tag{3.3}$$

where k denotes the number of generation back in time, and $best\_ind_k(i) = 1$ designates that the best individual was in population i, k generations ago.

In each adaptation phase the best subpopulation is rewarded with an increase in size, and receives a donation of individuals from all other subpopulations. To avoid the extinction of operators, only subpopulations with a size above some fixed lower bound are forced to make this donation. Large subpopulations create more offspring and thus have a larger probability of improving the global best fitness. This results in a strong bias towards larger subpopulations, making it difficult for smaller populations to compete. As an attempt to counter this effect, a random migration scheme is used: At certain intervals some of the best individuals from the best population migrate to a randomly selected population (algorithm 4).

Even though this algorithm might appear to be adapting at a different level than the Davis and Julstrom algorithms, this is in fact not the case. Just as the previous algorithms, Subpop is a population-level adaptation algorithm using absolute update rules. The parameters adapted are in effect the same as for Davis and ADOPP (global operator probabilities). The basic loop of the EA for this algorithm is reminiscent of the one for Davis and ADOPP, with the natural generalisation to n populations instead of one.

Algorithm 4 Adaptation procedure for the Subpop algorithm.

```
ADAPT()
 1  if (generations mod E) = 0
 2     then
 3          bestPop ← FIND-BEST-POPULATION()
 4          for i ← 0 to numberOfPopulations
 5               do if population[i] ≠ bestPop
 6                    then
 7                         DONATE(population[i], bestPop)
 8  if (generations mod M) = 0
 9     then
10          bestPop ← FIND-BEST-POPULATION()
11          MIGRATE(bestPop)
```

Parameter settings: Evaluation interval (E): 4 generations, Shift amount (SA): 10%, Migration interval (M): 4 generations, Migration amount (MA): 5 individuals, Interval of adaptation (I): 1 generation.

Self Adaptive Operator Scheduling (SIDEA, SPDEA)

The algorithms in this section are novel methods of operator adaptation partly inspired by the work of Spears in 1995 [65]. In Spears' paper, scheduling is done between two crossover operators by adding a single bit to the genotype that indicates the preferred operator. This bit is used either locally (at the individual-level) or globally (at the population-level) to determine which operator is to be applied. When used locally, the choice of operator is made based on the bit-settings of the parent(s) involved. When used globally, the average bit-setting of the whole population is used as a measure of quality to base this decision on. As mentioned in section 3.1.1, self-adaptation relies on the assumption that good individuals often have good parameter settings. Instead of an explicit adaptation phase in the algorithm, good parameter settings are automatically favoured by the EA. The only requirement is that the parameters undergo variation operations like the rest of the genome.

I generalised Spears' method by expanding the single bit to an array of n probabilities, each denoting the probability that a certain operator is applied. Unlike Spear's approach, the representation of the scheduling information is not compatible with that of the problem representation and therefore cannot be modified automatically as part of the genotype. It was therefore necessary to design a specialised variation operator for this task.

The problem of finding the optimal probability setting for n operators is a numerical optimisation problem so that in principle, any variation operator from this domain would suffice. However, given the fact that the probabilities must all be positive and sum to one, only a small subset of the n-dimensional search space constitutes legal solutions. If an arbritrary EA variation operator is used it would be necessary to apply some repair scheme after each operation to ensure feasibility.

To avoid this I used an adapted version of the mutation operator used in Differential Evolution (DE) [67], which produces solutions that are only rarely illegal. The DE variation operator uses three individuals to create an offspring by applying a mutation step followed by a crossover step. During the mutation step, a new genotype $\vec{x}'$ is created as

$$\vec{x}' = \vec{x}_1 + F * (\vec{x}_3 - \vec{x}_2) \tag{3.4}$$

The crossover step subsequently creates a new individual by combining components from $\vec{x}'$ with components from the corresponding individual in the previous population (see section 2.2.4). For our purpose, however, the mutation step alone has exactly the properties we need: Selecting three random points on a hyperplane and adding the vectorial difference between two of them to the third results in a new position on the plane. In other words, given three legal probability settings it will produce a new one with probabilities that sum to 100%. An example of this behaviour for 2 dimensions is

given in figure 3.1. Equation (3.4) of course gives no guarantees that the entries of the created vector have values between zero and one. This is however easily fixed by forcing values that lie outside the domain back to the nearest boundary.

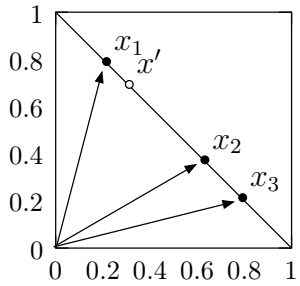


Figure 3.1.: The DE mutation operator for two dimensions. Three random individuals x_1, x_2, x_3 are chosen and a temporary individual x' is created by taking the vectorial difference between x_3 and x_2 and adding it to x_1.

The mutation operator is used whenever new offspring is to be created. Each time, three random parents are selected, and the new probability setting for the child is calculated using (3.4) (algorithm 5).

As with Spears' algorithm, two different levels of adaptation exist. If individual-level adaptation is used, the probability setting of the child is used to choose an operator. If adaptation works at the population-level, the average of all probability-settings in the population is used to select an operator. The two versions of the algorithm will be referred to as the SIDEA and the SPDEA respectively.

Parameter setting: F-value for the DE mutation: 0.5.

Operator Scheduling Inspired by Evolution Strategies (SIESA)

Self-adaptation has been applied in the Evolution Strategies (ES) community since 1977 [4]. Here it was used to adapt the behaviour of the mutation operation during the course of a run. The mutation operator consists of an addition of normally distributed values $z_i \sim \mathbf{N}(0, \sigma_i'^2)$ to the components in the genotype. The variances, $\sigma_i'^2$, are adapted so that the effect of mutation is different for different individuals and for different components of the genotype. Again, under the assumption that individuals of high fitness often have good parameter settings, the algorithm can find the optimal variance setting using only the selection pressure already present in the algorithm.

Taking self-adaptation one step further, we now use the ES mutation step as an alternative to the DE mutation operator described in the previous section. This means that the evolutionary algorithm will simultaneously optimise the problem at hand, the operator probability setting, and the setting determining the optimal variance for the mutation of this probability setting.

Since the search space defined by the operator probability settings is highly interdependent, it is meaningless to adapt each variance σ_i^2 at the component-level. Instead one value σ is associated with each individual. The necessary mutation equation from the ES literature is [3]

$$\sigma' = \sigma * \exp(s_0) \tag{3.5}$$

where $s_0 \sim \mathbf{N}(0, \tau_0^2)$ and $\tau_0^2 = \frac{1}{n}$. Since this mutation operator does not have the convenient properties of the DE operator, the probability-setting has to be normalised after each mutation. It is difficult to tell exactly how big the impact of this normalisation is compared to the effect of the mutation itself.

Algorithm 5 Basic loop for the SIDEA, SPDEA and SIESA algorithms.

```
1  INITIALISE(population)
2  while ¬ done
3        do
4           newPop ← population
5           for i ← eliteSize to population.SIZE()
6                 do
7                    newProbSetting = CALCULATE-PROBABILITY-SETTING()
8                    CHOOSE-OPERATOR(newProbSetting)
9                    newIndividual ← CROSSOVER-OR-MUTATION(population)
10                   newIndividual.probSetting ← newProbSetting
11                   if newIndividual .FITNESS() > population[i] .FITNESS()
12                      then
13                            newPop[i] ← newIndividual
14          population ← newPop
```

Parameter setting: τ_0^2: $\frac{1}{n}$ (ES default value [3]).

4. The Traveling Salesman Problem

The Traveling Salesman Problem (TSP) stands as of the most thoroughly investigated combinatorial problems in Computer Science. The problem's simple definition makes it an ideal representative of the class of NP-hard problems. It can be formulated as follows:

> Given a list of cities and the pairwise cost of travel between them, find the path with minimal cost that visits them all and returns to the city of origin.

In order to formalise this definition, several basic computer scientific concepts must be defined. The first is that of a *graph*. Contrary to the normal definition, in computer science this term is used to describe a set of *nodes* (vertices) and a set of *edges*. A *complete* graph has an edge between all pairs of vertices. These edges can be directed and can have weights associated with them. The degree of a vertex is the amount of edges that it is connected to. A *path* in a graph is a series of nodes connected to each other with edges, so that you can move from one end of the path to the other. A *cycle* is a path that ends where is starts, and a *Hamiltonian cycle* is a cycle covering all nodes in the graph exactly once (figure 4.1). The Traveling Salesman Problem can now be for formally defined as follows

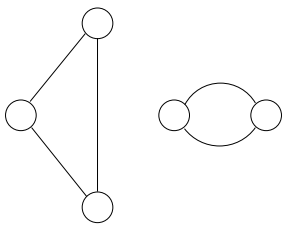


Figure 4.1.: An example of a simple undirected graph containing two cycles.

> Determine the shortest Hamiltonian Cycle in a complete weighted graph.

The exact origin of the problem is not completely clear. The Hamiltonian cycle problem (a special case of the TSP) was mentioned as early as 1759 by Euler, who was interested in solving the *knights tour* problem, which is the problem of finding a path on a chess board that visits all squares, and returns to the first, using only the characteristic move associated with the knight-piece. The term "Traveling Salesman Problem" was apparently used for the first time in the 1930's, and was in the late 40's popularised by the RAND corporation, which attracted many of the top minds in operations research at that time. A more in-depth description of the history of TSP can be found in the book "The Traveling Salesman Problem" by Lawler, Lenstra, Rinnooy Kan and Shmoys [44].

The TSP is popular for reasons other than just its simple definition and appealing name. There are a variety of real world problems that are either direct instances of the TSP or that can be easily transformed to it. David Johnson [36] mentions instances in Circuit board drilling with 17,000 nodes and X-ray crystallography instances with 14,000

nodes. An even more extreme case concerns the optimisation of VLSI (Very Large-Scale Integration) chip design, where problem sizes can exceed a million nodes.

Gerhard Reinelt [59] has created TSPLIB, a extensive collection of TSP instances for benchmark purposes. A large range of problem sizes is available from this source. Furthermore, the web page `http://www.math.princeton.edu/tsp/` shows a list of VLSI instances and a collection of data forming a so-called *TSP World Problem*, where the goal is to find the shortest path that visits all 1,904,711 recorded cities on the earth[1]. No optimal solution to this problem has at this point been found, although presently the best result is only a mere 0.088% from the current best lower bound.

4.1. Previous Work on the TSP

Throughout the last decades a variety of different strategies have been employed to solve the problem. These can be divided in two fundamentally different approaches: *tour construction* heuristics and *local optimisation* heuristics. The first approach builds a tour one city at a time. A simple example of this is the *Nearest Neighbor* algorithm which consistently chooses the city that is closest to the one chosen previously. Intuitively, this seems a reasonable approach, and it is probably close to the intuitive algorithm used by many postmen on their daily tours. An alternative is the *Greedy* algorithm, in which a Hamiltonian cycle is built one edge at a time by considering all available edges at each step and choosing the one of minimal length. An edge is available only if it can be included in the Hamiltonian path without introducing sub-cycles or vertexes with a degree greater than two. Typically, these algorithms create tours which are 15–20% longer than the optimal solution. More advanced tour construction algorithms exist, but have so far not succeeded in producing solutions better than 10% in excess of the optimum. Given their unsatisfactory performance, algorithms of the first category are generally not used as TSP problem solvers in practice. They are, however, used to create initial tours (seeds) for our next class of TSP algorithms: the local search algorithms.

The local search algorithms use local manipulations to iteratively improve a given solution. The 2-OPT and 3-OPT algorithms are well known algorithms of this type. The 2-OPT algorithm removes 2 edges from the tour and replaces them with two other edges if this improves the overall length of the tour. 3-OPT is the natural generalisation of the 2-OPT, replacing up to three edges (figure 4.2 and figure 4.3). Johnson and McGeoch [37] report results of about 6.9% and 4.6% above the optimal for these two algorithms if they are seeded with tours generated by a randomised version of the *greedy* algorithm described above.

Probably the most famous of all TSP algorithms is the Lin-Kernighan algorithm from 1973 [63]. This also belongs to the class of local optimisation algorithms but is significantly more complex than the 2-OPT and 3-OPT algorithms. A complete description is beyond the scope of this thesis, but I will shortly introduce some of the main ideas:

Lin and Kernighan acknowledged the success of the k-OPT algorithms, which had

[1] The front page of this thesis shows an illustration of the positions of these cities. The picture was downloaded from `http://www.math.princeton.edu/tsp/`.

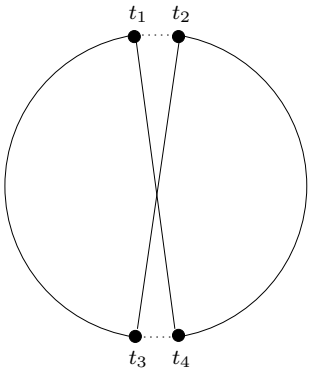


Figure 4.2.: A single operation of the 2-OPT operator. The dotted lines indicate the tour before the operation.

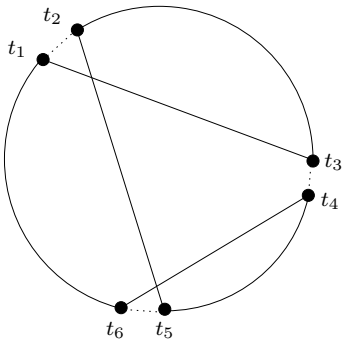


Figure 4.3.: A single operation of 3-OPT operator. Note that for any three edges there are several ways of recombining them.

previously provided promising results for k=2 and k=3 [10, 45]. For increasing values of k solutions seemed to improve, but at a great computational expense. Furthermore, it was difficult to know in advance when an increased value of k would give improved results. The basic idea of the Lin-Kernighan algorithm was to apply different settings of k during the course of the solution process.

Let S denote the complete graph consisting of all cities and all edges between them. The basic observation is that for any given suboptimal tour T, a certain number of edges are "out-of-place" and should be replaced by more optimal edges taken from $S - T$. The goal is thus to find a list of pairs p_i, consisting of an edge x_i from T and an edge y_i from $S - T$, for which an exchange would be beneficial for the overall tour. The 2-OPT algorithm considers two of such edge-pairs at a time, whereas the 3-OPT algorithm considers lists of length three. Lin-Kernighan considers edge-pairs as long as it assesses that improvements can be made. Each Lin-Kernighan step works as follows:

Gradually a list of to-be-exchanged pairs of edges is constructed. Each addition to this list corresponds to an increase of k by one. Initially two edges are chosen, one from T and one from $S - T$. The pair that is chosen is the pair that will benefit most from an exchange – the most out-of-place pair. This pair is added to the list. Hereafter, another pair is selected, now from $T - x_1$ and $S - T - y_1$, respectively. Executing the exchange of all pairs in the list at this point would correspond to a 2-OPT move. However, the Lin-Kernighan algorithm continues to select edge-pairs until some stopping criteria is met. When this occurs the value of k that produced the greatest overall gain is used. Note that this is not necessarily the k value used in the last iteration: For each step in the iteration we find the edge-pair whose replacement gives the highest gain, but this value can be negative. Lin and Kernighan argue that this helps the algorithm to avoid local optima. As with 2-OPT and 3-OPT, these steps are repeated as long as improvements can be made.

As usual, the devil is in the details. There are several issues that require a more thorough description before the above algorithm can be implemented. Especially the

choice of edges x_i and y_i is not as trivial as described above. Several special cases have to be considered in order to ensure that all states of the list of to-be-exchanged pairs in fact give rise to legal tours. This is important because the algorithm must be able to stop an iteration at any point (for any value of k). Furthermore, Lin and Kernighan put further restrictions on the choice of edges by requiring that $x_1, x_2, \ldots, x_k$ and $y_1, y_2, \ldots, y_k$ are disjoint sets. This disallows previously inserted edges to be removed and previously deleted edges to be inserted (in a single iteration) and is reminiscent of the tabu-rules used in tabu-search (section 2.2.2). The reader is referred to Lin and Kernighan's original paper [63] and Johnson and McGeoch's case study paper [37] for further details of the algorithm's implementation. The latter also describes some useful tricks on how to improve the efficiency of the 2-OPT, 3-OPT and Lin-Kernighan algorithms. In order to achieve optimal running times, significant implementation efforts are required in all cases.

Recent years have produced even stronger implementations of the Lin-Kernighan algorithm. Keld Helsgaun's LKH algorithm [28] seems to be one of the most promising approaches currently available. It finds optimal solutions for all solved instances in TSPLIB, and is the algorithm responsible for the current best World Tour mentioned above (found in September 2003).

Although the Lin-Kernighan algorithm is highly succesfull, progress in other fields should also be mentioned. Both in Evolutionary Computation and in Simulated Annealing communities the TSP problem been widely studied [43]. A great deal of work has been devoted to the design of new operators that exploit more or less complex features of the problem-instance to create meaningful offspring in the evolutionary process. Through time, these efforts have resulted in a long list of different mutation and crossover operators. However, evolutionary approaches have generally been prone to slow convergence speeds. Furthermore, up until recently the evolutionary operators could often not produce satisfactory results on their own, and had to be combined with for instance a 2-OPT or 3-OPT algorithm.

In recent years several new operators have been introduced and shown to produce optimal or near-optimal results for most problems [51, 69]. Also, experimentation with combinations of evolutionary algorithms and classic approaches continues to produce promising results [75]. The previous record for the best World Tour was actually produced by a hybrid of iterated Lin Kernighan and a genetic algorithm[2].

The new versions of Lin-Kernighan however have such good performance, that it is questionable that any general search heuristic such as Evolutionary Algorithms or Simulated Annealing can really compete. The only drawback of the Lin-Kernighan is that an efficient implementation is not easily done. However, the same can be said for some of the advanced operators required to make evolutionary algorithms give satisfactory performance.

The Traveling Salesman Problem however remains interesting as a benchmark problem for Evolutionary Algorithms. It is a well understood problem, with easily available prob-

[2]This algorithm was created by H. D. Nguyen, I. Yoshihara, K. Yamamori, and M. Yasunaga. No paper was published. For a short description see `http://www.research.att.com/~dsj/chtsp/nguyen.txt`.

lem instances and can provide an assessment of the quality of an evolutionary algorithm. It is in this quality that I consider the TSP in the present study.

In the remains of this chapter, I present an approach that aims to automate the process of operators selection for the TSP problem. The chapter is a modified version of two publications on the subject, which can be found in appendix F.

4.2. Investigating the Effect of Operator Scheduling

The application of adaptive schemes is often accompanied with a certain concern regarding the convergence speed. Adaptation seems likely to slow down convergence to some degree, and it is difficult to assess whether the added benefits of adaptive parameter tuning outweighs a possible loss in performance time.

Turning to adaptive operator scheduling, even the motivation can sometimes be hard to grasp. If we assume for a moment that the main goal is to find one or two optimal variation operators for a given problem, operator scheduling might not seem to make much sense at all. If a manual comparison between operators is done as a preliminary step, all subsequent runs can be done with the optimal operator setting without any delay.

Previous studies [65] show that the above assumption might constitute a needless restriction on the obtainable performance of an evolutionary algorithm. Indeed, it seems that in certain cases, algorithms benefit from having more than two operators to choose from. Furthermore it seems unnatural to force a fixed setting on the operators throughout the whole run, when different phases of the solution process might require the use of particular operators.

The goal of my first study was to determine the effect that adaptive operator scheduling has on the TSP problem. A manual comparison of operators was done to determine the best combination of crossover and mutation operators and performance was subsequently compared to an adaptive operator scheduling strategy. The main question was whether the elaborate manual tuning of operators could be avoided without a decrease in performance.

4.2.1. Experiments

Only a single adaptive operator scheduling scheme was used in this study. Based on its popularity, the adaptive operator scheduling algorithm by Davis was chosen. A few modifications were made. The version used is described in section 3.2.2.

Classically, a generation in a evolutionary algorithm consist of both a mutation and a crossover phase. I chose to maintain this design, and thus to separate the choice of operators for these two phases. The manual comparison is done for each pair of mutation and crossover operators and the adaptive operator scheduling scheme has an operator pool for each of them. It is within these groups that Davis' adaptation scheme is applied (the operator probabilities in each group sum to 1.0). The adaptive operator scheduling algorithm used will be referred to as *AOSGA*. The various non-adaptive algorithms using a fixed combination of two operators will be referred to as *fixed-op* algorithms.

As mentioned, a key question was whether the operator scheduling scheme was a viable alternative to the elaborate manual tuning of operators. There are two criteria to measure the success of an algorithm:

1. Quality: The results of the operator scheduling algorithm should be as good as the best of the fixed-op versions.

2. Speed: Convergence speed should not be significantly reduced.

The quality issue is investigated by comparing the results of the adaptive algorithm with all fixed-op combinations of a single mutation and crossover operator. Speed is tested by plotting the current best solution as a function of used evaluations during the course of the solution process (averaged over 30 or 100 runs).

Experiments were conducted on three symmetrical TSP instances: A 48-city problem, a 180-city problem and a 442-city problem (gr48, brg180 and pcb442), all of which are taken from the TSP benchmark problem collection TSPLIB [59].

Some initial experimentation was done to determine the best parameter settings for the algorithms. A crossover rate of 0.6 and an overall mutation rate of 1.0 were used. This is only reasonable if the elite size is kept relatively high – 80 out of a population size of 100 proved to be a good compromise between a classical generational EA and the steady state model used by Davis. Some of the TSP operators have additional parameters which were fine-tuned by hand based on single-operator runs of the algorithm. Details on these parameter settings can be found in the original paper (appendix F).

			Best fixed-op EA			AOSGA	
TSP	Optimum	Evals.	Operators	Best	Average	Best	Average
gr48	5,046	50,000	IVM+HEU	**5,046**	5,184.8(82.2)	**5,046**	5,166.0(75.9)
brg180	1,950	2 mil.	IVM+OX1	1,980	2,036.2(32.0)	1,970	2,067.0(46.0)
pcb442	50,778	3 mil.	SIM+OX2	54,847	57,278(910)	53,532	55,442.0(810)

Table 4.1.: The best of all non-adaptive algorithms (fixed-op EAs) compared to the AOSGA. The values are based on 1000 repetitions for the gr48 problem, and 100 for brg180 and pcb442. The mean values are given with their standard deviations in parenthesis. Boldface indicates that the optimum was found.

4.2.2. Operators

Table 4.2 lists the 6 mutation operators and the 10 crossover operators used in the experiments. They all operate on a path representation of the TSP. The selection of operators was inspired by Larrañaga's survey paper from 1999 [43]. A complete description of the details of these operators is beyond the scope of this paper. For more information the reader is referred to the original papers (see table 4.2) or Larrañaga's paper [43].

Name	Authors
Displacement Mutation (DM)	Michalewicz [49]
Exchange Mutation (EM)	Banzhaf [2]
Insertion Mutation (ISM)	Fogel [17]
Simple Inversion Mutation (SIM)	Holland [32]
Inversion Mutation (IVM)	Fogel [16]
Scramble Mutation (SM)	Syswerda [68]
Partially mapped Crossover (PMX)	Goldberg and Lingle [23]
Cycle Crossover (CX)	Oliver, Smith and Holland [55]
Order Crossover (OX1)	Davis [11]
Order Based Crossover (OX2)	Syswerda [68]
Position Based Crossover (POS)	Syswerda [68]
Heuristic Crossover (HEU)	Grefenstette [26]
Edge Recombination Crossover (ER)	Whitley, Timothy and Fuquay [78]
Maximal Preservative Crossover (MPX)	Mühlenbein, Gorges-Schleuter and Krämer [48]
Voting Recombination Crossover (VR)	Mühlenbein [47]
Alternating Position Crossover (AP)	Larrañaga, Kuijpers, Poza, and Murga [42]

Table 4.2.: The six mutation operators and 10 crossover operators that were used for the experiments.

4.2.3. Results

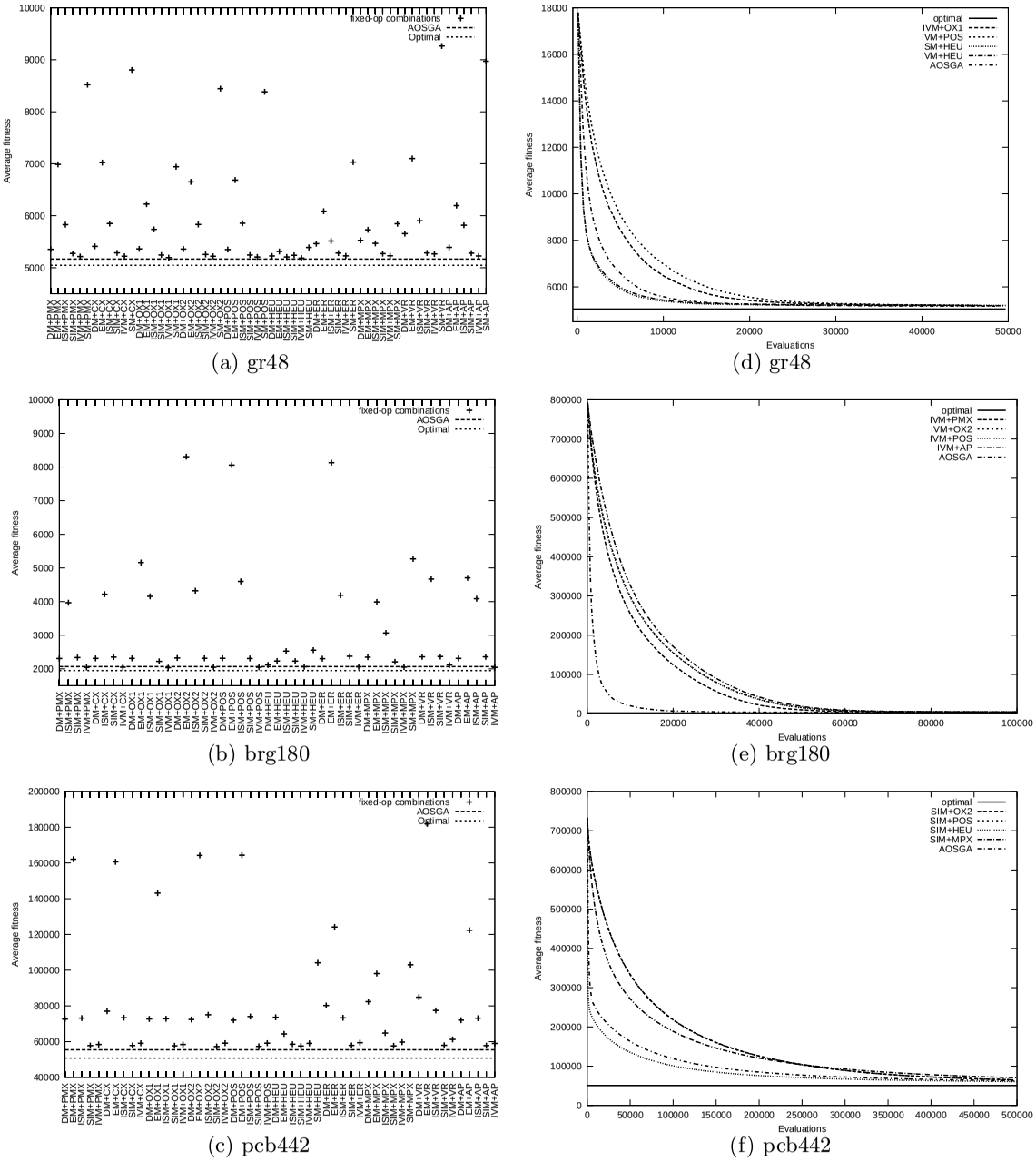
All pairwise combinations of mutation and crossover operators were applied to the three TSP problems. For each of these combinations, a simple tournament selection scheme was used. In table 4.1 the combination with the best average performance is compared with the results of the AOSGA using all operators. The results on gr48 are based on 1,000 repetitions, while only 100 repetitions were done for the two other problems. Execution time ranged from 3 seconds for the gr48 problem to 6–7 minutes for the pcb442 problem (Processor: Xeon 1.7 GHz). A graphical overview of the comparisons is given in figure 4.4.

To get an impression of the convergence speed of the AOSGA, progress during the course of a run was logged and compared to the progress of four of the best performing fixed-op algorithms (4.4 d–f). The graphs indicate that convergence speed of the AOSGA is not worse than that of the average fixed-op algorithm. For the larger problems, it appears that the AOSGA reaches acceptable solutions somewhat faster than the fixed-op algorithms.

4.2.4. Discussion

The results demonstrate that the two performance requirements are fulfilled. The adaptive algorithm can compete with any combination of single operators without a significant decrease in efficiency. The algorithm successfully manages to identify valuable operators during the course of a run and thereby eliminates the need to carry out such experiments by hand.

Concerning the quality of the solutions that the algorithm produces, table 4.1 and figure 4.4 indicate that the AOSGA obtains results comparable to the best fixed-op EA.



(a) gr48 (d) gr48

(b) brg180 (e) brg180

(c) pcb442 (f) pcb442

Figure 4.4.: Fixed-op combinations vs. the AOSGA. The a–c panels show the best fitness averaged over all runs (algorithms with extremely high average tour length are not shown), The d–e panels show the tour length progress for the average run.

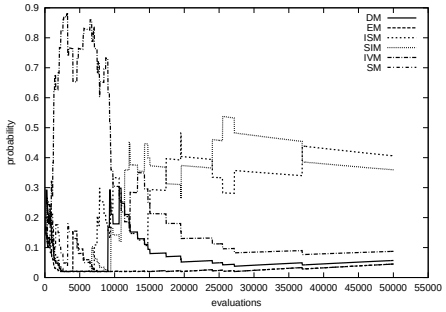


Figure 4.5.: Probability distribution for the different mutation operators (gr48).

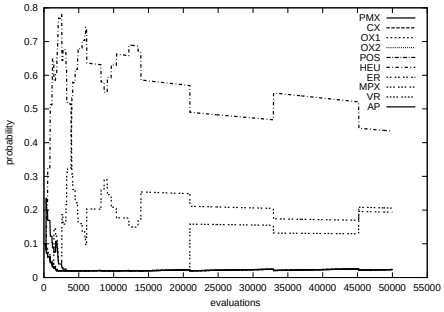


Figure 4.6.: Probability distribution for the different crossover operators (gr48).

For the two larger problems, the AOSGA converges to these results significantly faster than it's fixed-op counterparts.

As illustrated in figure 4.5 and 4.6, operator scheduling definitely takes place, and different operators gain dominance during different phases in a run.

4.3. A Comparison of Adaptation methods

The results of the first study indicate that operator scheduling makes sense on the TSP problem. Even though adaptive schemes in general might introduce a decrease in convergence speed, this effect was apparently either negligible or it was countered by the positive effects of operator scheduling. In any case, it was shown that operator scheduling is a potentially useful tool that can replace the elaborate manual comparison of operators.

However, for it to be a viable alternative, it is important that the process of implementing these scheduling algorithms is less of a burden than the manual comparison of the operators. The Davis algorithm used in the first study was somewhat cumbersome to implement. Significant bookkeeping is necessary to let individuals maintain a list of their parents and children, and to update these when individuals die. This second study was done to investigate whether alternative, more easily implemented algorithms might perform equally well.

Another issue that was raised during my first study was that the absolute results that were found for the TSP instances in the original paper were of rather low quality. This was attributed to the inability of the used operators to fine-tune solutions in the end phase of the solution process. I therefore introduced two new operators to the operator pool: The Edge Assembly Crossover (EAX) operator [51] and the Inver-over operator [69]. In initial investigations, especially the EAX operator showed great promise, giving near optimum results on all problems. In particular, it seemed to succeed in the final phases of the evolutionary process, where the previous operators tended to make the EA converge prematurely. Expectations were thus that all operator scheduling schemes should consistently favour this operator in the end phase of the evolutionary process.

4.3.1. Experiments and results

Initially, several experiments were conducted to tune the parameters of the algorithms. These showed that the algorithms benefitted from larger populations as the problem instances grew, confirming the findings by Nagata and Kobayashi [51]. As in the previous investigation, large elite sizes were used. The large populations are thus only partly replaced in each generation. The ADOPP variant uses a steady state approach, generating only one new individual in each generation (in agreement with Julstrom's original paper [39]). The other algorithms were run with elite sizes of 75% – 90% of the population. With these extensive elite sizes the large populations function mainly as libraries of genetic material that maintain a certain diversity in the population. During initial experimentation it was noted that with these large elite sizes, the selection phase had little effect. I therefore used a scheme adopted from Differential Evolution, where an implicit selection is done every time a new individual is created. Only if a newly created individual has a better fitness value than the individual it is about the replace will it become a part of the new population. The population and elite sizes used are listed in table 4.3.

	gr48	brg180	pcb442	nrw1379	pr2392	pcb3038
Population Size	500	600	2000	4000	4500	5000
Iterations	60,000	350,000	500,000	1,200,000	1,200,000	1,700,000

Table 4.3.: Population and elite size settings for the different problems.

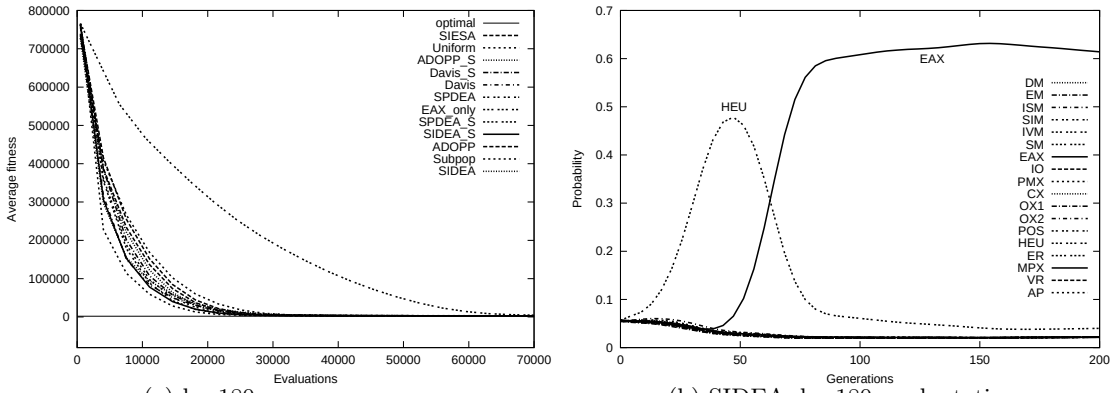
When implementing an adaptive operator scheduling algorithm, one has the choice of including all operators in one pool or dividing mutation and crossover operators in two separate pools. For Davis, ADOPP, SIDEA and SPDEA, both designs were implemented and included in the test set. The versions using separate pools are labelled with the suffix `_S`.

Two non-adaptive algorithms were included in order to evaluate the absolute value of Adaptive operator scheduling: The Uniform algorithm and the EAX_only algorithm. The Uniform algorithm includes all operators but uses equal probabilities for the application of them. The EAX_only uses only the EAX operator.

For all algorithms, 100 runs were done on six different TSP instances, all taken from the TSP benchmark problem collection TSPLIB [59]. The sizes of the problems ranged from 48 to 3038 cities. The average results are presented in table 4.4 (Standard deviations can be found in appendix C).

To give an impression of convergence speed figure 4.7(a) shows the average best fitness over time for the brg180 problem. The general pattern of this figure is recurrent across the range of tested problems. Figure 4.7(b) shows the operator probabilities over time for the SIDEA algorithm applied on the brg180 problem. This figure is also representative for all test cases: Across both problem instances and operator scheduling methods all algorithms consistently prioritise the EAX operator in the final phase. In the initial phases the greedy nature of the other operators (typically HEU) is most effective to gain

fast fitness improvements. This clearly improves performance compared to the algorithm applying the EAX operator exclusively.



(a) brg180 – convergence (b) SIDEA: brg180 – adaptation

Figure 4.7.: Convergence and Adaptation for brg180. The left figure clearly shows the EAX operator to convergence significantly slower than the adaptive algorithms. The right figure shows that the EAX operator is scheduled highly in the final phases of a run.

Based on the data in table 4.4, it is not possible to single out one algorithm as being the overall best. However, a certain categorisation of the algorithms can be made. Across the range of problems, the Davis(_S), SIDEA(_S), SPDEA(_S) and Subpop algorithms all performed about equally well. The data responsible for the mean values in the table were approximately normally distributed. The standard deviations are given in appendix C. Although there are non-overlapping confidence intervals (see appendix A between many of the results in table 4.4, and differences thus can be seen as being statistically significant, it varies from problem to problem which one algorithm performs best. The ADOPP and SIESA algorithms performed slightly worse than this first class. For ADOPP's case this might be ascribed to the algorithm being more greedy than, for instance, the Davis algorithm, and therefore might overcompensate bad operators when they by chance improve the overall solution. On the other hand, it seems that the SIESA adapts too slowly. This is not entirely surprising, since it is using self-adaptation at two levels. Another reason could be the somewhat disruptive renormalisation after each operation.

Perhaps the most striking results in table 4.4 concern the two non-adaptive algorithms. The EAX_only algorithm performs significantly worse than any of the other algorithms. This suggests that the EAX operator is first and foremost a fine-tuning operator and should be used together with some more exploration-oriented operators.

The Uniform algorithm, on the other hand, performs remarkably well. Especially the convergence graphs show surprisingly fast convergence. It should however be noted that the convergence graphs show the population's best fitness over time and that no information about the general state of the population can be derived from these graphs. The apparent conflict between the Uniform algorithm's good convergence speed and the somewhat worse end-results of this algorithm indicates that even though the algorithm is able to sustain a good elite of individuals, it does not have sufficient diversity in its

	gr48	brg180	pcb442	nrw1379	pr2392	pcb3038
Davis_S	5046.00	1950.30	50778.21	56689.46	378067.16	137758.94
Davis	5046.36	1950.00	50781.23	56685.04	378055.16	137752.69
SIDEA_S	5047.77	1950.20	50778.14	56699.82	378084.82	137779.77
SIDEA	5046.52	1950.30	50778.07	56715.85	378336.45	137793.97
SPDEA_S	5046.33	1950.40	50778.21	56698.31	378097.53	137781.66
SPDEA	5046.00	1950.30	50778.07	56691.05	378074.36	137755.16
SIESA	5047.21	1952.60	50781.83	56695.51	378098.00	137765.02
Subpop	5046.40	1950.60	50778.14	56682.53	378057.28	137743.03
Uniform	5048.34	1952.80	50778.91	56826.32	379570.91	137956.90
ADOPP_S	5046.66	1951.80	50778.56	56702.35	378100.71	137779.00
ADOPP	5046.66	1951.70	50778.35	56689.42	378079.92	137745.00
EAX_only	5046.00	1951.30	59558.96	128619.73	2823286.85	677724.28
Optimal	5046.00	1950.00	50778.00	56638.00	378032.00	137694.00

Table 4.4.: Comparison between the 11 adaptive operator scheduling algorithms and the non-adaptive EAX_only algorithm on six benchmark problems. The presented values are average results over 100 runs.

best individuals to find optimal solutions. The smaller amount of good individuals is naturally explained by the fact that the EAX operator is applied with only a fraction of the probability that it receives by the different adaptation schemes.

4.3.2. Discussion

Overall, the results show that performance is significantly improved when a multitude of operators are used. Even when one operator (e.g. EAX) is suspected to outperform all others, it might not be optimal throughout the whole run, and adaptive operator scheduling can exploit this fact to increase performance.

Based on the comparison of operator scheduling algorithms in this study, it is not possible to single out one of them as being the best. Many of the different methods performed equally well, and thus indicate that the selection of an operator scheduling method may be based on considerations as implementation efficiency or personal taste. The Davis algorithm requires significant bookkeeping and was found to be somewhat elaborate to implement, while especially the SIDEA, SPDEA, SIESA and Subpop variants were implemented fairly easily. Also the fact that the self-adaptive variants have only few parameters to tune might be a reason to favour these over the others. As an extreme case, even a uniform selection between operators seems to provide reasonable results at minimal implementation costs.

5. Multiple Sequence Alignment

Multiple Sequence Alignment (MSA) is one of the most fundamental problems in the field of bio-informatics (computational biology). Its significance lies in the fact that it is an essential ingredient for a multitude of other problems in the field. To illustrate, I will present a few examples taken from chapter 14 of Dan Gusfield's book "Algorithms on Strings, Trees, and Sequences" [27].

Protein families Protein families are collections of proteins that have similar biological properties. MSA can be used to create *profiles* that act as representatives for the sequences in such families. Profiles consist of a matrix that for each column gives the distribution of characters that appear in this column in an alignment of the whole family. New sequences can now efficiently be aligned to such profiles instead of the whole family. Another possibility regarding protein families is the identification of so called *motifs*. These are subsequences that occur in many of the family members and are thus characteristic for the family. This allows future proteins to be classified more easily. Furthermore, if certain motifs occur extremely frequently among the members of the family, it might indicate that these amino acids[1] are somehow connected to the biological function of the protein.

RNA secondary structure prediction A multiple sequence alignment can also contain information on possible correlated positions in the sequences. For the tRNA molecule this information has been used as a basis for secondary structure prediction [8], i.e. determining how this molecule folds on itself. The idea is that if a large amount of sequences agree on certain columns, there is an indication that the nucleotides in these columns might have mutated together. This would be the case if these nucleotides are paired in the secondary structure of the protein. Based on this information an edge matrix can be created that gives scores to all possible pairs of nucleotides, which can then be used to determine the the most likely secondary structure.

Phylogenetic trees The problem of constructing a phylogenetic tree which indicates the evolutionary history of a given set of sequences can also be solved using MSA. Certain types of MSA algorithms (progressive alignment) construct alignments by pairwise merging (aligning) smaller alignments, with the base case being an alignment of two sequences. To use this methods, some strategy has to be used to decide

[1] Amino acids are smaller molecules that are the primary components of a protein. Strictly speaking, proteins are composed of residues of these amino acids. The terms *residue* and *amino acid* will however be used interchangeably in this chapter.

which pairwise groups of sequences to align first. One strategy is to group and align sequences based on high similarity. Since similarity indicates a close biological relationship the MSA algorithm can now produce an evolutionary tree while solving the alignment problem.

5.1. The Problem

As the name implies the MSA problem consists of finding an alignment for a set of sequences. In general these sequences can be over any set of characters (the alphabet). In bio-informatics however, the relevant sequences often consist of nucleotides from the DNA alphabet {A, C, G, T}, or amino acids from the 20-character protein alphabet. In order to find a good alignment, it is necessary to have some objective measure to assess the quality of any provisional alignment. This is actually not as simple as it sounds. Different scoring schemes have been defined, but to this day, one of the greatest problems in solving MSA lies in the fact that alignments regarded as optimal by these scoring schemes do generally not correlate to a sufficient degree with hand manipulated alignments.

Defining a scoring scheme for a two sequence problem is an easier task. The most basic scheme assigns $+1$ to every matched column, -1 to every column consisting of two non-identical characters and -2 if one of the characters in the column is a space (blank symbol). However, empirical biological evidence indicates that spaces in the alignment should be treated differently. Instead of examining them one at a time, spaces should be considered in groups and scored as one object. Such a sequence of spaces is called a gap. Several scoring strategies for gaps exist, the most popular being the affine gap cost. Here every gap is penalised first by an initial Gap Opening Penalty (GOP) and thereafter with an Gap Extension Penalty (GEP) for each space it contains. Another extension to the basic scoring scheme is to differentiate non-matching columns. In the described basic scheme every non match was scored with -1. Substitutions in the alignment are likely to represent a mutation at some point in the evolutionary history. Such mutations between different amino acids are not all equally probable, and it thus seems natural to distinguish between the different substitutions and to assign scores based on the probability of their occurrence. Several scoring matrices are available where such scores can be looked up. The most used are the PAM [13] and BLOSUM [29] matrices.

The two sequence scoring scheme can be used as a basis for the n-sequence scoring function. This is called the Sum-Of-Pairs (SP) objective function[2] and consists of the scores that every pair of sequences in the multiple sequence alignment induce according to the two-sequence scoring scheme. A variation of this scheme assigns weights to the different pairwise alignments, thus marking some pairs as being more significant than others (WSP). This allows knowledge about evolutionary distances between the sequences to be incorporated into the algorithm. The general form of the WSP objective function is thus:

[2]Originally by Carrillo and Lipman [7].

$$WSP = \sum_{i=2}^{N} \sum_{j=1}^{i-1} W_{ij} \times \mathrm{COST}(S_i, S_j) \tag{5.1}$$

The SP score corresponds to $W_{ij} = 1$ for all combinations of i and j.

5.2. Previous Work

The MSA problem using the SP objective function has been proven to be NP-complete [77]. There is thus little hope[3] of finding an algorithm that can produce exact solution to problems of realistic size in feasible time. Again, this is an incentive for looking at heuristic search algorithms such as the evolutionary algorithm. Before describing evolutionary approaches to this problem I will however shortly present some alternative methods.

A dynamic programming solution is possible, although it can only be applied for a small number of strings. To give an impression of the basic ideas, I will present the algorithm for the two sequence alignment, which can be generalised to three or more sequences in a straightforward manner. Also, I will assume the most simple gap score presented above (-2 for each gap). This can also be generalised to the alternative scoring functions (with slightly more effort).

5.2.1. Dynamic Programming

Dynamic programming is used to significantly reduce the number of solutions that has to be considered. As described in section 2.1.5, the basic idea is to maintain a table of previously computed solutions to subproblems so that solutions for larger problems can be calculated quickly by combining these already computed values.

Consider two sequences x and y of length n and m, respectively. The idea is to construct a matrix M in which each entry $M(i, j)$ contains the maximum score for the alignment of the prefixes $x[0 \ldots i]$ and $y[0 \ldots j]$. The entry $M(0, 0)$ is thus zero, and the entry $M(n, m)$ contains the score of the whole alignment. Each entry $M(i, j)$ in the matrix can be calculated by looking at the three previous entries $M(i, j-1)$, $M(i-1, j-1)$ and $M(i-1, j)$. These three choices correspond to: (1) Matching a character from y with a gap. (2) Matching a character from each of the strings. (3) Matching a character from x with a gap. Since we are trying to maximise the overall score, the entry in $M(i, j)$ should be calculated as those of these three possible moves that gives the maximum score. We have previously defined the cost for each of these moves, and can now calculate $M(i, j)$ as:

$$M(i,j) = \max \begin{cases} M(i, j-1) - 2 \\ M(i-1, j-1) + \mathrm{match}(i, j) \\ M(i-1, j) - 2 \end{cases} \tag{5.2}$$

[3] Under the widely accepted assumption that $\mathbf{P} \neq \mathbf{NP}$.

where match(i, j) is the cost of matching $x(i)$ with $y(i)$. Of course, (5.2) only holds when $i > 0, j > 0$. The left and upper edges of the table are filled out with $M(i, j) = -2 * j$ and $M(i, j) = -2 * i$, respectively.

For this to work, the matrix has to be filled out so that the necessary previous values are available at the right time. This can for instance be done row-by-row or column-by column. Figure 5.1 displays the matrix resulting from the alignment of two arbitrary sequences `DAIMI` and `DAM` with arrows displaying which of the 3 moves where chosen for each entry. When the matrix has been filled out the actual alignment can be found by starting in $M(n, m)$, and backtracking up to the $M(0, 0)$ by using the moves that were chosen while filling out the matrix. The complexity of this algorithm is the time taken to fill out the matrix: $O(nm)$.

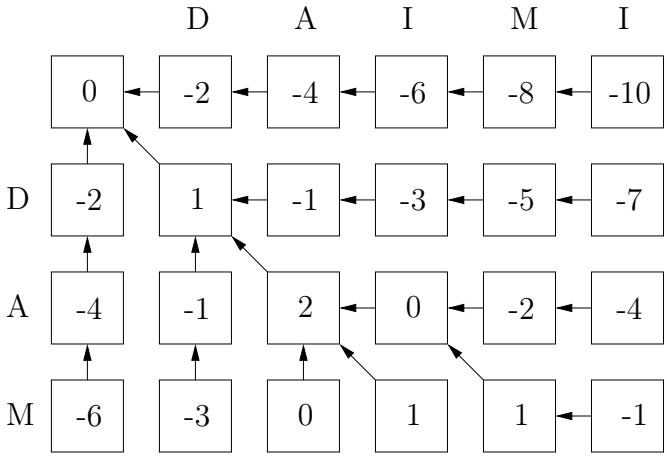


Figure 5.1.: Using dynamic programming to align the sequences `DAIMI` and `DAM`.

As mentioned, the dynamic programming approach can be generalised to three or more sequences. This is done by using a k-dimensional array (for k strings) and by filling it out as described for the 2-dimensional case. However, the amount of entries to be filled out grows as n^k, and this quickly creates both problems in space consumption and execution time. Another issue is that the calculation of each entry now no longer depends on three other entries, but in general on $2^k - 1$ previously calculated values.

Several methods have been proposed to speed up the dynamic programming approach. These are heuristic approaches that try to rule out certain parts of the n-dimensional array that can be guaranteed not to contain the optimal alignment path. The most elegant way of achieving this is to change the direction of the dynamic programming scheme. Instead of calculating a new matrix entry by looking at the values of the previous entries, each entry waits until it receives the values that it depends on. As soon as a new entry has been calculated its value is sent forward to those entries that are influenced by it. For the 2-sequence problem presented previously this means the entry to the right, the entry to the lower right and the entry beneath the current one. The algorithm maintains a queue of entries which have not yet been calculated. An entry enters this queue when the first of the entries that it depends on has been calculated. At this point, there is not

yet enough information to calculate its correct value. Only when all the values that it depends on have been calculated can the maximisation of equation (5.2) be completed. The invariant here is that when the current entry reaches the front of the queue, all values it depends on will in fact have been calculated. When an entry is put in the queue it is initialised with the appropriate term of the maximisation in (5.2). While it is in the queue, the other values that it depends on will at some point be calculated. Whenever this happens, the appropriate part of the maximisation of (5.2) is carried out. When it reaches the front of the queue it will have undergone calculation equivalent to that of (5.2).

Note that no optimisation has yet taken place. The forward version of dynamic programming is no faster than the ordinary backward one. It is however a clever trick to help the following optimisation idea: If we for some entries, at the moment that they are calculated, already know that the optimal path cannot pass through it, we omit passing its value on to the cells that it influences. In this way we can potentially prune away a large part of the calculations. The question is, however, how is it possible to decide that the optimal path cannot go trough the current entry. The idea is not very different from the branch-and-bound algorithm in presented in section 2.1.4.

Let us consider a case with k sequences $s_1, \ldots, s_k$ For a given entry $M(i_1, i_2, \ldots, i_n)$ in the k-dimensional array, let $d_{pq}(i_p, i_q)$ denote the optimal alignment score when aligning the two suffixes $s_p[i_p \ldots n]$ and $s_q[i_q \ldots n]$ from the set of sequences. Let $d'_{pq}(i_p, i_q)$ denote the pairwise alignment score that occurs in the sum-of-pair scoring scheme. Clearly the following statement holds for any entry in the matrix:

$$\sum_{p<q} d_{pq}(i_p, i_q) \leq \sum_{p<q} d'_{pq}(i_p, i_q) \tag{5.3}$$

For each entry, the alignment scores $d_{s_p s_q}(i_{s_p}, i_{s_q})$ for all pairs of sequences can be calculated by $k(k-1)/2$ pairwise alignments.

Let us now assume that we know some lower bound L for the score for our problem. Whenever we fill out an entry in our k-dimensional array, we can now check whether the following condition is true.

$$M(i_1, i_2, \ldots, i_n) + \sum_{p<q} d_{pq}(i_p, i_q) < L \tag{5.4}$$

If this is the case, then no alignment coming through the entry $M(i_1, i_2, \ldots, i_n)$ will be optimal, and we therefore do not have do send the value of this entry on to the entries that depend on it.

There are several methods for determining lower bounds for an alignment. Normally, some heuristic is used. It is clear that the success of the above method depends on the quality of the lower bound. For good lower bounds, a large speed-up can be achieved.

Methods as the one described are the basis for the MSA algorithm [46], which can produce optimal alignments (according to the SP-score) for up to 5 or 7 sequences of a few hundred residues each.

5.2.2. Star Alignment

Despite the efforts to improve the performance of the dynamic programming approach, it remains impractical for problems with many sequences. Several heuristic approaches have therefore been proposed. While these do not guarantee an optimal solution, they can often produce near-optimal alignments for many strings in reasonable time. One of these methods is the *star alignment algorithm* [62]. In this algorithm, one sequence is chosen to be the *centre*. This is typically the sequence that has the highest overall similarity to all the other sequences in the data set. The other sequences are aligned to this centre using pairwise alignment. These pairwise alignments are now merged into one large alignment one at a time. All pairwise sequences contain the same reference sequence (the centre sequence), and merging them is therefore just a matter of adding the necessary gaps to both versions of the centre sequence, and then adding the string to the alignment. Executing the k pairwise alignments takes time $O(kn^2)$ and each join operation takes time $O(kl)$, where l is the length of the final alignment. This gives a total complexity of $O(k^2l + kn^2)$.

5.2.3. Progressive alignment: ClustalW

Another approach is to pairwise align sequences two by two, and subsequently pairwise align alignments to form the multiple sequence alignment. This approach is called *progressive alignment*.

Often, a binary tree is constructed to indicate how the sequences can best be grouped. Similarity of sequences is normally used as a criterion. The biological interpretation of similar sequences is that their evolutionary distance is small, i.e. that they represent closely related taxa[4]. By using similarity as the basis for the generation of the tree, one is therefore in fact creating a phylogenetic tree. Similar sequences provide the most accurate evidence to base the alignment on. Aligning these sequences first gives a better foundation for the alignment of the more distant sequences.

The ClustalW algorithm [70] is a heuristic progressive alignment algorithm. It was presented by Thompson, Higgins and Gibson in 1994 and is still one of the most commonly used algorithms for multiple sequence alignment. It basically works as stated above with a binary tree constructed by the Neighbor-Joining method [60]. However, a large amount of empirical biological facts are incorporated to improve the accuracy of the alignments. For instance:

- A weighted SP-score is used. These weights are calculated based on the branch length of the binary tree. The weights are used to compensate for the fact that sequences are often not randomly distributed across all evolutionary distances. Alignments tend to consist of many similar sequences, which biases the construction of the alignment. Closely related sequences are thus down-weighted.
- Studies indicate that gaps appear more frequently after some amino acids than others. Gap penalties are therefore adjusted based on the current residue and

[4] Any group or individual that forms a biological unit (e.g. a species or a family).

position. Other studies show that gaps do not appear within distances of 8 residues of each other. Penalties are thus increased within this distance.

- The PAM and BLOSUM scoring matrices exist in different versions. Each of these are appropriate for alignments of sequences with certain distances. Instead of using one scoring matrix for the whole alignment process, this choice is adjusted based on relatedness of the currently considered sequences.

For more details the reader is referred to Thompson, Higgins an Gibson's original paper [70].

5.2.4. Iterative alignment methods

Another alternative to the methods described above are the so called *iterative* alignment algorithms. The idea is to gradually improve an alignment by various operations on the sequences (moving gaps, inserting gaps, etc.). The initial values for such an algorithm can either be completely random alignments, or alignments produced by one of the methods mentioned above. In the last decade several different iterative approaches have been proposed. Both Simulated Annealing approaches [31, 35, 40] and EA approaches [52, 6, 9, 34] belong to this category. One of the first evolutionary algorithms to be applied to the problem was the SAGA algorithm presented by Notredame and Higgins in 1996 [52]. In the context of the present study, SAGA is interesting because it applies adaptive operator scheduling to the MSA problem. We will take a closer look at this algorithm in section 5.4.

One important question concerning the iterative methods is how the quality of the initial alignments influences the quality of the final result. The concern is that providing too good initial values might cause the algorithm to get stuck in a local optimum. A paper worth mentioning in this regard is one by Thomsen, Fogel and Krink [72], where it is investigated whether their MSAEA program can be effectively used to improve alignments by the ClustalV algorithm (a predecessor to ClustalW).

5.2.5. BAliBASE

Many of the above algorithms use different scoring schemes and it is thus difficult to compare the different methods based on the quality of the results. The BAliBASE database [71] is a tool created to help with this process. The database contains 142 multiple sequence alignments, which are manually tuned based on the 3D structure knowledge of proteins. These sequences are grouped into reference sets with different degrees of similarity between the sequences.

The creators of BAliBASE propose two methods for comparing alignments with the reference alignments in the database. Sum-of-pairs score (SPS) and Column score (CS). SPS is defined as follows: Given a candidate alignment A consisting of n sequences and m columns, then for each column i and each pair of residues A_{ip} and A_{iq}, we define p_{ipq} such that $p_{ipq} = 1$ if these two residues are also aligned in the reference alignment, and $p_{ipq} = 0$ otherwise. For each column i, we now have the following score:

$$S_i = \sum_{k=1}^{N} \sum_{j=1, j \neq k}^{N} p_{ijk} \tag{5.5}$$

This score is summed over all columns and normalised:

$$SPS = \frac{\sum_{i=1}^{M} S_i}{\sum_{i=1}^{M_r} S_{ri}} \tag{5.6}$$

where M_r denotes the number of columns in the reference alignment and S_{ri} is the S_i score for the i'th column in the reference alignment.

The column score is now defined as follows: For each column i, the score $C_i = 1$ is given if all residues in the alignment are also aligned in the reference alignment. Otherwise, $C_i = 0$. Again, this score is summed for all columns and normalised:

$$CS = \sum_{i=1}^{M} \frac{C_i}{m} \, . \tag{5.7}$$

For many alignments, certain regions are more important than others. Especially for distantly related sequences only small regions are conserved among large regions of low similarity. Such regions have been identified in BAliBASE as so-called annotation files. The BAliBASE score program can utilise such files to give scores based only on such informative regions.

5.3. Investigating the Effect of Operator Scheduling

The encouraging results from the TSP study motivated an investigation on other problem domains. MSA was an obvious choice, since recent years have produced a great number of operators for this problem. More importantly, adaptive operator scheduling had actually been applied to this problem in the past (in the SAGA algorithm [52]) but the effects of the technique itself had not been examined exhaustively.

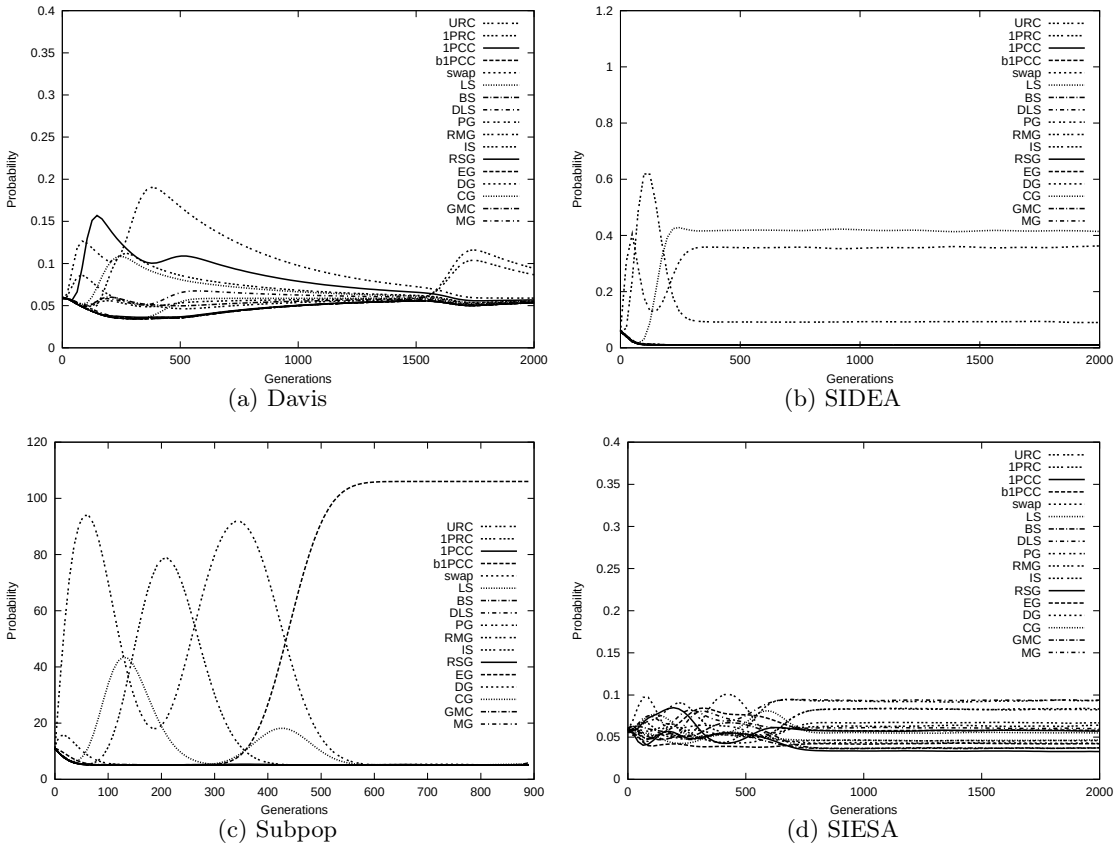
5.3.1. Experiments and Results

The collection of operators for the MSAEA algorithm served as a good basis for an investigation of adaptive operator scheduling on the MSA problem. The original description [72] contained only three mutation operators and a crossover operator. In a subsequent paper [73] more operators were introduced and recently this set was further expanded to a total of 13 mutation operators and 4 crossover operators. Appendix D contains a short description of the functionality of each of these operators.

The algorithms presented in section 3.2.2 were compared in a comparative analysis of five benchmark problems taken from BAliBASE: 1hfh, 1pfc, 1aboA, 1idy, 451c. For each algorithm, 100 repetitive runs were done, each with a population size of 200. The ADOPP variants were run as steady state algorithms (i.e. an elite size of 199), while an elite size of 160 was used for the others. An unweighted sum-of-pairs scoring scheme

was used with an affine gap-cost function. The GOP and GEP were set to 8 and 12, respectively. For practical reasons the score function was inverted so that the problem effectively became a minimisation problem.

Figure 5.3 shows the average fitness over time (generations) for the five benchmark problems. The actual results can be found in appendix D (table D.1 and table D.2). To get a visual impression of the operator scheduling process, figure 5.2 shows the distribution of operator probabilities over time.



(a) Davis (b) SIDEA

(c) Subpop (d) SIESA

Figure 5.2.: Adaptation graphs showing the distribution of operator probabilities as a function of the number of generations. For the Subpop algorithm, the graph shows the amount of individuals in each population.

The results in table D.1 do not seem particularly impressive[5]. Only small improvements are made compared to the seed provided by clustalw. Furthermore, the BAliBASE scores (given in parenthesis in the entries of table D.1) indicate that even these small increases in fitness do not always correspond to a better alignment. This problem is, however, due to the scoring functions used. For high score values, the correlation between the fitness

[5]The results are however not significantly worse than other attempts at improving ClustalW seeds. See for instance the latest MSAEA paper [73]. Note that in their paper, no annotation files are used. All BAliBASE scores are therefore somewhat lower.

of scoring schemes and the BAliBASE scores decreases. This effect was also observed by Thomsen, Fogel and Krink in their most recent paper on the MSAEA [73]. In the present study I consider MSA strictly as an optimisation benchmark, and will thus focus on the fitness values given by the scoring function.

Despite the small scale of improvements, the results give rise to some interesting observations. As in the study of TSP, it is difficult to isolate the best algorithm, since results vary across the problems investigated. If any, the Subpop algorithm should get credit for performing best in three out of five cases. The differences are however rather small.

Another observation concerns the decision on whether to use a single pool for all operators or distributing operators across two pools (separating mutation and crossover operators). It seems that the separate pool strategies (designated with an `_S` suffix) generally perform somewhat worse than the single pool strategies. Most interesting is, however, that the Uniform algorithm does not perform significantly worse than the average operator scheduling strategy. The positive effect of operator scheduling that we saw for the TSP problem is seemingly less pronounced for the MSA problem. Furthermore, the adaptation graphs (figure 5.2) show that the continued use of the same operator is seemingly not an advantage to the overall progress of the solution for the MSA problem. This effect is particularly apparent for the Subpop problem in figure 5.2, where different operators dominate alternately at extremely short intervals.

In contrast to the TSP problem, where the final phase allowed the EAX operator to steadily fine-tune the solution, no such behaviour is found here. While this could be explained by the argument that such an operator simply was not present in our operator set, the results indicate that there are other issues that might contribute to this difference between TSP and MSA.

In TSP, small changes in the search space often lead to fitness changes of corresponding magnitude. Local manipulation, such as swapping two cities, does not have global consequences (it merely changes two edges in the path). In contrast, the combination of the MSA representation and the employed scoring scheme suggests that operations for this problem are somewhat more disruptive. This is supported by observations on the progress of single runs of the algorithms. It seems that chances for improving the solutions are rather low, and that improvements occur in leaps rather than small steps (an effect that is smoothed out in the average graphs of figure 5.3). An example of a disruptive manipulation is the insertion of one gap, which can result in a whole sequence being misaligned, so that a large change in fitness occurs.

Of course, this preliminary study was not sufficient to support all these statements conclusively. The process of solving MSA problems on the basis of a ClustalW seed, only represents certain assets of the MSA problem. Before any conclusions could be made, a thorough study of the the whole process (from random initialisation to final result) has to be conducted.

As mentioned, operator scheduling had been applied to MSA previously. This was done by Notredame and Higgins in 1996 [52]. Their paper introduced some new heuristic operators and used Davis' adaptive operator scheduling scheme to schedule them. While they reported good results the paper lacked an investigation on the effects of the different elements of the algorithm's design. The limited success of operator scheduling in my

preliminary study of the MSA problem motivated a closer look at this algorithm.

5.4. The SAGA program

In 1996, Notredame and Higgins introduced a new evolutionary algorithm for the MSA problem. Their approach involved a large set of operators, and to ensure that their many operators were optimally applied, an operator scheduling scheme was used to adapt the probabilities of operator usage during the course of the evolutionary process. Intuitively, this approach seems promising. However, the original paper presented no investigation of the effect of this operator scheduling scheme on the performance of the algorithm.

One can argue that the end justifies the means, and that the good performance presented in their paper makes the question of the cause of these improvements irrelevant. However, adaptation in evolutionary algorithms, and specifically adaptive operator scheduling seem widely recognised in the literature as being generally beneficiary. While this might well be true for many problem domains, the results from section 5.3 indicate that this notion might not hold for the particular problem of MSA. Given that SAGA represents a study that claims to have used the method with success, we felt that a closer investigation was justified. This was done by comparing the standard SAGA algorithm with an alternative which chooses operators with a uniform probability.

The study of SAGA was extended beyond the topic of operator scheduling. Two other issues were also investigated:

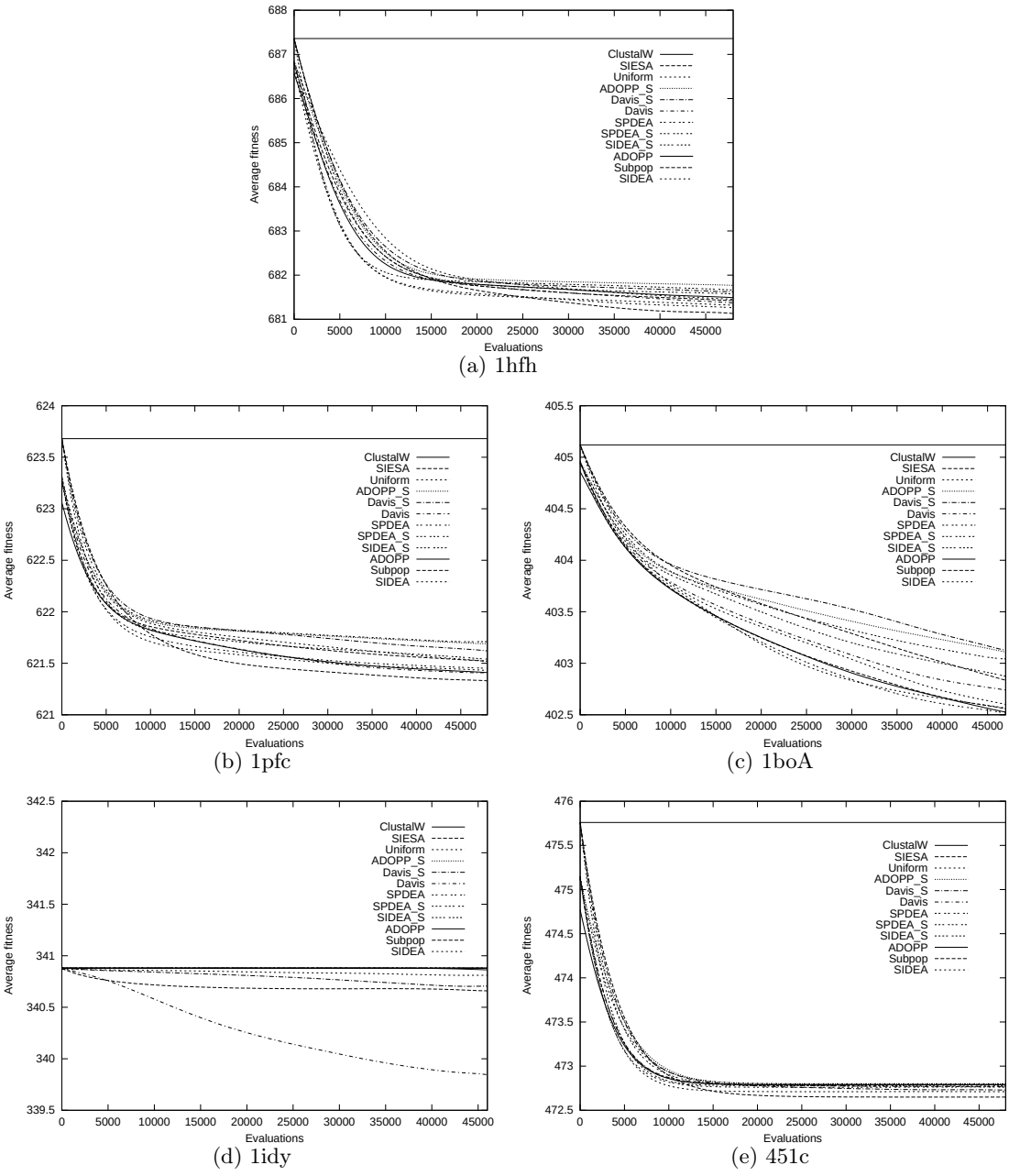
Crossover operators Notredame and Higgins note in their paper that at least some of their crossover operators have a rather disruptive effect on the search process. Our initial investigations using similar crossover operations could confirm this and found that the effect of crossover was generally limited. A comparison of performance of SAGA with and without crossover operators is therefore done.

Seeding The SAGA algorithm (as other evolutionary approaches) is rather slow. To speed it up, the population could be initialised with reasonable alignments produced by an efficient algorithm such as ClustalW. Notredame and Higgins mention this possibility, but discard it on the ground that it would cause the evolutionary algorithm to be trapped in local optima. We investigate here whether this assumption is true.

This chapter is based on a paper written in collaboration with René Thomsen (see appendix F).

5.4.1. Description of SAGA

SAGA distinguishes itself from other EAs for the MSA problem by its large set of operators. A total of 25 variation operators are used – 19 mutation and 6 crossover. These operators are described in the original paper [52].



(a) 1hfh

(b) 1pfc

(c) 1boA

(d) 1idy

(e) 451c

Figure 5.3.: Convergence graphs for the 11 adaptive operator scheduling algorithms on the five MSA benchmark problems. Differences between performance are generally small. Note that the non-adaptive Uniform algorithm performs remarkably well.

For our experiments we used the newest available version of SAGA (SAGA version 0.95[6]). This version provides three objective functions to score alignments with: The SP score, the weighted SP (WSP) score and the COFFEE score. The latter was introduced by Notredame, Holm and Higgins in 1998 [54], and was subsequently added to the SAGA package. Instead of using substitution matrices to score pairs of residues, COFFEE generates a library based on the pairwise alignment of all sequences with ClustalW, and scores matched residues based on their consistency with this library. This was shown to give good results when the considered sequences had only a small degree of similarity. Details on the COFFEE score can be found in the original paper [54]. The SP and WSP functions were described in section 5.2.5.

The original SAGA algorithm terminates when the last 100 generations have not produced any improvements. In order to conduct our experiments, we changed the algorithm to allow termination after a specific number of evaluations. This made the experimentation phase more efficient without introducing a bias towards any specific settings.

5.4.2. Experimental Setup and Results

The parameter settings used in the original SAGA paper were generally adopted without change. The population size was set to 100 and the maximum allowed number of evaluations was 200000. The SP and WSP score function used the BLOSUM-45 substitution matrix and GEP and GOP values of 12 and 8, respectively. Each run was repeated 30 times.

For our experiments, we chose 7 protein sequence data sets from the BAliBASE database. These are listed in table 5.1.

Experiments were done on the configurations corresponding to the different issues of interest (operator scheduling/uniform choice of operators, seeding/non-seeding and crossover/no-crossover). This was done for the three score functions mentioned. Results are displayed in table 5.2. The *Configuration* column shows which configuration is used (abbreviations are explained in the table heading), and the remaining columns give the CS and SPS BAliBASE reference scores for each of the 7 problems. A corresponding table displaying the actual fitness values for the different configurations is given in appendix D. (table D.3). The same appendix contains the standard deviations for the values in table 5.2 (table D.4). The values responsible for the means in table 5.2 were generally shown to be somewhat divergent from a normal distribution. The Wilcoxon Signed Rank test was therefore used to verify whether differences were significant. For details see the original paper in appendix F. A short description of the Wilcoxon Signed Rank test is given in appendix A.

The results do not indicate that operator scheduling gives any improved performance compared to a uniform choice of operators. Seeding generally seems to improve the achieved solution. On four out of seven cases (*1aboA*, *kinase*, *1hfh*, and *1pii*) it outperformed the randomly initialised version of the algorithm. For crossover, results are somewhat inconclusive: it improves performance for *kinase* and *1hfh*, but seems to be a

[6]SAGA V0.95 is publicly available from `http://igs-server.cnrs-mrs.fr/~cnotred/`.

Data set	NSEQ	LSEQ (min,max,avg)	SEQID
1idy	5	(49,58,53.6)	<25%
1aboA	5	(49,80,63.6)	<25%
kinase	5	(263,276,270.2)	<25%
1hfh	5	(116,132,121.2)	20–40%
1pfc	5	(108,117,112)	20–40%
1pii	4	(247,259,251.5)	20–40%
451c	5	(70,87,80)	20–40%

Table 5.1.: The BAliBASE data sets used in the experiments. NSEQ = number of sequences, LSEQ = length of sequences, SEQID = percent residue identity.

slight disadvantage for *1aboA*. In conclusion, using a configuration of seeding, crossover, operator scheduling and SP or WSP, or alternatively: seeding, crossover, uniform-choice and SP or WSP gives the best results.

This optimal configuration was hereafter compared to the SAGA default configuration, the ClustalW (the seed) and T-Coffee[7]. The comparison is again based on the CS and SPS BAliBASE reference scores. Results are displayed in table 5.3. In five out of seven cases, the new configuration outperforms the other algorithms on one of the two evaluation measures. Only in two cases does the default setting of SAGA outperform the one we found, and in these cases the differences are not significant.

5.4.3. Summary of Results of the SAGA Investigation

In our investigation of SAGA we were able to determine a configuration that performed somewhat better than the default settings of the algorithm. Operator scheduling proved to have no effect. Seeding was shown to benefit performance while reducing execution time with a factor of about two. Crossover provided better results in some cases while it decreased performance in one case. Based on the small amount of test cases, no conclusions can however be drawn on its overall effect. For more information on these results, see the paper describing this study (appendix F) and the supplementary material in appendix D.

5.5. Conclusion

The investigation of the SAGA algorithm confirmed the results of the initial investigation: The overall results gave no indication that operator scheduling is an advantage compared to the simple strategy of choosing operators randomly with a uniform distribution. This observation goes against the common conception that operator scheduling is beneficial in any situation where many operators are available. One hypothesis to explain this effect is that the applied operator scheduling schemes were too greedy, and that fluctuations

[7]T-Coffee is a faster heuristic algorithm introduced by Notredame, Higgins and Heringa [53] to avoid the long running times with SAGA. As the name implies, it uses the COFFEE score.

Configuration	Data set 1idy CS/SPS	1aboA CS/SPS	kinase CS/SPS	1hfh CS/SPS	1pfc CS/SPS	1pii CS/SPS	451c CS/SPS
R, C, O, SOP	0.380/0.565	0.484/0.721	0.353/0.606	0.878/0.944	0.991/0.997	0.805/0.895	0.622/0.746
R, C, O, WSOP	0.380/0.550	0.518/0.730	0.304/0.558	0.889/0.950	0.940/0.979	0.805/0.891	0.638/0.744
R, C, O, COFFEE	0.167/0.510	0.570/0.775	0.541/0.734	0.847/0.936	0.893/0.959	0.800/0.882	0.611/0.787
R, NC, O, SOP	0.380/0.540	0.566/0.769	0.272/0.565	0.853/0.931	0.990/0.997	0.775/0.877	0.620/0.717
R, NC, O, WSOP	0.380/0.550	0.562/0.765	0.272/0.558	0.857/0.936	0.940/0.979	0.768/0.873	0.666/0.732
R, NC, O, COFFEE	0.207/0.520	0.570/0.774	0.468/0.705	0.853/0.937	0.890/0.958	0.801/0.883	0.635/0.794
R, C, NO, SOP	0.380/0.550	0.508/0.732	0.362/0.617	0.894/0.951	0.989/0.996	0.805/0.896	0.620/0.744
R, C, NO, WSOP	0.380/0.550	0.496/0.725	0.298/0.569	0.897/0.955	0.940/0.979	0.809/0.893	0.641/0.765
R, C, NO, COFFEE	0.151/0.507	0.570/0.774	0.531/0.732	0.851/0.936	0.891/0.958	0.801/0.883	0.606/0.784
R, NC, NO, SOP	0.380/0.548	0.555/0.764	0.305/0.583	0.863/0.935	0.992/0.997	0.799/0.890	0.600/0.718
R, NC, NO, WSOP	0.380/0.550	0.544/0.758	0.278/0.566	0.859/0.935	0.940/0.979	0.804/0.890	0.627/0.739
R, NC, NO, COFFEE	0.203/0.533	0.569/0.775	0.450/0.692	0.822/0.924	0.890/0.958	0.802/0.884	0.639/0.796
S, C, O, SOP	0.380/0.595	0.566/0.768	0.621/0.792	0.896/0.953	0.985/0.995	0.810/0.899	0.589/0.698
S, C, O, WSOP	0.380/0.543	0.566/0.766	0.627/0.795	0.889/0.950	0.940/0.979	0.810/0.893	0.618/0.709
S, C, O, COFFEE	0.186/0.542	0.570/0.776	0.554/0.737	0.845/0.934	0.890/0.958	0.800/0.882	0.630/0.794
S, NC, O, SOP	0.380/0.596	0.567/0.767	0.531/0.755	0.883/0.947	0.988/0.996	0.810/0.897	0.620/0.711
S, NC, O, WSOP	0.380/0.541	0.568/0.769	0.529/0.752	0.894/0.953	0.940/0.979	0.810/0.892	0.630/0.715
S, NC, O, COFFEE	0.207/0.528	0.570/0.776	0.527/0.729	0.859/0.939	0.892/0.959	0.800/0.882	0.631/0.794
S, C, NO, SOP	0.380/0.594	0.567/0.767	0.608/0.787	0.909/0.959	0.985/0.995	0.810/0.899	0.585/0.696
S, C, NO, WSOP	0.380/0.548	0.567/0.768	0.616/0.800	0.903/0.957	0.940/0.979	0.810/0.892	0.615/0.707
S, C, NO, COFFEE	0.183/0.525	0.570/0.776	0.559/0.736	0.834/0.929	0.892/0.959	0.800/0.882	0.629/0.796
S, NC, NO, SOP	0.380/0.594	0.568/0.766	0.547/0.763	0.890/0.950	0.988/0.996	0.810/0.898	0.605/0.705
S, NC, NO, WSOP	0.380/0.550	0.568/0.767	0.581/0.782	0.878/0.945	0.940/0.979	0.810/0.892	0.624/0.712
S, NC, NO, COFFEE	0.218/0.552	0.570/0.776	0.518/0.726	0.816/0.921	0.893/0.959	0.800/0.882	0.633/0.795

Table 5.2.: BAliBASE evaluation measures on tested benchmarks using different SAGA configurations. *R* (random initialisation), *S* (ClustalW-seed), *C/NO* (crossover/no-crossover), *O/NO* (operator-scheduling/uniform-choice), *SOP/WSOP/COFFEE* (scoring functions).

	SAGA (seeding)		SAGA (random)		ClustalW		T-Coffee	
Data set	CS	SPS	CS	SPS	CS	SPS	CS	SPS
1idy	**0.380**	**0.595**	**0.380**	0.565	**0.380**	0.588	0.000	0.150
1aboA	**0.566**	**0.768**	0.484	0.721	0.540	0.755	0.150	0.570
kinase	0.621	**0.792**	0.353	0.606	**0.630**	0.788	0.600	0.769
1hfh	**0.896**	**0.953**	0.878	0.944	0.770	0.900	0.870	0.938
1pfc	0.985	0.995	**0.991**	**0.997**	0.750	0.903	0.940	0.979
1pii	0.810	**0.899**	0.805	0.895	0.740	0.855	**0.820**	**0.899**
451c	0.589	0.698	0.622	0.746	**0.700**	0.755	0.640	**0.786**

Table 5.3.: BAliBASE evaluation measures on tested benchmarks using different MSA programs. *SAGA(seeding)* is SAGA with the best-found settings and ClustalW seeding (see table 5.1) and *SAGA(random)* is SAGA using default parameter settings and random initialisation. All SAGA results represent mean values obtained from 30 runs.

in the probabilities of the operators thus were too large. Preliminary experimentation with extremely conservative scheduling schemes however did not give better results. It appears, therefore, that the MSA problem is fundamentally different than for instance the TSP problem. It seems that a number of conditions have to be met before operator scheduling is likely to be successful:

It must be possible to apply a measure of quality of the operators during the course of the run. This measure can either be fixed for the entire run or it can change according to certain phases in the solution process, but it must be possible to differentiate operators based on some measurement of quality. This requires operator consistency. The operator should be associated with a certain type of action that on average has a certain probability of providing a good solution. If too much randomisation is incorporated into an operator, it reduces consistency and makes it harder to assign a quality value to the operator. This is especially true if the functionality of the different operators overlaps (which is often the case). Operators are scheduled based on fortunate operations (producing good offspring). If operations are fortunate in the sense that they are essentially random, the operator scheduling algorithm (or a human for that matter) has no chance of improving overall performance by giving some operators a high probability.

Consistency must however also be present in the search space itself. If operators consistently use similar operations, but these operations give radically different results, then the success of an operation might be as much an artifact of the current position in the search space as of the quality of the operator itself. It would seem that for the MSA problem this might be an issue, since small changes in the alignment can result in large changes of fitness.

6. Numerical Optimisation

As a final investigation, it seemed natural to explore the effects of operator scheduling on the problem domain of numerical optimisation. This is a classic area for evolutionary computation and many new algorithms use this domain for testing purposes. Problems are easily available, and for these standard problems, results of one algorithm can easily be compared to the results of others.

Differential Evolution (presented in section 2.2.4) has in recent years been shown to be a very efficient algorithm for this domain [67]. Apart from a comparison of the different operator scheduling schemes I will therefore also consider the effects of incorporating the DE variation operator as one of the scheduled operators.

6.1. Investigating the Effect of Operator Scheduling

The set of operators for this study was primarily composed of a number of well-known operators for numerical problems. I also included a few other operators that are not as widely used, but which gave reasonably good results in initial investigations.

Gaussian mutation adds normally distributed values to all elements of an individuals representation. The mean of this distribution is zero. In my implementation the variance is slowly decreased as the inverse of the amount of generations. This way, the mutation will take smaller and smaller steps at the search progresses.

Cauchy mutation works as the Gaussian mutation, but here the random values are distributed under the Cauchy distribution.

SOC mutation is a mutation operator by Krink, Rickers and Thomsen [41]. It uses a normal distribution, where the variance is based on a value from the power law distribution. This results in mutations that will most often be small, with only an occasional large step.

Uniform crossover was presented in section 2.2.3. For each component in the child's genotype, a random choice is taken between the corresponding components of the two parents.

One-point crossover A cut-point is chosen. All components up to this cut-point are copied from the first parent and the remaining are copied from the second.

Two-point crossover uses two cut-points and is a simple generalisation of One-point crossover. See section 2.2.3

Arithmetic crossover For each component x_i in the child a weighted average of components of the two parents p_1 and p_2 is computed as $x[i] = r_i * p_1[i] + (1 - r_i) * p_2[i]$, where r_i is a random value between 0 and 1. A new random value is computed for each component.

Random-pool crossover Each component is copied from a random parent in the population (note that this operator combines information from potentially all individuals in the population).

Single-gene arithmetic crossover is a combination of arithmetic crossover and uniform crossover. Only for one component is a weighted average computed. The rest is transferred randomly from the two parents.

The parameters for the above operators were roughly tuned based on individual runs of an evolutionary algorithms using only one operator at a time. The focus of this investigation is the effect of operator scheduling, and not the performance of the operators themselves, and I will therefore not go into the details of the exact parameter settings. It suffices to say that the same settings were used for all experiments.

Preliminary experiments showed that contrary to the experiences with TSP, the performance of the algorithms did not improve for larger population sizes. These were therefore set to 100. Large elite sizes also seemed to reduce performance. Elite sizes were therefore reduced to 5–10 individuals. The parameters of the adaptive operator scheduling schemes were set as described in section 3.2.2 with the exception of the Subpop algorithm, for which it was necessary to reduce the frequency and size of migration due to the smaller population sizes used ($MI = 16$ generations, $MA = 2$ individuals).

In the studies on TSP and MSA, I used a selection method known from differential evolution. In each generation a new population was created and filled up position by position. For each position a newly created individual was only allowed to replace the old individual in that position if it had better fitness. This method turned out to give inferior results for the problems considered here. For this study I therefore used the more traditional strategy of allowing every new individual to enter the population. Still, no explicit selection phase was used. The elitist strategy was sufficient to keep the best individuals and weed out the worst.

Experiments were done on a small sample of numerical optimisation functions (see table 6.1). For each of these function definitions, I used settings of 20, 50 and 100 dimensions to create 9 problems in all. All functions have a minimum value of 0.

All 11 adaptive operator scheduling schemes were applied to each problem. For comparison, I included a single-operator algorithm using only the DE variation operator. For the Griewank and Rastrigin problems the number of evaluation used was set to 20000 times the dimensionality of the problem, thus $400,000$, $1,000,000$ and $1,5000,000$ evaluations, respectively. The Rosenbrock problem was noted to be more difficult, and here $500,000$, $1,500,000$ and $5,000,000$ evaluations were used. Average results over 50 runs are presented in table 6.1. Representative graphs for adaptation and convergence are found in figures 6.2 and 6.3.

Griewank	$f(\vec{x}) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$-600 < x_i < 600$
Rastrigin	$f(\vec{x}) = \sum_{i=1}^{n} x_i^2 + 10 - 10 \cdot \cos(2\pi x_i)$	$-5.12 < x_i < 5.12$
Rosenbrock	$f(\vec{x}) = \sum_{i=1}^{n-1} 100 \cdot (x_{i+1} - x_i^2)^2 + (x_i - 1)^2$	$-100 < x_i < 100$

Figure 6.1.: The three benchmark functions used for experiments.

From the results in table 6.1 it is clear that none of the algorithms can compete with the Differential Evolution algorithm. This particular algorithm is however notorious for it good performance on these problems so this comes as no great surprise.

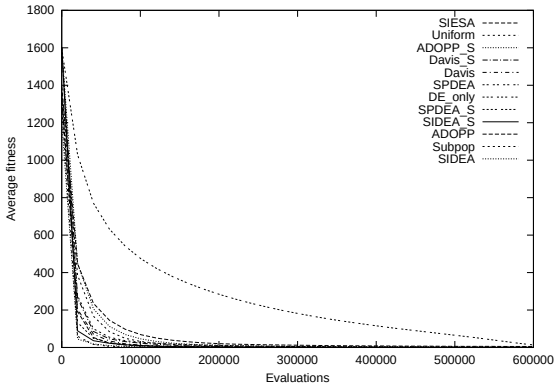


Figure 6.2.: Convergence of the different algorithms on the Rastrigin_100d problem.

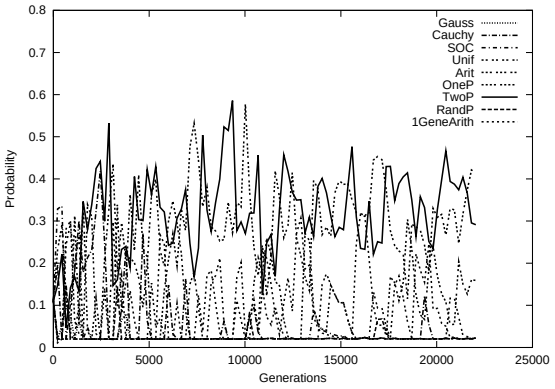


Figure 6.3.: Distribution of operator probabilities during the course of the run. Clearly, a few operators are consistently scheduled higher than the rest.

A natural next step was to include the DE variation operator in the pool of operators. One could expect that as for the EAX operator on TSP, there are certain phases of the solution process which might benefit from alternative operators, and perhaps the convergence speed of the DE algorithm could be improved by incorporating it in an adaptive operator scheduling algorithm.

The DE variation operator is however tightly integrated into the DE algorithm. Simply including the variation operator into the pool of operators did not work as intended. The DE operator did not receive credit from the scheduling process at all, and results were similar to that of table 6.1. The most likely explanation was that the selection scheme was different from that of the DE algorithm, and apparently, this was fatal for the DE variation operator.

To verify this claim I changed the selection scheme back to the DE-inspired one used for TSP and MSA. This time, the differential evolution operator performed well. However, as stated before, this was not fair to the other operators, that apparently were better off

without the DE-selection scheme.

I devised the following compromise: Every time a DE operator was used the usual DE selection methods was applied (i.e. the individual had to outperform the individual currently residing at that position in the population). Every time any other operator was used, the produced individual was included in the new population regardless of its fitness. This way, if the DE variation operator dominated the search, selection would mainly be done on its terms, and vice versa for the other operators. Again, the algorithms were tested on all nine problems. Results are found in table 6.2. Standard deviations of both tables can be found in appendix E.

6.2. Discussion

By comparing table 6.1 and 6.2 it is clear that the results with the expanded operator pool are generally not better than they were without the DE operator. The potential of the DE operator is clearly not properly exploited. It seems that the population diversity that the DE normally succesfully maintains is destroyed by the actions of the other operators, leading the search to local optima.

One interesting case is that of the Subpop algorithm. This is actually the only scheduling algorithm than in some cases manages to exploit the full potential of the DE operator. This might be explained by the fact that the Subpop algorithm maintains different populations for each operator, allowing distinct search processes to exist side by side with minimal interference. Subpop is also the only adaptive operator scheduling algorithm that consistently schedules the DE operator highly throughout the search process. This makes it clear that the adaptive algorithms can only appreciate the value of the DE operator when it is conducting its own private search – not when searching together with the other operators. Why the Subpop algorithm only manages to do this for certain problems is a question I cannot answer at this point. Further experiments with the operator should be done to determine if it can be made more robust.

The exact interaction between the different search operations is not completely clear. It is however clear that we here have an example of an incompatible operator that does not benefit in any way from an adaptive operator scheduling strategy. Even worse, performance is significantly reduced by the use of such an approach on this problem. This is the first example of a situation where operator scheduling is not as "safe-and-sound" as it is often portrayed.

Another observation that can be made from the data is the remarkable performance of the Uniform operator. It performs at similar level as the average adaptive operator scheduling scheme. This behaviour was also seen for the MSA problem, where I attributed it to inconsistencies in the search. There is however nothing suggesting that this would be the reason here. When looking at the adaptation graphs (e.g. figure 6.3), I found that the algorithms do manage to find the best operators, and that these patterns are rather consistent throughout the run. Ironically, the problem might simply be that the scheduling algorithms are too successful. The benchmark problems in table 6.1 have been designed to be as hard as possible, with a multitude of local optima. If a scheme is too

greedy and converges too fast, it is often seen to get stuck in rather poor local optima. For these specific problems, it seems that the best strategy is to slowly let the search progress, maintaining as diverse a population as possible, and thereby avoiding local optima. This is supported by figure 6.2, showing the convergence of the different operator scheduling algorithms compared to the DE_only algorithm. It is immediately clear that the DE-algorithm has significantly slower convergence than any of the scheduling algorithms. However, it clearly produces the best results. This suggests that rewarding an operator for its performance might be a fundamentally flawed strategy. Selecting the best operators will improve convergence speed, but can have negative effects on the achieved results. For the benchmark problems in this study, maintaining a diverse population is essential, and the operator scheduling schemes might be counterproductive on this front. Similar patterns were observed by Tuson and Ross [76] on the Long Path problem [33] using operator scheduling between one mutation and one crossover operator.

	griew_20d	griew_50d	griew_100d	rastr_20d	rastr_50d	rastr_100d	rosenb_20d	rosenb_50d	rosenb_100d
Davis_S	3.09e-02	1.16e-02	5.74e-03	4.98e-04	1.20e-03	2.98e-03	1.81e+01	6.27e+01	1.28e+02
Davis	2.61e-02	5.97e-03	3.83e-03	1.05e-05	9.79e-05	4.32e-04	1.87e+01	5.66e+01	1.12e+02
SIDEA_S	3.76e-02	6.50e-03	4.48e-03	2.51e-04	5.48e-04	1.04e-03	1.95e+01	5.73e+01	1.12e+02
SIDEA	2.74e-02	5.62e-03	4.14e-03	3.58e-05	3.39e-05	1.02e-05	1.66e+01	5.25e+01	8.58e+01
SPDEA_S	2.90e-02	9.37e-03	6.96e-03	2.50e-04	9.13e-04	2.07e-03	1.69e+01	5.38e+01	1.23e+02
SPDEA	2.24e-02	5.29e-03	3.72e-03	2.50e-06	7.66e-06	2.74e-05	1.86e+01	5.34e+01	9.58e+01
SIESA	2.99e-02	7.14e-03	5.88e-03	8.41e-05	3.70e-04	8.09e-04	2.34e+01	4.66e+01	1.18e+02
Subpop	2.92e-02	6.40e-03	3.45e-03	1.88e-06	9.40e-07	3.40e-07	2.51e+01	8.91e+01	1.77e+02
Uniform	2.63e-02	9.53e-03	4.23e-03	2.74e-04	8.64e-04	2.12e-03	1.92e+01	5.05e+01	1.16e+02
ADOPP_S	3.58e-02	1.59e-02	1.18e-02	1.06e-03	3.01e-03	1.12e-01	2.84e+01	6.38e+01	1.39e+02
ADOPP	2.67e-02	6.98e-03	7.78e-03	2.72e-04	8.57e-04	3.75e+00	1.63e+01	4.71e+01	1.81e+02
DE_only	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00

Table 6.1.: Average best results over 50 runs for experiments without the DE variation operator in the operator pool.

	griew_20d	griew_50d	griew_100d	rastr_20d	rastr_50d	rastr_100d	rosenb_20d	rosenb_50d	rosenb_100d
Davis_S	2.93e-02	9.68e-03	6.15e-03	5.36e-04	1.31e-03	2.84e-03	1.76e+01	5.82e+01	1.19e+02
Davis	3.19e-02	4.82e-03	3.45e-03	1.20e-05	1.00e-04	4.61e-04	1.84e+01	5.27e+01	1.20e+02
SIDEA_S	2.66e-02	7.96e-03	4.86e-03	2.29e-04	6.24e-04	1.08e-03	1.83e+01	5.38e+01	1.21e+02
SIDEA	2.32e-02	4.90e-03	3.01e-03	2.22e-05	1.39e-05	7.84e-06	1.67e+01	5.29e+01	6.64e+01
SPDEA_S	3.29e-02	8.08e-03	5.63e-03	2.75e-04	1.00e-03	2.17e-03	1.68e+01	5.01e+01	1.19e+02
SPDEA	2.28e-02	6.18e-03	3.79e-03	9.02e-06	1.10e-05	6.71e-05	1.63e+01	5.06e+01	9.68e+01
SIESA	3.35e-02	8.87e-03	5.85e-03	1.15e-04	4.15e-04	1.00e-03	2.04e+01	5.06e+01	1.12e+02
Subpop	2.95e-03	2.56e-03	3.40e-03	0.00e+00	1.19e-01	1.99e-02	0.00e+00	3.42e-01	2.80e+01
Uniform	2.66e-02	6.84e-03	6.12e-03	2.86e-04	1.09e-03	2.25e-02	2.19e+01	4.75e+01	1.18e+02
ADOPP_S	3.49e-02	1.54e-02	1.30e-02	1.17e-03	2.93e-03	1.50e-01	2.41e+01	6.05e+01	1.45e+02
ADOPP	3.97e-02	1.02e-02	7.42e-03	3.01e-04	9.77e-04	2.07e+00	1.70e+01	4.75e+01	1.43e+02

Table 6.2.: Average best results over 50 runs for experiments with the DE variation operator included in the operator pool.

7. Conclusion

The rising popularity of evolutionary algorithms through the last decades has produced a large number of problem specific operators. Ideally, the best of these operators should be applied whenever applicable. The problem is that there is often no easy way of determining the performance of an operator based on the descriptions in the literature, and even if there was, the performance of operators varies across different problem instances. The evolutionary programmer is thus forced to do some sort of comparison for himself. This can be done manually, but requires significant amounts of time. Alternatively, an adaptive operator scheduling scheme can be used to automate the process.

A variety of such adaptive operator scheduling algorithms are available, but since no comparisons between the different schemes have been made, it is difficult to choose which one to use. The papers presenting the different methods also disagree on the effect of adaptive operator scheduling. Some present improvements, while others conclude that their scheme has little effect.

This thesis was an attempt to clarify some of these issues. Apparently, adaptive operator scheduling has dependencies on both the problem domain and the adaptive operator scheduling scheme used. With various studies I have tried to isolate which of these factors are dominant. I have also compared different operator scheduling schemes on the same problems to identify general differences in performance and determine whether certain scheduling schemes might have advantages on certain problem domains. Finally. I investigated the issue of absolute performance. When EAs using operator scheduling are compared to EA using only one or two fixed operators operators, any observed improvements are composed of two contributions: the positive effects of having a large number of operators available and the effects of operator scheduling. To isolate the latter all algorithms were compared to an algorithm choosing randomly from the pool of operators (the Uniform algorithm).

My studies on the Traveling Salesman Problem showed that the various adaptive operator scheduling schemes performed about equally well. The Subpop algorithm stood out as the algorithm that produced the best results most often, but since the design of this algorithm differs somewhat from that of the others (because of multiple populations), slightly different parameter settings were used (population size, elite size) and the differences could be ascribed to this fact. The availability of many operators was clearly very beneficial for the algorithms. Compared to the algorithm using only the EAX operator huge improvements were observed. The non-adaptive Uniform algorithm however also performed quite well. It was worse than all the adaptive variants, but much better than the algorithm using only the EAX operator.

Also for the MSA problem the different adaptive operator scheduling schemes showed similar performance. Again the Subpop algorithm was on average slightly better than the others. The most surprising result was perhaps that in this study, the performance of the Uniform algorithm was even closer to that of the adaptive algorithms. In some cases it actually had better performance than some of the adaptive algorithms. These results indicated that operator scheduling apparently had little effect on the MSA problem. This was confirmed by the subsequent investigation on the SAGA algorithm.

For the numerical optimisation benchmark functions, there was some more variation between the performance of different adaptive operator scheduling methods. No algorithm was however better than the others over all problems. The subsequent inclusion of the Differential Evolution operator led to disappointing results. It was however interesting to observe that an apparent incompatibility between the DE operator and the rest of the operators resulted in significantly reduced performance. The only adaptive operator scheduling algorithm that was capable of exploiting the potential of the DE operator was the Subpop algorithm. This algorithm maintains more distinct populations, allowing the DE search to operate without being disturbed by the other operators.

Also for the the numerical benchmarks the Uniform algorithm produced surprisingly good results, not significantly worse than that of the different adaptive operator scheduling schemes. Contrary to the MSA problem, scheduling did in fact occur, and certain algorithms were consistently scheduled higher than others. It was therefore somewhat unexpected that no significant improvements are made. One explanation is that this is due to a fundamental flaw of adaptive operator scheduling schemes. Isolating a few well-performing operators and increasing the frequency of their use also increases the greediness of the search. For the numerical benchmark functions in chapter 6, maintaining a diverse population seems to be essential in order to achieve good results. Operator scheduling might in cases as these be directly counterproductive.

Generally, there was little difference in performance between the different operator scheduling schemes used. Combining all operators in one large pool seems to be somewhat better than splitting them up in two distinct pools (i.e. one for mutation and one for crossover). The Subpop algorithm seems to be slightly better on average if a large enough population is used. Even in situations where different operators disrupt each others search, the Subpop algorithm is in some cases able to exploit the potential of all operators by providing separate populations for each of them. Given there similar performance, it seems unnecessary to implement schemes with significant bookkeeping such as that of Davis. Instead, the self-adaptive variants SIDEA and SPDEA might constitute a compelling alternative. They are easily implemented and rely only on a single parameter.

The studies however also show that much increased performance can be gained simply by adding many operators to an evolutionary algorithm. The performance might in some cases be increased further by introducing a operator scheduling scheme, but choosing the operators randomly generally gives very reasonable results with very few implementation efforts.

7.1. Future work

The conclusions of this thesis would we strengthened if more problem domains had been investigated. It would be interesting to see whether the Uniform algorithms generally gives satisfactory results or whether situations exist where adaptive operator scheduling is essential to achieve good performance with a large set of operators.

The adaptive operator scheduling schemes chosen for the test suite were selected for their implementation simplicity. This decision was made based on the observation that if adaptive operator scheduling was to be a practical alternative to the manual selection of operators, it should be easy to implement. If alternative methods could provide significant improvements in performance, this could however be a motivation to implements a more elaborate operator scheduling strategy. The adaptation of operators can in principle be based on any information in the current or past states of the population. One could imagine arbitrarily complex schemes using machine learning techniques to find patterns in the data. The question is, however, if these schemes would be more successful. In section 5.5 I discussed some issues that might explain the poor performance of operator scheduling methods for the MSA problem. I proposed that inconsistency, both in the actions of the operators and in the search space itself, were fatal to the adaptive operator scheduling process. Based on these observations, it is not clear that more advanced adaptive operator scheduling schemes would have much greater success. In the case where the current ones fail, the reason seems to be a lack of consistency in the search. This problem applies to any operator scheduling scheme. If there is no consistency in the action of the operators, it is simply impossible to design a good operator scheduling scheme, however advanced it might be. It can however not be ruled out that certain machine learning techniques can deduce meaningful patterns from the search and use these to adjust operator probabilities. Additional studies will have to show.

A. Statistical methods

Given the stochastic nature of evolutionary algorithms, experiments must be repeated a number of times before any conclusions on their performance can be drawn.

In the various studies of this paper, all results have been presented as *mean* values over some amount of repetitions. The mean M for over n independent observations x_i can be calculated as

$$M = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{A.1}$$

The mean value provides information about what result we on average can expect from an algorithm, but says nothing about how much deviation from this mean we can expect from single executions. It is often interesting to know how tightly the observations are clustered around the mean. For this purpose, we use the *standard deviation*, which is defined as

$$SD = \sqrt{\frac{\sum (x_i - M)^2}{n - 1}} \tag{A.2}$$

Another interesting question might how accurate our estimate of the mean is. As mentioned, each mean value is based on an experiment consisting of a number of runs. We are interested to know how much these mean values vary if we were to repeat the same experiment a number of times. This can be understood as the standard deviation of the mean, but is normally called the *standard error* (SE). It can be calculated as

$$SE = \frac{SD}{\sqrt{n}} \tag{A.3}$$

If our results are distributed as a normal distribution, we can now calculate the *confidence interval* (SI) for our mean. The confidence interval is a range where we expect to find the true value of the mean with a certain probability. Often 95% confidence intervals are used, thus indicating that within this range, there is a 95% chance of finding the true value of the mean.

$$CI_{95} = M \pm (1.96 * SE) \tag{A.4}$$

Naturally, small intervals indicate that we have good confidence in the value, while large intervals suggest that more repetitions of the run might be necessary.

For all experiments in this thesis, the standard deviations have been reported. It is also mentioned whether the observations were approximately normally distributed. Based on this information, confidence intervals can help compare different mean values. If the intervals for two values do not overlap, we can conclude that there is a statistically significant difference between them. Since confidence intervals are rather easily calculated from the standard deviations, they have not been explicitly reported in this thesis.

For results that are not normally distributed (A.4) cannot be used. Other so-called non-parametric methods exist that are able to compare groups of observations by other means.

In the study of SAGA (section 5.4) the *Wilcoxon Rank Sum* test was used to analyse the results. This method can compare two paired groups. It calculates the differences between each pair, and based on these differences, it finds a probability value P that denotes the probability that the observed differences could have been observed if the two groups were in fact drawn under the same distribution. The methods thus makes it possible to investigate if two groups are significantly different. A high value of P indicates high similarity, while a low value indicates that there are significant differences between the two groups. In the study of SAGA the pairs were composed of the mean results of two experiments where only a single parameter was changed. An example of such a pair could be (R,C,O,SOP; S,C,O,SOP) (see table 5.2).

The choice of the Wilcoxon Rank Sum test was motivated by the descriptions on `http://www.graphpad.com/instatman/instat3.htm`.

B. Source code

The source code that was produced during the studies in this thesis is available from

`www.daimi.au.dk/~wb/thesis/`

All code was written in C++.

C. Supplementary material: TSP

	gr48	brg180	pcb442	nrw1379	pr2392	pcb3038
Davis_S	0.000	1.714	1.200	20.441	39.931	23.284
Davis	2.057	0.000	19.580	18.298	40.001	22.936
SIDEA_S	6.721	1.407	0.985	19.042	47.249	34.634
SIDEA	3.249	1.714	0.700	26.403	361.027	38.483
SPDEA_S	2.408	1.970	1.200	20.307	67.286	39.168
SPDEA	0.000	1.714	0.700	17.051	42.652	26.057
SIESA	5.569	4.845	21.478	21.457	58.394	35.303
Subpop	3.035	2.387	0.985	19.157	30.749	27.272
Uniform	7.628	4.513	2.366	36.556	393.276	69.963
ADOPP_S	3.970	3.861	1.909	20.489	73.642	31.061
ADOPP	3.970	3.775	1.533	17.069	57.580	24.824
EAX_only	0.000	3.380	1170.146	2963.523	47395.058	10385.276

Table C.1.: Standard deviation values for the mean values in table 4.4. The values that the means are based on were shown to be distributed roughly as the normal distribution. Confidence intervals can be calculated based on the equation in appendix A.

D. Supplementary material: MSA

D.1. Operators used in the MSAEA

This is a list of the operators used in the initial experiments on the MSA problem. Note that the functionality of many of these operators overlap. It is however not clear in advance which operators will prove to be most valuable.

Swap Choose a random sequence and an index for which a swap between a residue and a gap-symbol is possible. Execute the swap.

LocalShuffle Choose a random sequence and an index. If the sequence has a gap as neighbour, execute the swap. If there are gaps on both sides, choose one randomly

DirectedLocalShuffle A directed version of localShuffle. If there are gaps on both sides of the residue, chose the column for which this operation gives highest column score.

BlockShuffle Choose a random sequence and a random block of consecutive residues. Move the block one position to the left or to the right if there is a gap there.

PassGaps Choose a random sequence. Identify all gap-regions in the sequence, and choose one of them randomly. Move one residue from one side of the gap to the other (in a randomly chosen direction).

MoveGap Choose a random sequence. Choose a random gap-symbol and move it to a random position between the first and the last residue.

RandomMoveGap Choose a random sequence. Choose a random gap, ignoring any terminal gaps. Move this gap anywhere between the first an the last residue in the sequence.

InsertGaps Choose a random sequence. Pick a random number in the range $[1; 5]$. Take a corresponding amount of terminal gaps (if any) and distribute them consecutively somewhere between the first and last-occuring residue.

RemoveSingletonGap Identify all single gaps in the alignment. Choose one. Move it to the right side of the alignment (making it a terminal gap).

ExtendGaps Choose a random sequence. Identify all gap-regions. Choose one of them randomly and add one gap to this region (taking it from the terminal gaps).

DecreaseGaps Choose a random sequence. Identify all gap-regions. Choose one randomly and remove one gap from it (making it a terminal gap).

ClusterGaps Choose a random sequence. Identify all gap-regions and all single-gaps. Merge a random single-gap with a random gap-region that has a length greater than 1.

GrowMatchedColumns Select a fully aligned column (if any). Swap residues with gaps in the different sequences in an attempt to fully align an adjacent column.

UniformRowCrossover Merge two parents by randomly selecting rows from each of them.

OnePointRowCrossover Select a row as cut-point. All rows up to this row are taken from one of the parents and the rest is taken from the other. This can be done in two ways to create two children. The best of these is used.

OnePointColumnCrossover Select a column as cut-point in parent1. Find the residues from this column in parent2 (these will typically not be in a straight column). Truncate all sequences from parent1 at the cut-point and transfer them to child 1. Now take the appropriate suffixes of the sequences in parent2 and transfer them to child1. The fact that these suffixes do not start in the same column is solves by adding gaps to the shortest ones (the structure of columns from parent2 are thus maintained). Again, this can be done in two ways to create two children. The best of these is chosen.

BiasedOnePointColumnCrossover Similar to onePointColumnCrossover. If possible, fully aligned columns are chosen as cut-points.

	1hfh	1pfc	1aboA	1idy	451c
Davis_S	681.62 (0.903|0.771)	621.62 (0.904|0.752)	403.07 (0.754|0.536)	340.70 (0.580|0.376)	472.73 (0.740|0.678)
Davis	681.33 (0.900|0.762)	621.40 (0.902|0.749)	402.70 (0.755|0.531)	339.76 (0.557|0.379)	472.77 (0.733|0.664)
SIDEA_S	681.65 (0.907|0.776)	621.55 (0.905|0.755)	402.86 (0.763|0.539)	340.88 (0.588|0.380)	472.78 (0.734|0.666)
SIDEA	681.28 (0.904|0.769)	621.43 (0.905|0.756)	402.56 (0.765|0.540)	340.88 (0.588|0.380)	472.71 (0.745|0.688)
SPDEA_S	681.58 (0.906|0.776)	621.70 (0.904|0.753)	403.02 (0.756|0.537)	340.88 (0.588|0.380)	472.77 (0.740|0.678)
SPDEA	681.24 (0.903|0.767)	621.45 (0.905|0.756)	402.48 (0.764|0.538)	340.88 (0.588|0.380)	472.76 (0.741|0.681)
SIESA	681.42 (0.904|0.769)	621.50 (0.905|0.755)	402.50 (0.767|0.540)	340.88 (0.588|0.380)	472.79 (0.733|0.663)
Subpop	681.08 (0.902|0.766)	621.34 (0.905|0.757)	402.77 (0.760|0.539)	340.65 (0.583|0.380)	472.64 (0.754|0.708)
Uniform	681.43 (0.905|0.773)	621.51 (0.904|0.753)	402.54 (0.765|0.540)	340.81 (0.586|0.380)	472.80 (0.733|0.664)
ADOPP_S	681.75 (0.906|0.777)	621.68 (0.904|0.753)	403.07 (0.758|0.539)	340.88 (0.588|0.380)	472.80 (0.735|0.668)
ADOPP	681.48 (0.905|0.771)	621.41 (0.905|0.756)	402.49 (0.764|0.538)	340.86 (0.587|0.380)	472.79 (0.736|0.671)
ClustalW	687.36 (0.900|0.770)	623.68 (0.903|0.750)	405.12 (0.755|0.540)	340.88 (0.588|0.380)	475.76 (0.755|0.700)

Table D.1.: Results for the initial investigation of the MSA problem using the MSAEA operators. The entries represent average results over 100 repetitive runs. The format of the entries is `fitness (SP score|CS score)`, where `fitness` is the fitness value calculated by the EA's scoring function, while the `SP score` and the `CS score` are BAliBASE reference scores (calculated using annotation files).

	1hfh	1pfc	1aboA	1idy	451c
Davis_S	0.41 (0.233|0.203)	0.27 (0.315|0.262)	0.74 (0.278|0.201)	0.60 (0.035|0.038)	0.14 (0.266|0.246)
Davis	0.40 (0.296|0.257)	0.23 (0.260|0.222)	0.52 (0.239|0.172)	1.16 (0.165|0.110)	0.10 (0.178|0.171)
SIDEA_S	0.59 (0.156|0.134)	0.26 (0.198|0.166)	0.89 (0.197|0.139)	nan (nan|nan)	0.07 (0.010|0.021)
SIDEA	0.91 (0.260|0.222)	0.25 (0.260|0.218)	0.75 (0.221|0.155)	nan (0.059|0.038)	0.12 (0.204|0.192)
SPDEA_S	0.42 (0.285|0.245)	0.28 (0.284|0.237)	0.83 (0.229|0.163)	nan (nan|nan)	0.06 (0.191|0.177)
SPDEA	0.89 (0.216|0.184)	0.27 (0.260|0.218)	0.68 (0.221|0.155)	nan (0.129|0.083)	0.07 (0.243|0.225)
SIESA	0.51 (0.247|0.211)	0.20 (0.155|0.130)	0.56 (0.133|0.093)	nan (0.059|0.038)	0.05 (0.126|0.115)
Subpop	0.89 (0.315|0.267)	0.21 (0.178|0.149)	0.57 (0.196|0.138)	0.70 (0.061|0.038)	0.14 (0.194|0.185)
Uniform	0.58 (0.155|0.134)	0.24 (0.003|0.008)	0.53 (0.079|0.054)	0.41 (0.083|0.053)	0.01 (0.103|0.094)
ADOPP_S	0.43 (0.006|0.016)	0.26 (0.003|0.007)	0.81 (0.018|0.004)	nan (nan|nan)	0.02 (0.010|0.020)
ADOPP	0.52 (0.007|0.019)	0.22 (0.003|0.009)	0.60 (0.019|0.012)	0.20 (0.008|nan)	0.04 (0.011|0.024)

Table D.2.: Standard deviation values for the mean values in table 4.4. The values that the means are based on were shown to be distributed roughly as the normal distribution. Confidence intervals can be calculated based on the equation in appendix A. The nan values arise from results with no variance at all.

Configuration	Data set						
	1idy	1aboA	kinase	1hfh	1pfc	1pii	451c
R, C, O, SOP	33590.800	39760.933	162894.400	67543.733	61530.800	131941.400	46954.933
R, C, O, WSOP	2744.000	3550.700	4397.567	5069.267	4227.000	6604.567	3796.333
R, C, O, COFFEE	47096.500	61817.267	58860.867	82306.833	85316.300	86837.900	64538.800
R, NC, O, SOP	33594.667	39685.467	163498.933	67632.533	61532.533	132291.367	46966.267
R, NC, O, WSOP	2744.000	3542.500	4403.867	5079.100	4227.000	6624.867	3797.700
R, NC, O, COFFEE	47044.333	61774.300	57656.000	82192.167	85315.167	86822.100	63993.067
R, C, NO, SOP	33592.000	39730.400	162830.533	67445.733	61532.000	131929.967	46939.067
R, C, NO, WSOP	2744.000	3549.233	4395.167	5066.800	4227.000	6601.500	3796.700
R, C, NO, COFFEE	47049.567	61783.767	58853.633	82162.833	85325.100	86837.900	64593.600
R, NC, NO, SOP	33592.533	39673.200	163282.133	67600.400	61532.000	132086.667	46971.200
R, NC, NO, WSOP	2744.000	3542.667	4399.800	5078.933	4227.133	6609.533	3796.700
R, NC, NO, COFFEE	46940.400	61788.333	56668.900	81901.400	85279.133	86807.833	63634.167
S, C, O, SOP	33592.000	39670.267	162084.000	67402.533	61532.000	131862.000	46995.067
S, C, O, WSOP	2744.400	3541.633	4366.633	5068.867	4227.000	6598.433	3795.000
S, C, O, COFFEE	47089.767	61703.300	59237.467	82261.367	85317.500	86838.267	64389.833
S, NC, O, SOP	33591.467	39678.800	162241.200	67478.933	61532.000	131873.033	46990.933
S, NC, O, WSOP	2744.267	3541.033	4367.967	5069.133	4227.000	6598.267	3797.467
S, NC, O, COFFEE	47069.200	61818.033	58619.467	82195.833	85306.333	86836.800	63803.033
S, C, NO, SOP	33592.000	39668.933	162075.200	67359.733	61532.000	131862.633	46999.867
S, C, NO, WSOP	2744.433	3541.233	4366.733	5063.833	4227.000	6598.233	3795.533
S, C, NO, COFFEE	47046.200	61702.500	58892.200	82228.700	85325.167	86838.633	64394.367
S, NC, NO, SOP	33592.000	39675.467	162153.867	67426.533	61532.000	131869.600	46976.267
S, NC, NO, WSOP	2744.000	3541.233	4365.900	5071.233	4227.000	6598.233	3794.500
S, NC, NO, COFFEE	46967.700	61825.833	58232.667	81960.433	85272.133	86835.700	63506.867

Table D.3.: Fitness scores on tested benchmarks using different SAGA configurations (Mean of overall best alignment in 30 runs). *R* (random initialisation), *S* (ClustalW-seed), *C/NO* (crossover/no-crossover), *O/NO* (operator-scheduling/uniform-choice), *SOP/WSOP/COFFEE* (scoring functions).

Configuration	Data set 1idy CS/SPS	1aboA CS/SPS	kinase CS/SPS	1hfh CS/SPS	1pfc CS/SPS	1pii CS/SPS	451c CS/SPS
R, C, O, SOP	0.000/0.056	0.109/0.064	0.077/0.069	0.054/0.026	0.010/0.004	0.029/0.015	0.090/0.047
R, C, O, WSOP	0.000/0.000	0.100/0.053	0.075/0.091	0.052/0.027	0.000/0.000	0.031/0.017	0.123/0.063
R, C, O, COFFEE	0.080/0.067	0.000/0.005	0.029/0.012	0.037/0.015	0.016/0.006	0.000/0.000	0.038/0.015
R, NC, O, SOP	0.000/0.045	0.044/0.026	0.031/0.034	0.041/0.021	0.010/0.004	0.067/0.035	0.110/0.044
R, NC, O, WSOP	0.000/0.000	0.044/0.028	0.039/0.049	0.045/0.021	0.000/0.000	0.065/0.033	0.077/0.037
R, NC, O, COFFEE	0.064/0.052	0.000/0.005	0.049/0.026	0.034/0.013	0.000/0.000	0.003/0.003	0.047/0.017
R, C, NO, SOP	0.000/0.014	0.103/0.060	0.081/0.056	0.037/0.018	0.010/0.004	0.026/0.014	0.104/0.054
R, C, NO, WSOP	0.000/0.000	0.105/0.054	0.068/0.069	0.049/0.024	0.000/0.000	0.005/0.005	0.114/0.064
R, C, NO, COFFEE	0.071/0.053	0.000/0.009	0.037/0.019	0.035/0.014	0.007/0.003	0.003/0.002	0.036/0.015
R, NC, NO, SOP	0.000/0.036	0.053/0.025	0.051/0.032	0.047/0.022	0.010/0.003	0.035/0.019	0.106/0.039
R, NC, NO, WSOP	0.000/0.000	0.068/0.030	0.028/0.033	0.046/0.020	0.000/0.000	0.012/0.007	0.098/0.038
R, NC, NO, COFFEE	0.072/0.044	0.005/0.003	0.064/0.029	0.049/0.019	0.000/0.000	0.005/0.004	0.047/0.016
S, C, O, SOP	0.000/0.003	0.010/0.006	0.070/0.041	0.040/0.018	0.009/0.003	0.000/0.001	0.036/0.018
S, C, O, WSOP	0.000/0.030	0.010/0.011	0.074/0.040	0.037/0.017	0.000/0.000	0.000/0.004	0.054/0.026
S, C, O, COFFEE	0.066/0.051	0.000/0.000	0.035/0.016	0.037/0.015	0.000/0.000	0.002/0.001	0.048/0.017
S, NC, O, SOP	0.000/0.037	0.009/0.011	0.083/0.038	0.037/0.017	0.010/0.003	0.000/0.004	0.059/0.029
S, NC, O, WSOP	0.000/0.028	0.008/0.007	0.083/0.041	0.046/0.021	0.000/0.000	0.000/0.003	0.061/0.030
S, NC, O, COFFEE	0.064/0.060	0.000/0.000	0.033/0.016	0.026/0.010	0.009/0.004	0.000/0.000	0.045/0.016
S, C, NO, SOP	0.000/0.003	0.009/0.011	0.067/0.035	0.034/0.016	0.009/0.003	0.000/0.001	0.026/0.013
S, C, NO, WSOP	0.000/0.017	0.009/0.005	0.069/0.035	0.038/0.018	0.000/0.000	0.000/0.003	0.046/0.022
S, C, NO, COFFEE	0.073/0.057	0.000/0.000	0.039/0.014	0.042/0.017	0.013/0.005	0.000/0.000	0.039/0.014
S, NC, NO, SOP	0.000/0.004	0.008/0.013	0.087/0.049	0.038/0.018	0.010/0.003	0.000/0.003	0.045/0.024
S, NC, NO, WSOP	0.000/0.000	0.008/0.013	0.072/0.033	0.040/0.018	0.000/0.000	0.000/0.003	0.057/0.028
S, NC, NO, COFFEE	0.037/0.036	0.000/0.000	0.030/0.015	0.043/0.017	0.016/0.006	0.002/0.001	0.046/0.016

Table D.4.: Standard deviation values for the mean values in table 5.2. The values that the means are based on were shown to be deviate from the normal distribution. No confidence intervals were calculated. Instead the Wilcoxon Signed Rank test was used to verify whether differences were significant (see appendix A).

E. Supplementary Material for the Numerical Optimisation Study

	griewank			rastrigin			rosenbrock		
	20-D	50-D	100-D	20-D	50-D	100-D	20-D	50-D	100-D
Davis_S	0.023	0.012	0.005	0.000	0.000	0.001	10.928	29.498	54.578
Davis	0.019	0.007	0.005	0.000	0.000	0.000	15.479	21.482	36.821
SIDEA_S	0.035	0.008	0.007	0.000	0.000	0.000	10.623	25.228	49.880
SIDEA	0.023	0.006	0.006	0.000	0.000	0.000	0.752	29.325	58.739
SPDEA_S	0.023	0.011	0.006	0.000	0.001	0.001	1.684	22.003	43.865
SPDEA	0.021	0.007	0.006	0.000	0.000	0.000	11.931	34.076	43.004
SIESA	0.022	0.010	0.007	0.000	0.000	0.000	20.289	1.377	34.600
Subpop	0.026	0.009	0.006	0.000	0.000	0.000	21.468	36.737	52.072
Uniform	0.021	0.009	0.004	0.000	0.000	0.001	12.413	15.366	33.993
ADOPP_S	0.024	0.011	0.006	0.000	0.001	0.317	26.517	29.850	48.474
ADOPP	0.029	0.007	0.009	0.000	0.000	3.222	0.574	7.108	131.017
DE_only	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Table E.1.: Standard deviation values for the mean values in table 6.1. The values that the mean values are based on were shown to be approximately normally distributed.

	griewank			rastrigin			rosenbrock		
	20-D	50-D	100-D	20-D	50-D	100-D	20-D	50-D	100-D
Davis_S	0.023	0.011	0.004	0.000	0.000	0.000	9.224	32.869	54.938
Davis	0.028	0.008	0.005	0.000	0.000	0.000	13.934	19.912	46.594
SIDEA_S	0.022	0.010	0.005	0.000	0.000	0.000	7.604	26.957	46.471
SIDEA	0.023	0.006	0.005	0.000	0.000	0.000	0.603	23.488	55.376
SPDEA_S	0.031	0.008	0.005	0.000	0.001	0.001	1.639	14.495	46.498
SPDEA	0.023	0.009	0.007	0.000	0.000	0.000	0.676	31.781	48.629
SIESA	0.031	0.009	0.006	0.000	0.000	0.000	14.775	14.620	27.944
Subpop	0.008	0.006	0.007	0.000	0.384	0.141	0.000	1.229	28.637
Uniform	0.027	0.007	0.006	0.000	0.000	0.141	23.545	8.645	35.633
ADOPP_S	0.036	0.011	0.007	0.000	0.001	0.407	19.567	27.057	53.595
ADOPP	0.044	0.011	0.008	0.000	0.000	2.823	8.104	7.711	45.040

Table E.2.: Standard deviation values for the mean values in table 6.2. The values that the mean values are based on were shown to be approximately normally distributed.

F. Publications

During the studies for this thesis three papers were written. Most of the contents of the papers has been been incorporated in different chapters of this thesis. For completeness, however, I have included them in this appendix.

Using adaptive operator scheduling on problem domains with an operator manifold: Applications to the Travelling Salesman Problem

This paper represents an initial investigation of the Traveling Salesman Problem using only the Davis operator scheduling algorithm. It was written in the summer of 2003 and can be found in the proceedings of the 2003 Congress on Evolutionary Computation (CEC 2003) [5].

An Investigation of Adaptive Operator Scheduling Methods on the Traveling Salesman Problem

This paper is a natural sequel to the first. A number of adaptive operator scheduling schemes were implemented and tried out on various instances of the Traveling Salesman Problem. The paper was submitted to the 4th European Conference on Evolutionary Computation in Combinatorial Optimization. I have not yet received notification of its acceptance. If accepted, it will be published by Springer as part of their Lecture Notes in Computer Science series.

Multiple Sequence Alignment Using SAGA: Investigating the Effects of Operator Scheduling, Population Seeding, and Crossover Operators

In this paper the SAGA program is investigated. An initial study had suggested that operator scheduling has little effect on the Multiple Sequence Alignment problem. The authors of SAGA however claimed the technique to be successful in their program. An investigation was done to verify their claim. This paper was written together with René Thomsen and submitted to the 2nd European Workshop on Evolutionary Bioinformatics. No notification of its acceptance has as yet been received. If accepted, it will be published by Springer as part of their Lecture Notes in Computer Science series.

Using adaptive operator scheduling on problem domains with an operator manifold: Applications to the Travelling Salesman Problem

Wouter Boomsma
EVALife group, Dept. of Computer Science, University of Aarhus
Ny Munkegade, Bldg. 540, DK-8000 Aarhus C, Denmark
wb@daimi.au.dk

Abstract- A growing problem in the field of evolutionary computation is the large amount of genetic operators available for certain problem domains. This tendency is especially pronounced in areas where heuristics are used to create highly specialised operators. Even within the same problem domain, the performance of such operators often depends on the specific problem instance at hand. This results in a tedious and time-consuming process of comparing individual operator performances every time a new problem is to be solved.

This paper investigates the use of adaptive operator scheduling to automate the operator selection process. The approach is tested on instances of the Travelling Salesman Problem - a problem for which a long list of operators exists. Results show that benefits are twofold: Operator selection is achieved automatically and an overall performance improvement is observed.

1 Introduction

It has become generally accepted that genetic algorithms benefit from the incorporation of domain specific knowledge. Many real-world problems are solved by designing a problem specific representation and corresponding operators that manipulate individuals with methods inspired by the context of the problem. The possibilities for creating more or less sophisticated heuristic operators are endless. Papers presenting new operators for specific problem domains appear frequently.

When confronted with a new problem to solve, the evolutionary programmer is thus forced to take a pick in a large pool of operators. Even though the literature will contain comparisons between certain selections of operators, the uncertainty remains: Which combination of operators performs best for the specific problem at hand? A good overview requires the programmer to invest time and effort in comparing all operators individually.

In this paper I investigate whether adaptive operator scheduling is a possible solution to this problem. The idea is that the whole set of candidate operators is implemented, and that the optimal choice of operators is found automatically by the algorithm.

Previous work is not conclusive on the effects that adaptive operator scheduling has on performance. Davis presents good results on neural networks in his original paper from 1989[1]. Srinivas & Patnaik tested an approach on both numerical benchmark problems and smaller instances of the TSP with good results on most problems[4]. Julstrom does well on the same problems with a different approach[5]. Thomsen & Krink achieve encouraging results on a set of numerical benchmark functions[2]. However, Eiben, Sprinkhuizen-Kuyper & Thijssen report that performance was not improved in their competing crossover approach[3], and Tuson & Ross only achieved better performance on certain problems[6]. In the light of these results, significant performance improvements over non-adaptive algorithms are not anticipated. The adaptive approach might introduce some improvements as an added bonus, but solving the tedious operator selection problem is the main objective.

The algorithm is tested on the travelling salesman problem - a well known NP-hard combinatorial problem for which many operators have been designed. TSP consists of determining the shortest Hamiltonian cycle in a complete graph, i.e. visiting all nodes and minimising the summed weight of the edges passed on the way.

The paper is organised as follows: Section 2 presents a selection of operators that have been developed for the TSP during the last 30 years. In section 3 the adaptive operator scheduling genetic algorithm (AOSGA) is described followed by a description of the conducted experiments in section 4. Section 5 focuses on a performance comparison between the AOSGA and the genetic algorithms with a fixed combination of operators (fixed-op GAs). I show that that the adaptive approach gives robust results equal or slightly better than any combinations of fixed operators. A discussion of the results and some concluding remarks are presented in section 6.

2 The Operators

Table 1 lists the 6 mutation operators and 10 crossover operators used in the experiments. They all operate on a path representation of the TSP.

The selection of operators was inspired by Larrañaga's survey paper from 1999[21]. In recent years other operators have been proposed (e.g. [22, 23, 24]), and some of these are known to outperform the ones listed in table 1. Time has not yet permitted me to implement these more advanced operators. This is however not a serious limitation.

Name	Authors
Partially mapped Crossover (PMX)	Goldberg and Lingle [7]
Cycle Crossover (CX)	Oliver, Smith and Holland [8]
Order Crossover (OX1)	Davis [9]
Order Based Crossover (OX2)	Syswerda [10]
Position Based Crossover (POS)	Syswerda [10]
Heuristic Crossover (HEU)	Grefenstette [11]
Edge Recombination Crossover (ER)	Whitley, Timothy and Fuquay [12]
Maximal Preservative Crossover (MPX)	Mühlenbein, Gorges-Schleuter and Krämer [13]
Voting Recombination Crossover (VR)	Mühlenbein [14]
Alternating Position Crossover (AP)	Larrañaga, Kuijpers, Poza, and Murga [15]
Displacement Mutation (DM)	Michalewicz [16]
Exchange Mutation (EM)	Banzhaf [17]
Insertion Mutation (ISM)	Fogel [18]
Simple Inversion Mutation (SIM)	Holland [19]
Inversion Mutation (IVM)	Fogel [20]
Scramble Mutation (SM)	Syswerda [10]

Table 1: The operators

Since the goal of this paper is to analyse costs and benefits of an adaptive approach compared to an approach with fixed operators, we are interested in relative performance between the approaches rather than the absolute performance of single operators.

3 The AOSGA model

The AOSGA (figure 1) is based on an algorithm presented by Lawrence Davis in 1989[1]. His paper represents one of the first efforts to incorporate adaptive operator scheduling in a genetic algorithm. The idea is as follows: All individuals in the population keep track of the operator used to create them and which other individuals (parents) were involved. New individuals are rewarded if their fitness exceeds the current best fitness in the population. Furthermore, the individual's ancestors are recursively rewarded (to depth `M`) with some percentage `P` of their child's reward. Once every `I`'th generation, every operator is rewarded by the sum of the rewards of the individuals that it has produced. Based on these operator rewards, the new probability settings are computed. The parameter `W` determines the window of adaptation (how many of the recently created individuals should be considered) and the parameter `S` determines how much the probability is shifted. The probability update equation for the ith operator in the operator pool is:

$$\mathrm{p}'[i] = (1 - \mathrm{S}) * \mathrm{p}[i] + \mathrm{S} * \frac{\mathrm{reward}[i]}{\mathrm{totalReward}} \qquad (1)$$

A few modifications were made to Davis' original description. First, the steady state approach was generalised to a generational GA with a variable elite size (Setting the elite size to one below the population size gives us Davis' steady state approach). A generation consists of recombination, mutation and selection, and for each of these a group of operators is defined. It is within these groups that Davis' probability adaptation scheme is applied (the probabilities in each group thus sum to 1.0).

AOSGA
  initialize($population$)
  while (!done) {
    recombine($population$)
    mutate($population$)
    select($population$)
    if($generations$ mod $I == 0$)
      adapt()
  }

adapt()
  PassOnRewardsToParents()
  for each of the W individuals created most recently **do**
    assignOperatorRewards()
  updateOperatorProbabilities()

Figure 1: Pseudo-code for the AOSGA

A lower bound (2%) on the operator probabilities was introduced to avoid that operators go extinct during the run. This is done to ensure that all operators have a chance of revival after a period of dominance of a rival operator. To further encourage this recovery, the operator probability setting slowly expires: If the AOSGA completes a generation

TSP	Optimum	Evaluations used	Best fi xed-op GA			AOSGA	
			Operators	Best	Average	Best	Average
gr48	5,046	50,000	IVM+HEU	**5,046**	5,184.8(2.6)	**5,046**	5,166.0(2.4)
brg180	1,950	2 mil.	IVM+OX1	1,980	2,036.2(3.2)	1,970	2,067.0(4.6)
pcb442	50,778	3 mil.	SIM+OX2	54,847	57,278(91)	53,532	55,442.0(81)

Table 2: Best fi xed-op GA vs. the AOSGA

without improving the best solution, it adjusts the operator probabilities slightly towards the original values. The probability update function is in this case:

$$\mathrm{p}'[i] = (1 - 0.005) * \mathrm{p}[i] + 0.005 * \frac{1}{\mathrm{NrOfOps}} \qquad (2)$$

where NrOfOps is the number of operators in the pool.

4 Experiments

This study was motivated by a wish to avoid the tedious operator selection process when faced with a new problem. Two criteria must be fulfi lled for this approach to be successful:

1. Quality: The results of the AOSGA should be as good as the best of the fi xed-op versions.

2. Speed: Convergence speed should not be signifi cantly reduced.

The quality issue is investigated by comparing the results of the AOSGA with all fi xed-op combinations of a single mutation and crossover operator. Speed is tested by plotting the current best solution as a function of used evaluations during the course of the solution process (averaged over some amount of runs).

Experiments were conducted on 3 symmetrical TSP instances: A 48-city problem, a 180-city problem and a 442-city problem (gr48, brg180 and pcb442), all of which are taken from the TSP benchmark problem collection TSPLIB[25].

Some initial experimentation was done to determine the best parameter settings for the algorithm. A crossover rate of 0.6 and an overall mutation rate of 1.0 were used. This is only reasonable if the elite size is kept relatively high - 80 out of a population size of 100 proved to be a good compromise between a classical generational GA and the steady state model used by Davis. Some of the TSP operators have additional parameters which were fi ne-tuned by hand based on single-operator runs of the algorithm. Details on these parameter settings can be found in the original descriptions of the operators (see table 1). The settings used were:

OX2	Maximum percentage of genes to move: 0.95
POS	Maximum percentage of genes to fi x: 0.05
VR	Maximum percentage of population that can be chosen as parents: 0.25
	Vote threshold: 0.05
SM	Maximum size of random sub-tour (in percentage of genome size): 0.5

All operator probabilities were initially set to $1/$NrOfOps. The parameters involved in controlling the adaptation scheme were set as follows:

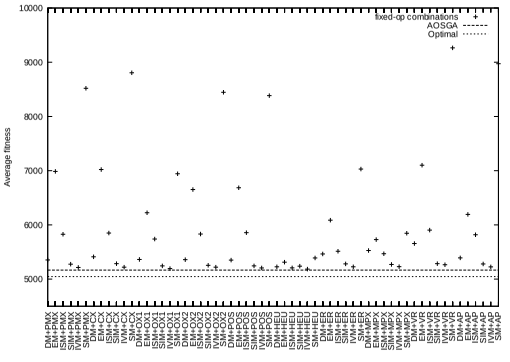
P	S	W	M	I
90%	15%	100	10	1

which were inspired by the values that Davis proposed. The robustness of the algorithm to variations in these setting is however rather high, and other values gave equally good results.

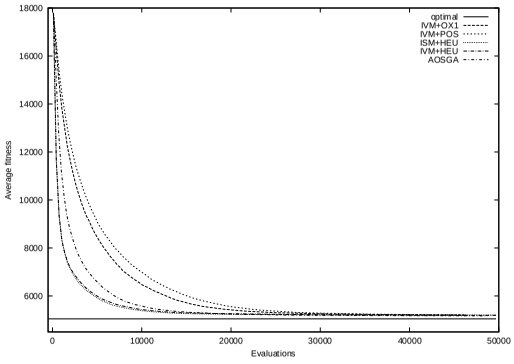
5 Results

All pairwise combinations of mutation and crossover operators were applied to the 3 TSP problems. For each of these combinations, a simple tournament selection scheme was used. In table 2 the combination with the best average performance is compared with the results of the AOSGA using all operators. The results on gr48 are based on 1000 repetitions, while only 100 repetitions were conducted for the two other problems. Mean values are given with the standard error in parentheses. Execution time ranged from 3 seconds for the gr48 problem to 6-7 minutes for the pcb442 problem (Processor: Xeon 1.7 GHz). A graphical overview of the comparisons is presented in fi gure 2.

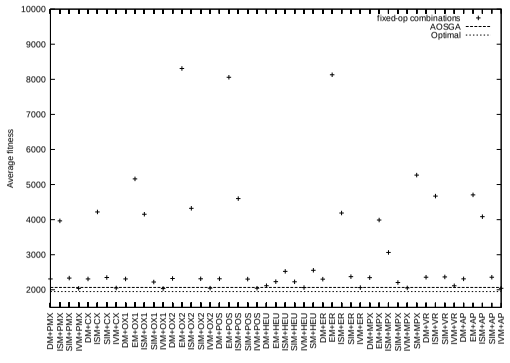
To get an impression of the convergence speed of the AOSGA, progress during the course of a run was logged and compared to the progress of 4 of the best performing fi xed-op algorithms. Results are found in fi gure 2(d-f). The graphs indicate that convergence speed of the AOSGA is not worse than that of the average fi xed-op algorithm. For the larger problems, it would even seem that the AOSGA reaches acceptable solutions somewhat faster than the fi xed-op algorithms.



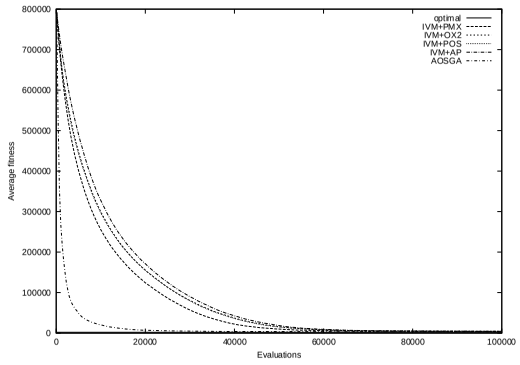
(a) gr48



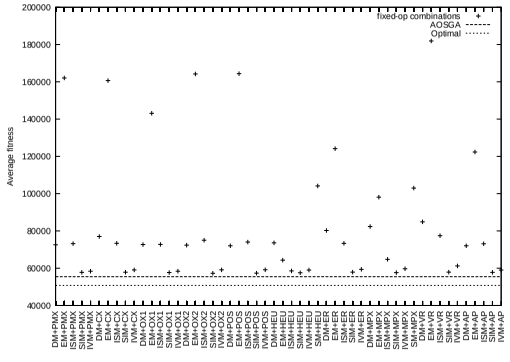
(d) gr48



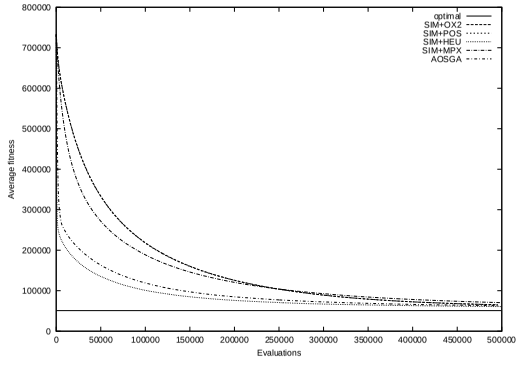
(b) brg180



(e) brg180



(c) pcb442



(f) pcb442

Figure 2: Fixed-op combinations vs. the AOSGA. (a-c) show the best fitness averaged over all runs (algorithms with a fitness higher than 10000 are not shown), (d-e) show the average progress during the course of a run.

6 Discussion

The results demonstrate that the two performance requirements are fulfilled. The AOSGA can competite with any combination of single operators without a significant decrease in efficiency. The algorithm successfully manages to identify valuable operators during the course of a run and thereby eliminates the need to carry out such experiments by hand.

Concerning the quality of the solutions that the algorithm produces, table 2 and figure 2 indicate that the AOSGA obtains results comparable to the best fixed-op GA. For the two larger problems, the AOSGA converges to these results significantly faster than it's fixed-op counterparts.

As illustrated in figure 3 and 4, operator scheduling definitely takes place, and different operators alternately dominate during a run. Since parameter settings for the individual operators were tuned for single-operator runs, all operators were optimised for all-round best performance. This might not be the optimal setting when the operators are controlled by the AOSGA, which can schedule operators that are specialised for certain tasks (e.g. exploration/exploitation). Introducing task division by specialising operators (e.g. coarse/fine-tuning) the potential of operator scheduling could be exploited to a higher degree.

Futher experimentation should also be conducted concerning the mechanism of adaptation itself. Incorporation of operator probabilities as part of the genotype combined with self-adaptation would require less bookkeeping than the adaptive algorithm presented here. If the performance of this alternative is comparable to the AOSGA, it would therefore be preferable the somewhat cumbersome AOSGA.

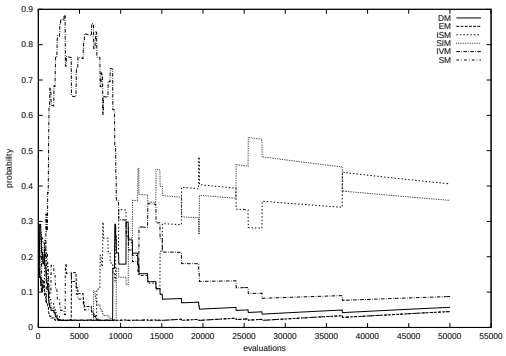


Figure 3: Probability distribution for the mutation operators (gr48)

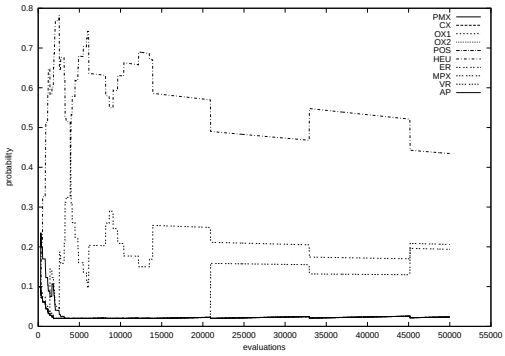


Figure 4: Probability distribution for the crossover operators (gr48)

Acknowledgements

I would like to thank everyone at EVALife for their support and valuable input during this study and the writing of this paper.

Bibliography

[1] L. Davis, "Adapting operator probabilities in genetic algorithms," in *Proceedings of the Third International Conference on Genetic Algorithms* (J. D. Schaffer, ed.), (San Mateo, CA), Morgan Kaufman, 1989.

[2] R. Thomsen and T. Krink, "Self-adaptive operator scheduling using the religion-based ea," in *Proceedings of Parallel Problem Solving from Nature VII (PPSN-2002)*, pp. 214–223, Springer Verlag, 2002.

[3] I. G. S.-K. Agoston E. Eiben and B. A. Thijssen, "Competing crossovers in an adaptive ga framework," in *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, pp. 787–792, 1998.

[4] M. Srinivas and L. M. Patnaik, "Adaptive probabilities of crossover and mutation in genetic algorithms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 24, no. 4, pp. 656–667, 1994.

[5] B. A. Julstrom, "What have you done for me lately? Adapting operator probabilities in a steady-state genetic algorithm," in *Proceedings of the Sixth International Conference on Genetic Algorithms* (L. Eshelman, ed.), (San Francisco, CA), pp. 81–87, Morgan Kaufmann, 1995.

[6] A. Tuson and P. Ross, "Cost based operator rate adaption: An investigation," in *Parallel Problem Solving*

from Nature – PPSN IV (H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, eds.), (Berlin), pp. 461–469, Springer, 1996.

[7] D. E. Goldberg and R. Lingle Jr., "Alleles, loci, and the traveling salesman problem," in *Proceedings of the First International Conference on Genetic Algorithms and Their Applications* (J. J. Grefenstette, ed.), Lawrence Erlbaum Associates, Publishers, 1985.

[8] I. M. Oliver, D. J. Smith, and J. R. C. Holland, "A study of permutation crossover operators on the travelling salesman problem," in *Genetic algorithms and their applications : Proc. of the second Int. Conf. on Genetic Algorithms* (J. J. Grefenstette, ed.), (Hillsdale, NJ), pp. 224–230, Lawrence Erlbaum Assoc., 1987.

[9] L. Davis, "Applying adaptive algorithms to epistatic domains," in *Proceedings of the International Joint Conference on Artificial Intelligence*, vol. 1, pp. 161–163, 1985.

[10] G. Syswerda, *Schedule optimization using genetic algorithms*, ch. 21, pp. 332–349. 1991.

[11] J. J. Grefenstette, *Incorporating problem specific knowledge into genetic algorithms*, pp. 42–60. 1987.

[12] D. Whitley, T. Starkweather, and D. Fuquay, "Scheduling problems and traveling salesman: The genetic edge recombination operator," in *Proceedings of the Third International Conference on Genetic Algorithms* (J. D. Schaffer, ed.), (San Mateo, CA), Morgan Kaufman, 1989.

[13] H. M¨uhlenbein, M. Gorges-Schleuter, and O. Kr¨amer, "Evolution algorithms in combinatorial optimization," *Parallel Computing*, vol. 7, pp. 65–85, 1988.

[14] H. M¨uhlenbein, "Parallel genetic algorithms, population genetics and combinatorial optimization," in *Proceedings of the Third International Conference on Genetic Algorithms* (J. D. Schaffer, ed.), (San Mateo, CA), Morgan Kaufman, 1989.

[15] M. P. P. Larrañaga, C. M. H Kuijpers and R. H. Murga, "Decomposing bayesian networks: triangulation of the moral graph with genetic algorithms," *Statistics and Computing (UK)*, vol. 7, no. 1, pp. 19–34, 1997.

[16] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*. Berlin: Springer, 1992.

[17] W. Banzhaf, "The "molecular" traveling salesman," *Biological Cybernetics*, vol. 64, pp. 7–14, 1990.

[18] D. B. Fogel, "An evolutionary approach to the travelling salesman problem," *Biological Cybernetics*, vol. 60, no. 2, pp. 139–144, 1988.

[19] J. H. Holland, *Adpatation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press, 1975.

[20] D. Fogel, "A parallel processing approach to a multiple travelling salesman problem using evolutionary programming," in *Proceedings of the Fourth annual Symposium on Parallel Processing*, (Fullerton, California), pp. 318–326, April 1990.

[21] R. H. M. I. I. P. Larrañaga, C. M. H. Kuijpers and S. Dizdarevic, "Genetic algorithms for the travelling salesman problem: A review of representations and operators," *Artificial Intelligence Review*, vol. 13, no. 2, pp. 129–170, 1999.

[22] Y. Nagata and S. Kobayashi, "Edge assembly crossover: A high-power genetic algorithm for the travelling salesman problem," in *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA97)* (T. B¨ack, ed.), (San Francisco, CA), Morgan Kaufmann, 1997.

[23] G. Tao and Z. Michalewicz, "Inver-over operator for the TSP," in *Parallel Problem Solving from Nature – PPSN V* (A. E. Eiben, T. B¨ack, M. Schoenauer, and H.-P. Schwefel, eds.), (Berlin), pp. 803–812, Springer, 1998. Lecture Notes in Computer Science 1498.

[24] H.-K. Tsai, J.-M. Yang, and C.-Y. Kao, "Solving traveling salesman problems by combining global and local search mechanisms," in *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002* (D. B. Fogel, M. A. El-Sharkawi, X. Yao, G. Greenwood, H. Iba, P. Marrow, and M. Shackleton, eds.), pp. 1290–1295, IEEE Press, 2002.

[25] G. Reinelt, "TSPLIB — a traveling salesman problem library," *ORSA Journal on Computing*, vol. 3, no. 4, pp. 376–384, 1991.

An Investigation of Adaptive Operator Scheduling Methods on the Traveling Salesman Problem

Wouter Boomsma

Department of Computer Science,
University of Aarhus, Denmark
wb@daimi.au.dk

Abstract. The implementation of an evolutionary algorithm necessarily involves the selection of an appropriate set of genetic operators. For many real-world problem domains, an increasing number of such operators is available. The usefulness of these operators varies for different problem instances and often changes during the course of the evolutionary process. This motivates the use of adaptive operator scheduling (AOS) to automate the selection of efficient operators. However, little research has been done on the question of which scheduling method to use. This paper compares different operator scheduling methods on the Traveling Salesman Problem. Among these is the well known method by Davis and a more recent method by Julstrom. Furthermore, several new AOS techniques are introduced and comparisons are made to two non-adaptive alternatives.
The results show that most of the introduced algorithms perform as well as Davis' algorithm while being significantly less cumbersome to implement. Overall, the use of AOS is shown to give significant performance improvements - both in quality of result and convergence time.

1 Introduction

Genetic operators are the algorithmic core of evolutionary computation. The quality of these operators is essential for the performance of the evolutionary algorithms (EAs), and consequently, much research is done in this field. Especially for combinatorial problem domains, the incorporation of problem specific knowledge is often essential in order to achieve satisfactory performance. Endless possibilities exist for the design of such operators, and collections of problem-specific operators grow rapidly.

Given their heuristic nature, the performance of these operators often varies across different problems in the domain. For instance, certain operators might fail to scale up gracefully, becoming computationally infeasible for larger problem instances. Even though the literature might provide comparisons between operators on certain problem instances, the evolutionary programmer is left with a difficult choice of operator selection when designing an EA for a new problem.

An elaborate manual comparison of all operators would provide an assessment of the quality of the operators, but does not exploit the fact that the

usefulness of the operators often changes during the course of a single run. Furthermore, it does not take into account that dependencies between operators can exist and that through interaction multiple operators might provide better results than when applied alone.

In a previous study[1], I investigated whether adaptive operator scheduling (AOS) can provide a solution to this problem. Experiments were done on instances of the symmetrical Traveling Salesman Problem (TSP), a well known NP-hard combinatorial problem for which a multitude of operators exist. It can be defined as the search for a minimal Hamiltonian cycle in a complete graph, and can be understood as the problem of visiting n cities (and returning to the first), using the path of smallest total distance.

The main concern in the original investigation was that a large amount of operators might slow down the optimisation process compared to an algorithm using the optimal choice of operators. It was shown that this concern was unfounded: the algorithm using AOS converged as fast as the best combination of mutation and crossover operators with equally good results. The AOS scheme used in these initial experiments (Davis) was however rather cumbersome to implement, which is unfortunate when AOS is motivated as a better alternative to the elaborate manual comparison of operators.

In this paper several alternative methods of AOS are presented and compared on a selection of TSP benchmark problems. It is shown that the presence of multiple operators significantly improves performance. Furthermore, results indicate that the alternative AOS methods perform as well as Davis' method, and that simpler methods can thus be used without a decrease in performance.

2 Adaptive Operator Scheduling

In his survey paper on adaptation in EAs [2], Angeline introduces a categorisation of adaptive evolutionary computation based on two criteria: (1) the level at which adaptation is applied and (2) the nature of the update rules. The level of adaptation specifies at which level the adapted parameters reside. For *population-level* adaptation, the parameters exist at the population level, and thus apply to all individuals in the population. Likewise, *individual-level* adaptation works on parameters local to each individual and *component-level* adaptation has parameters for each component in the genotype of an individual. Update rules are classified as being *absolute*, or *empirical*. This last class also goes by the name of self-adaptation, which is the term that will be used throughout this paper. Algorithms with absolute update-rules use fixed guidelines to update parameter settings based on the current state of the population. Self-adaptive schemes use the selection pressure already present in the evolutionary process to evolve better parameter settings, the underlying assumption being that good individuals often have a good parameter setting.

The following AOS schemes were selected to experiment with some of the classes in Angeline's classification. Focus was on designing AOS algorithms that are easily implemented, which is vital if it is to replace the manual comparison of

individual operators. The end of each description contains a list of the parameter settings that were used for the algorithm. These settings were manually tuned based on preliminary experiments.

2.1 The Operator Scheduling Algorithm by Davis

Davis' algorithm[3] from 1989 represents one of the first efforts at operator adaptation in EAs. It uses a *population-level* adaptation using absolute update rules. More specifically, a global set of operator probabilities is adapted based on the performance of the operators in the last generations (window of adaptation W). The performance of operators is measured by the quality of the individuals they produce. Newly created individuals are rewarded if their fitness surpasses the fitness of all other individuals in the population. The size of this reward is determined by the amount of the improvement. Furthermore, a certain percentage of the reward is recursively passed on to the individual's ancestors (to a certain maximum depth M). This reward strategy is motivated by the fact that a series of suboptimal solutions is often necessary in order to reach a better solution. Corresponding operators should therefore be rewarded.

With certain intervals(I), the rewards are used to update the probability setting. For each operator i:

$$\mathrm{p}_i' = (1 - \mathrm{S}) * \mathrm{p}_i + \mathrm{S} * \frac{\mathrm{reward}_i}{\mathrm{totalReward}} \tag{1}$$

where p_i' is the new probability for operator i and the parameter S is the Shift factor, determining the influence that the current update should have on the total probability setting.

In this paper a slightly modified version of Davis' algorithm is used. Lower bounds are set on the probabilities to avoid extinction of operators and in adaption phases where no improvement is made the probabilities are shifted slightly toward their initial positions. Furthermore, the steady state evolutionary approach is replaced by a generational elitist EA. Preliminary experiments showed that these modifications gave better results and faster convergence (results not shown).

Parameter settings: Window of adaptation(W): 100 individuals, Interval of adaptation(I): 20-50 evaluations, Shift percentage(S): 15%, Percentage of reward to pass back(P): 90%, Number of generations to pass back(M): 10.

2.2 The ADOPP Algorithm

Julstrom's ADaptive OPerator Probabilities algorithm (ADOPP)[4] from 1995 is very similar to Davis' approach. The main difference between the two is the way in which ancestral information is represented. While Davis provides each individual with links to their parents, Julstrom explicitly provides each individual with a tree specifying which operators were used to create its ancestors. Davis' approach is more effective, since the tree structure is implicitly present in the

individuals and does not have to be copied every time new individuals are created. However, Davis' algorithm has some disadvantages which Julstrom avoids: Since the ancestral information is stored in the individuals, information is lost when individuals die. The depth of the implicit trees are therefore somewhat unreliable, and in general, Davis' algorithm is only able to maintain a moderate amount of ancestral information. Furthermore, it requires a great deal of book-keeping for the individuals in Davis' algorithm to sustain links only to live individuals, an inconvenience which is avoided in an implementation of ADOPP. Another difference between the two approaches is that Davis rewards individuals that improve the overall best individual in the populations, while Julstrom only requires individuals to exceed the median.

When converting operator rewards to a new probability setting, ADOPP uses a greedy variant of the update rule by Davis (corresponding to S=1.00). For each operator i:

$$p'_i = \frac{\text{reward}_i}{\text{totalReward}} \tag{2}$$

where p'_i is the new probability for operator i.

Parameter settings: Window of adaptation(QLEN): 100 individuals, Percentage of delta to pass back(DECAY): 80%, Interval of adaptation(I): 1 generation, Height of trees(DEPTH): 4.

2.3 Adaptation Using Subpopulation

This approach was inspired by the work by Schlierkamp-Voosen and Mühlenbein [5] on competing subpopulations, in which different subpopulations represent different settings of the same operator. This scheme is transformed into an AOS method by letting each population represent the use of a single operator. The algorithm thus maintains a number of subpopulations equal to the number of operators. For each of these populations, only one designated operator can be applied to the individuals currently residing there. During the course of a run, the relative sizes of the subpopulations are altered depending on their fitness. The fitness of a subpopulation is defined as the fitness of its best individual.

In each adaptation phase the best subpopulation is rewarded with an increase of size, and receives a donation of individuals from all other subpopulations. To avoid the extinction of operators, only subpopulations with a size above some fixed lower bound are forced to make this donation.

Large subpopulations create more offspring and thus have a larger probability of improving the global best fitness. This results in a strong bias towards larger subpopulations, making it difficult for smaller populations to compete. As an attempt to counter this effect, a random migration scheme is used: At certain intervals some of the best individuals from the best population migrate to a randomly selected population. This algorithm will be referred to as the Subpop algorithm. Just as the previous algorithms, also Subpop is a *population-level* adaptation algorithm using absolute update rules. The parameters adapted are in effect the same as for Davis and ADOPP (global operator probabilities).

Parameter settings: Evaluation interval(E): 4 generations, Shift amount(SA): 10%, Migration interval(M): 4 generations, Migration amount(MA): 5 individuals.

2.4 Self Adaptive Operator Scheduling

The algorithms in this section are novel methods of operator adaptation partly inspired by the work of Spears in 1995[6]. In Spears' paper, scheduling is done between two crossover operators by adding a single bit to the genotype which indicates the preferred operator. These bits are used either locally (at *individual-level*) or globally (at *population-level*) to determine which operator is to be applied. When used locally, the choice of operator is made based on the bit-setting of the parent(s) involved. When used globally, the average bit-setting of the whole population is used as a measure of quality to base this decision on.

Spears' method is generalised by expanding the single bit to an array of n probabilities. Unlike his approach, the representation of our scheduling information is not compatible with that of the problem representation and therefore cannot be modified automatically as part of the genotype. It was therefore necessary to design a specialised variation operator for this task.

The problem of finding the optimal probability setting for n operators is a numerical optimisation problem and in principle, any variation operator from this domain would suffice. However, given the fact that the probabilities must all be positive and sum to one, only a small subset of the n-dimensional search space constitutes legal solutions, which would require the use of some repair scheme after each operation to make the solutions feasible.

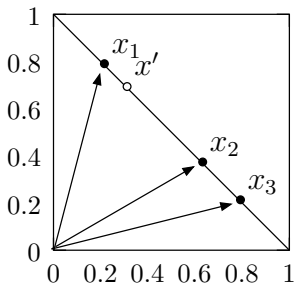


Fig. 1. DE mutation operator

To avoid this I used an adapted version of the mutation operator used in Differential Evolution(DE)[7], which produces solutions that are only rarely illegal. The DE variation operator uses 3 individuals to create an offspring by applying a mutation step followed by a crossover step. During the mutation step, a new genotype $\boldsymbol{x}'$ is created by:

$$\boldsymbol{x}' = \boldsymbol{x}_1 + F * (\boldsymbol{x}_3 - \boldsymbol{x}_2) \tag{3}$$

The crossover step subsequently creates a new individual by combining components from $\boldsymbol{x}'$ with components from the original individual. For our purpose, however, the mutation step alone has exactly the properties we need: Selecting three random points on a hyperplane and adding the vectorial difference between two of them to the third results in a new position on the plane. In other words, given three legal probability settings it will produce a new one with probabilities that sum to 1.0. An example of this behaviour for 2 dimensions can be seen in Fig. 1. Equation (3) of course gives no guarantees that the entries of the created

vector have values between zero and one. This is however easily fixed by forcing values that lie outside the domain back to the nearest boundary.

The mutation operator is used whenever new offspring is to be created. Each time, three random parents are selected, and the new probability setting for the child is calculated using (3).

As with Spears' algorithm, two different levels of adaptation exist. If *individual-level* adaptation is used, the probability setting of the child is used to choose an operator. If adaptation works at the *population-level*, an average of all probability-settings in the population is used to select an operator. The two versions of the algorithm will be referred to as the SIDEA and the SPDEA respectively.

Parameter setting: F-value for the DE mutation: 0.5.

2.5 Operator Scheduling Inspired by Evolution Strategies

Self-adaptation has been applied in the Evolution Strategies (ES) community since 1977[8]. Here it was used to adapt the behaviour of the mutation operator during the course of a run. The mutation operator consists of an addition of normally distributed values $z_i \sim \mathbf{N}(0, \sigma_i'^2)$ to the components in the genotype. The variances $\sigma_i'^2$ are adapted so that the effect of mutation is different for different individuals and for different components of the genotype. Again, under the assumption that good individuals often have good parameter settings, the algorithm can find the optimal variance setting using only the selection pressure already present in the algorithm.

Taking self-adaptation one step further, we now use the ES mutation step as an alternative to the DE mutation operator described in the previous section. This means that the evolutionary algorithm will simultaneously optimise the problem at hand, an operator probability setting, and a setting determining the optimal variance for the mutation of this probability setting.

Since the search space defined by the operator probability settings is highly interdependent, it is meaningless to adapt each variance σ_i^2 at component-level. Instead one value σ is associated with each individual. Equation (4) shows how these values are mutated[9].

$$\sigma' = \sigma * \exp(s_0) \tag{4}$$

where $s_0 \sim \mathbf{N}\left(0, \tau_0^2\right)$ and $\tau_0^2 = \frac{1}{n}$. Since this mutation operator does not have the convenient properties of the DE operator, the probability-setting has to be normalised after each mutation. It is difficult to tell exactly how big the impact of this normalisation is compared to the effect of the mutation itself. This operator will be referred to as SIESA.

Parameter setting: τ_0^2: $\frac{1}{n}$ (ES default value [9]).

3 Operators

Table 1 lists the 18 operators used in the experiments. They all operate on a path representation of the TSP. The selection of operators was inspired by Larrañaga's

survey paper from 1999[10]. The list was extended by two operators of recent date that have been shown to have good performance: The Edge Assembly Crossover [11] and the Inver-over operator[12]. For complete references the reader is referred to the paper covering my previous experimentation on AOS for TSP[1].

Displacement Mutation (DM)	Order Based Crossover (OX2)
Exchange Mutation (EM)	Position Based Crossover (POS)
Insertion Mutation (ISM)	Heuristic Crossover (HEU)
Simple Inversion Mutation (SIM)	Edge Recombination Crossover (ER)
Inversion Mutation (IVM)	Maximal Preservative Crossover (MPX)
Scramble Mutation (SM)	Voting Recombination Crossover (VR)
Partially mapped Crossover (PMX)	Alternating Position Crossover (AP)
Cycle Crossover (CX)	Inver-over operator (IO)
Order Crossover (OX1)	Edge Assembly Crossover (EAX)

Table 1. The Operators

4 Experiments and Results

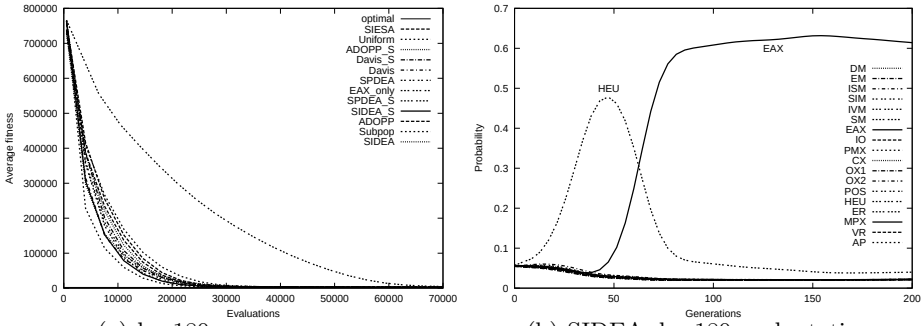
Initial experiments showed that the algorithms benefitted from larger populations as the problem instances grew. This confirms the findings in the paper describing the EAX operator[11]. The large populations are however only partly replaced in each generation. The ADOPP variant uses a steady state approach, generating only one new individual in each generation (in agreement with Julstrom's original paper [4]). As a generalisation to this approach the other algorithms were run with elite sizes of between 75% and 90% of the population. The large population sizes thus function mainly as libraries of genetic material, which maintain a certain diversity in the population.

	gr48	brg180	pcb442	nrw1379	pr2392	pcb3038
Population Size	500	600	2000	4000	4500	5000
Iterations	60,000	350,000	500,000	1,200,000	1,200,000	1,700,000

When implementing an AOS algorithm, one has the choice of including all operators in one pool or dividing mutation and crossover operators in two seperate pools. For Davis, ADOPP, SIDEA and SPDEA, both variants were implemented and included in the test set. The versions using seperate pools are labelled with the suffix `_S`.

Two non-adaptive algorithms were included in order to evaluate the absolute value of AOS. The Uniform algorithm includes all operators but uses equal probabilities for the application of them. The EAX_only uses only the EAX operator.

For all algorithms, 100 runs were done on 6 different TSP instances, all taken from the TSP benchmark problem collection TSPLIB[13]. The sizes of the problems ranged from 48 to 3038 cities. The average results are presented in Table 2.



(a) brg180 - convergence (b) SIDEA: brg180 - adaptation

Fig. 2. Convergence and Adaptation for brg180

To give an impression of convergence speed Fig. 2(a) shows the average fitness over time for the brg180 problem. The general pattern of this figure is recurrent across the range of tested problems. Fig. 2(b) shows the operator probabilities over time for the SIDEA algorithm applied on the brg180 problem. This figure is also representative for all cases. Across both problem instances and AOS methods all algorithms consistently prioritise the EAX operator in the final phase. In the initial phases the greedy nature of the other operators (typically HEU) is exploited to gain fast fitness improvements. This clearly improves performance compared to the algorithm applying the EAX operator exclusively.

Based on the data in Table 2, it is not possible to single out one algorithm as being the overall best. A certain categorisation of the algorithms can however be made. Across the range of problems, the Davis(_S), SIDEA(_S), SPDEA(_S) and Subpop algorithms all performed about equally well. To determine significance, column-wise Kruskal-Wallis rank sum tests were applied on the data responsible for the mean-values in the table. Although small differences were observed for each of the single problems, no general pattern was found.

The ADOPP and SIESA algorithms performed slightly worse than this first class. For ADOPP's case this might be ascribed to the fact that the algorithm is more greedy than for instance the Davis algorithm, and might thus overcompensate bad operators when they by chance improve the overall solution. On the other hand, it seems that the SIESA adapts too slowly. This is not entirely surprising, since it is using self-adaptation at two levels. Another reason could be the somewhat disruptive renormalisation after each operation.

Perhaps the most striking results in table 2 concern the two non-adaptive algorithms. The EAX_only algorithm performs significantly worse than any of the other algorithms. This suggests that the EAX operator is first and foremost a fine-tuning operator and should be used along with some more exploration-oriented operators.

The Uniform algorithm, on the other hand, performs remarkably well. Especially the convergence graphs show surprisingly fast convergence. It should however be noted that the convergence graphs show the population's best fit-

ness over time. No information about the general state of the population can be derived from these graphs. The apparent conflict between the Uniform algorithm's good convergence speed and somewhat worse end-results indicates that even though the algorithm is able to sustain a good elite of individuals, it does not have sufficient diversity in its best individuals to find optimal solutions. The smaller amount of good individuals is naturally explained by the fact that the EAX operator is applied with only a fraction of the probability that it receives by the different adaptation schemes.

	gr48	brg180	pcb442	nrw1379	pr2392	pcb3038
Davis_S	5046.00	1950.30	50778.21	56689.46	378067.16	137758.94
Davis	5046.36	1950.00	50781.23	56685.04	378055.16	137752.69
SIDEA_S	5047.77	1950.20	50778.14	56699.82	378084.82	137779.77
SIDEA	5046.52	1950.30	50778.07	56715.85	378336.45	137793.97
SPDEA_S	5046.33	1950.40	50778.21	56698.31	378097.53	137781.66
SPDEA	5046.00	1950.30	50778.07	56691.05	378074.36	137755.16
SIESA	5047.21	1952.60	50781.83	56695.51	378098.00	137765.02
Subpop	5046.40	1950.60	50778.14	56682.53	378057.28	137743.03
Uniform	5048.34	1952.80	50778.91	56826.32	379570.91	137956.90
ADOPP_S	5046.66	1951.80	50778.56	56702.35	378100.71	137779.00
ADOPP	5046.66	1951.70	50778.35	56689.42	378079.92	137745.00
EAX_only	5046.00	1951.30	59558.96	128619.73	2823286.85	677724.28
Optimal	5046.00	1950.00	50778.00	56638.00	378032.00	137694.00

Table 2. Average results for 100 runs

5 Discussion

Overall, the results show that performance is significantly improved when a multitude of operators are used. Even when one operator (e.g. EAX) is suspected to outperform all others, it might not be optimal throughout the whole run, and adaptive operator scheduling can exploit this fact to increase performance.

Based on the comparison of operator scheduling algorithms in this study, it is not possible to single out one of them as being the best. Many of the different methods performed equally well, which allows the selection of a method to be made based on considerations as implementation efficiency or personal taste. The Davis algorithm requires significant bookkeeping and was found somewhat elaborate to implement, while especially the SIDEA, SPDEA, SIESA and Subpop variants were implemented fairly easily. Also the fact that the self-adaptive variants have only few parameters to tune might be a reason to favour these over the others. As an extreme case, even a uniform selection between operators seems to provide reasonable results at minimal implementation costs.

Naturally, experiments should be carried out on other problem domains to determine the value of operator scheduling in general and establish whether the above conclusions hold for other domains.

Acknowledgements

The author would like to thank René Thomsen and Jakob Vesterstrøm for valuable discussions and comments during the preparation of this paper.

References

1. Boomsma, W.: Using adaptive operator scheduling on problem domains with an operator manifold: Applications to the travelling salesman problem. In: Proceedings of the 2003 Congress on Evolutionary Computation (CEC2003). Volume 2. (2003) 1274–1279
2. Angeline, P.J.: Adaptive and self-adaptive evolutionary computations. In Palaniswami, M., Attikiouzel, Y., eds.: Computational Intelligence: A Dynamic Systems Perspective. IEEE Press (1995) 152–163
3. Davis, L.: Adapting operator probabilities in genetic algorithms. In Schaffer, J.D., ed.: Proceedings of the Third International Conference on Genetic Algorithms, San Mateo, CA, Morgan Kaufman (1989)
4. Julstrom, B.A.: What have you done for me lately? Adapting operator probabilities in a steady-state genetic algorithm. In Eshelman, L., ed.: Proceedings of the Sixth International Conference on Genetic Algorithms, San Francisco, CA, Morgan Kaufmann (1995) 81–87
5. Schlierkamp-Voosen, D., Mühlenbein, H.: Strategy adaptation by competing subpopulations. In Davidor, Y., Schwefel, H.P., Männer, R., eds.: Parallel Problem Solving from Nature – PPSN III, Berlin, Springer (1994) 199–208
6. Spears, W.M.: Adapting crossover in evolutionary algorithms. In McDonnell, J.R., Reynolds, R.G., Fogel, D.B., eds.: Proc. of the Fourth Annual Conference on Evolutionary Programming, Cambridge, MA, MIT Press (1995) 367–384
7. Storn, R., Price, K.: Differential evolution a simple and efficient heuristic for global optimisation over continuous spaces. Journal of Global Optimization **11** (1997) 341–359
8. Bäck, T., Hoffmeister, F., Schwefel, H.P.: A survey of evolution strategies. In Belew, R., Booker, L., eds.: Proceedings of the Fourth International Conference on Genetic Algorithms, San Mateo, CA, Morgan Kaufman (1991) 2–9
9. Bäck, T.: Evolution strategies: An alternative evolutionary algorithm. In Alliot, J., Lutton, E., Ronald, E., Schoenhauer, M., Snyers, D., eds.: Artificial Evolution, Springer (1996) 3–20
10. Larrañaga, P., Kuijpers, C.M.H., Murga, R.H., Inza, I., Dizdarevic, S.: Genetic algorithms for the travelling salesman problem: A review of representations and operators. Artificial Intelligence Review **13** (1999) 129–170
11. Nagata, Y., Kobayashi, S.: Edge assembly crossover: A high-power genetic algorithm for the travelling salesman problem. In Bäck, T., ed.: Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA97), San Francisco, CA, Morgan Kaufmann (1997)
12. Tao, G., Michalewicz, Z.: Inver-over operator for the TSP. In Eiben, A.E., Bäck, T., Schoenauer, M., Schwefel, H.P., eds.: Parallel Problem Solving from Nature – PPSN V, Berlin, Springer (1998) 803–812 Lecture Notes in Computer Science 1498.
13. Reinelt, G.: TSPLIB — a traveling salesman problem library. ORSA Journal on Computing **3** (1991) 376–384

Multiple Sequence Alignment Using SAGA: Investigating the Effects of Operator Scheduling, Population Seeding, and Crossover Operators

René Thomsen and Wouter Boomsma

Bioinformatics Research Center (BiRC)
Department of Computer Science
University of Aarhus
Ny Munkegade, Bldg. 540.
DK-8000 Aarhus C, Denmark
{thomsen,wb}@daimi.au.dk

Abstract. Multiple sequence alignment (MSA) is a fundamental problem of great importance in molecular biology. In this study, we investigated several aspects of SAGA, a well-known genetic algorithm (GA) for solving MSA problems. The SAGA algorithm is important because it represents a successful attempt at applying GAs to MSA and since it is the first GA to use operator scheduling on this problem. However, it is largely undocumented which elements of SAGA that are vital to it's performance. An important finding in this study is that operator scheduling does not improve the performance of SAGA compared to a uniform selection of operators. Furthermore, the experiments show that seeding SAGA with a ClustalW-derived alignment allows the algorithm to discover alignments of higher quality compared to the traditional initialization scheme with randomly generated alignments. Finally, the experimental results indicate that SAGA's performance is largely unaffected when the crossover operators are disabled. Thus, the major determinant of SAGA's success seems to be the mutation operators and the scoring functions used.

1 Introduction

During the last decades, the amount of available biological sequence data (DNA, RNA, and proteins) has increased exponentially. This wealth of new information requires automated methods that can assist practitioners in the process of analyzing and interpreting the data. Valuable tools in this context are multiple sequence alignment (MSA) programs, which allow for the comparison of multiple sequences.

Among the vast number of possible applications, MSAs can be used to (i) infer phylogenetic relationships of organisms, (ii) discover conserved motifs that might be of great importance on the levels of transcription, translation, or structure/function, and finally (iii) assist secondary and tertiary structure prediction methods.

Initially, the most popular methods for obtaining MSAs used dynamic programming (DP) because DP can guarantee a mathematically optimal alignment given the commonly used sum-of-pairs (SOP) scoring function [1]. However, DP-based methods can only handle a relatively small number of sequences because the size of the MSA problem space increases dramatically with the number of sequences in the alignment and their length. In fact, finding the optimal MSA solution wrt. the SOP score is known to be NP-hard [2].

In order to solve larger problem instances several heuristics have been introduced. The most popular heuristic is ClustalW [3], an algorithm that belongs to the class of progressive alignment methods [4]. These methods gradually construct an alignment by first estimating the evolutionary distance between all sequences to be aligned and then align the sequences in order of decreasing similarity. Although the progressive methods are very fast they typically suffer from entrapment in local optima because they optimize the alignment in a pairwise manner, not taking the entire alignment into account.

To overcome this problem several stochastic heuristics have been applied to MSA, such as simulated annealing [5] and evolutionary algorithms (EAs) [6, 7]. Typically, these methods start with randomly generated candidate alignments that are gradually improved using several variation operators. However, most of the stochastic methods suffer from very long run-times [6].

SAGA (Sequence Alignment by Genetic Algorithm) [6] introduced the idea of operator scheduling (OS) for MSA based on the assumption that the scheduling of the operators would improve the overall performance of the algorithm. Preliminary experiments with several OS schemes on MSA using the MSAEA [8] did not indicate any improvements for OS compared to chosing the operators randomly with a uniform probability. More specifically, these observations raised the question of whether OS has any effect at all when applied to EAs for MSA.

Moreover, Thomsen et al. [9, 8] investigated the effects of seeding an EA with a ClustalW-derived solution. The experimental results indicated that seeding the EA resulted in a marked improvement in runtime needed to derive solutions of high quality. Furthermore, it was shown that the resulting alignments were significantly better than the ClustalW seed, making the EA useful as an alignment improver.

These findings motivated us to investigate the following aspects of SAGA: (i) operator scheduling compared to uniform choice of variation operators, (ii) seeding the initial population with a ClustalW-derived alignment compared to random initialization, and finally (iii) the effect of crossover operators compared to using mutation only. These investigations were conducted on selected MSA benchmark problems obtained from the BAliBASE sequence alignment database [10].

The experimental results show that, contrary to common belief, operator scheduling does not improve the overall performance of SAGA compared to a uniform selection of operators. Moreover, the results indicate that SAGA is able to obtain better results using seeds compared to random initialization given the

same number of fitness evaluations. Finally, the use of crossover operators did not generally increase the performance.

2 SAGA

In 1996 Notredame and Higgins introduced SAGA [6], one of the first genetic algorithms (GAs) for MSA. Basically, SAGA resembles a standard GA with a population of candidate alignments (individuals) that are subjected to variation and selection. During selection the individuals compete for survival and the most promising ones are transferred to the next generation. Contrary to other GAs solving MSA, SAGA provides a total of 25 variation operators (19 mutation and 6 crossover) with a variety of functionality, such as modifying gap regions (adding, deleting, moving gaps) and combining promising regions. A complete description of the operators is beyond of the scope of this paper, see [6] for a detailed description. The default initialization mode in SAGA is to randomly initialize all individuals by prepending a randomly chosen number of gaps to each sequence.

SAGA differs from all other MSA algorithms regarding the utilization of the operators. It uses an operator scheduling (OS) strategy originally described by Davis in 1989 [11] to select which operators to use. The OS strategy works as follows: Each operator is assigned a probability for its application. Initially, these probabilities are all equal, but during the course of the run they are adapted based on the recent performance of the operator. Operators are rewarded when they create an individual with a fitness greater than any of the current individuals in the population. Furthermore, operators responsible for the individuals' parents and more distant ancestors are rewarded with some percentage of the original reward. This is motivated by the fact that a series of suboptimal solutions is often necessary in order to reach a new optimal solution and corresponding operators therefore should be rewarded. With certain intervals, a new probability setting is computed as a weighted sum of the previous setting and the distribution of rewards among the operators.

The motivation for using OS in SAGA is that it is difficult to determine in advance which of the 25 operators to apply. The problem gets more complicated since the utilization of each operator might depend on the actual alignment problem being solved and optimal usage of operators might change during the course of the optimization run. The idea is that by measuring the performance of operators during the run, operators can be continuously scheduled so that the best operators are used at all times.

SAGA provides three different ways of scoring alignments, (i) sum-of-pairs without weights (SOP), (ii) sum-of pairs using weights (WSOP), and (iii) Consistency based Objective Function For alignmEnt Evaluation (COFFEE) [12]. The three scoring functions are briefly described below (see [6] and [12] for more details).

The SOP and WSOP scoring functions are almost identical except for the additional sequence weights used in WSOP. Equation 1 shows how the total

score of an alignment is calculated. N specifies the number of sequences in the alignment, S_i and S_j are two aligned sequences from the given alignment, and W_{ij} is the corresponding weight (a detailed description of the weighting scheme used is provided in [12]). When no weights are provided, i.e., using SOP, all weights are set to 100 (default setting in SAGA). The $COST$ function calculates the substitution scores for each pair of residues in sequence S_i and S_j using e.g, BLOSUM or PAM matrices. Furthermore, the $COST$ function includes affine gap penalties, e.g., penalties for gap opening and extension (GOP and GEP, respectively). Terminal gaps are only penalized for extension, not for opening. Moreover, SAGA transforms the (W)SOP scoring functions into minimization problems by scaling the entries of the substitution matrix.

$$(W)SOP = \sum_{i=2}^{N} \sum_{j=1}^{i-1} W_{ij} \times COST(S_i, S_j) \; . \tag{1}$$

The COFFEE score is defined as a maximization problem where the task is to maximise the consistency between the residue pairs in the alignment and the residue pairs observed in a library generated by pairwise alignments between all the sequences. Equation 2 shows how the score is calculated.

$$COFFEE = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} W_{ij} \times SCORE(A_{ij}) \; . \tag{2}$$

$SCORE(A_{ij})$ is the number of aligned pairs of residues that are shared between A_{ij} and the generated library, where A_{ij} is the pairwise projection of sequence S_i and S_j obtained from the evaluated candidate alignment.

For the experiments described in this study we used SAGA V0.95, which is publically available from `http://igs-server.cnrs-mrs.fr/~cnotred/`. The only modification that we made to the source code was to change the termination criteria so that SAGA terminates when a certain number of fitness evaluations has occurred. This change was made because the default termination criteria in SAGA to terminate after 100 consecutive generations without improvements made a general comparison between different settings difficult.

3 Experiments

3.1 BAliBASE Benchmarks

The BAliBASE database [10] contains 142 multiple sequence alignments, which are manually refined from the known 3D structures of proteins. This makes it possible to evaluate the quality of MSA algorithms regarding their ability to derive *true* (biological plausible) alignments. Table 1 shows the protein sequence data sets used in our experiments. All seven data sets were selected from the first reference set of the BAliBASE database (version 2, `http://bess.u-strasbg.fr/BioInfo/BAliBASE2/`).

Table 1. BAliBASE data sets used in the experiments. NSEQ = number of sequences, LSEQ = length of sequences, SEQID = percent residue identity

Data set	NSEQ	LSEQ (min,max,avg)	SEQID
1idy	5	(49,58,53.6)	<25%
1aboA	5	(49,80,63.6)	<25%
kinase	5	(263,276,270.2)	<25%
1hfh	5	(116,132,121.2)	20-40%
1pfc	5	(108,117,112)	20-40%
1pii	4	(247,259,251.5)	20-40%
451c	5	(70,87,80)	20-40%

3.2 Evaluation Measures

The results from the SAGA runs were evaluated according to each of the scoring functions described in Section 2. Furthermore, the quality of the overall best found alignments were compared with the BAliBASE reference alignments using the BAliBASE evaluation measures described below.

The BAliBASE sum-of-pairs score (SPS) is calculated as follows: Given a candidate alignment (individual) of N sequences containing M columns, the i'th column in the alignment was designated by A_{i1}, A_{i2}, ..., A_{iN}. For each pair of residues A_{ij} and A_{ik} we defined p_{ijk} such that $p_{ijk} = 1$ if residues A_{ij} and A_{ik} from the candidate alignment were aligned with each other in the reference alignment. Otherwise, $p_{ijk} = 0$. The score for the i'th column is thus:

$$S_i = \sum_{j=1, j\neq k}^{N} \sum_{k=1}^{N} p_{ijk} \; . \tag{3}$$

The overall SPS for the candidate alignment is:

$$SPS = \sum_{i=1}^{M} \frac{S_i}{\sum_{i=1}^{M_r} S_{ri}} \; . \tag{4}$$

where M_r is the number of columns in the reference alignment and S_{ri} is the score S_i for the i'th column in the reference alignment.

The BAliBASE column score (CS) is calculated as follows: Given an alignment as described above, the score C_i for the i'th column is equal to 1 if all the residues in the column are aligned in the reference alignment. Otherwise, C_i is set to 0. The overall CS for the candidate alignment is:

$$CS = \sum_{i=1}^{M} \frac{C_i}{M} \; . \tag{5}$$

The ranges of SPS and CS are 0.0-1.0, where higher values indicate closer resemblance with the BAliBASE reference alignment. When calculating both scores we

used the annotation files provided with BAliBASE to identify core blocks in the reference alignment, i.e., only the regions that were treated as being important were used in the calculation.

3.3 Experimental Setup and Data Sampling

In this study we used the default parameter settings provided with SAGA. The population size was set to 100 and the total number of fitness evaluations allowed was set to 200000 (max number of evaluations in the termination criterion). The SOP and WSOP scoring functions used the BLOSUM-45 substitution matrix and gap penalties were set to $GOP = 8$ and $GEP = 12$ (SAGA default settings).

Different configurations of SAGA were investigated regarding the seven alignment problems shown in Table 1 (see section 4 for more details). Each experiment was repeated 30 times with different random seeds, and the average of the 30 best alignment scores was recorded.

4 Results

The performance of SAGA was evaluated regarding the seven alignment benchmarks described in section 3.1. The experiments were designed to provide answers to the following three main questions: (i) does operator scheduling (OS) outperform simple uniform choice of operators?, (ii) what is the effect of using seeding compared to random initialization only?, and (iii) does crossover improve performance compared to using mutation only?

Table 2 summarizes the results from all the experiments. The *Configuration* column shows the choice of configuration for each particular experiment (see caption for Table 2 for information on the abbreviations used). The remaining columns show the BAliBASE CS and SPS measures (mean of 30 runs) for each of the tested alignments. The significance of the results was validated using the Wilcoxon Signed Rank Test (statistical p-values are provided in supplementary material).

Overall, the results from table 2 show that OS is not an advantage compared to a uniform choice of operator (the differences are not statistically significant ($p > 0.0625$)). Moreover, seeding improves the performance of SAGA in four of the seven test cases (the results are statistically significant with $p < 0.05$ for *1aboA*, *kinase*, *1hfh*, and *1pii*). For the remaining test cases, the differences observed are not significant. Finally, crossover improves the performance for *kinase* and *1hfh* whereas for *1aboA* using crossover is a disadvantage (the results are statistically significant with $p < 0.05$). For the four other test cases, using crossover does not make any performance difference.

In conclusion, SAGA using seeding, crossover, OS, and the SOP or WSOP scoring function (S,C,O, SOP/WSOP) or the same settings except OS (S,C,NO, SOP/WSOP) generally performed better than all other configurations. The small differences between these settings were not statistically significant according to the Wilcoxon Signed Rank Test.

Table 2. BAliBASE evaluation measures on tested benchmarks using different SAGA configurations. *R* (random initialization), *S* (ClustalW-seed), *C/NO* (crossover/no-crossover), *O/NO* (operator-scheduling/uniform-choice), *SOP/WSOP/COFFEE* (scoring functions)

	Data set						
	1idy	1aboA	kinase	1hfh	1pfc	1pii	451c
Configuration	CS/SPS	CS/SPS	CS/SPS	CS/SPS	CS/SPS	CS/SPS	CS/SPS
R, C, O, SOP	0.380/0.565	0.484/0.721	0.353/0.606	0.878/0.944	0.991/0.997	0.805/0.895	0.622/0.746
R, C, O, WSOP	0.380/0.550	0.518/0.730	0.304/0.558	0.889/0.950	0.940/0.979	0.805/0.891	0.638/0.744
R, C, O, COFFEE	0.167/0.510	0.570/0.775	0.541/0.734	0.847/0.936	0.893/0.959	0.800/0.882	0.611/0.787
R, NC, O, SOP	0.380/0.540	0.566/0.769	0.272/0.565	0.853/0.931	0.990/0.997	0.775/0.877	0.620/0.717
R, NC, O, WSOP	0.380/0.550	0.562/0.765	0.272/0.558	0.857/0.936	0.940/0.979	0.768/0.873	0.666/0.732
R, NC, O, COFFEE	0.207/0.520	0.570/0.774	0.468/0.705	0.853/0.937	0.890/0.958	0.801/0.883	0.635/0.794
R, C, NO, SOP	0.380/0.550	0.508/0.732	0.362/0.617	0.894/0.951	0.989/0.996	0.805/0.896	0.620/0.744
R, C, NO, WSOP	0.380/0.550	0.496/0.725	0.298/0.569	0.897/0.955	0.940/0.979	0.809/0.893	0.641/0.765
R, C, NO, COFFEE	0.151/0.507	0.570/0.774	0.531/0.732	0.851/0.936	0.891/0.958	0.801/0.883	0.606/0.784
R, NC, NO, SOP	0.380/0.548	0.555/0.764	0.305/0.583	0.863/0.935	0.992/0.997	0.799/0.890	0.600/0.718
R, NC, NO, WSOP	0.380/0.550	0.544/0.758	0.278/0.566	0.859/0.935	0.940/0.979	0.804/0.890	0.627/0.739
R, NC, NO, COFFEE	0.203/0.533	0.569/0.775	0.450/0.692	0.822/0.924	0.890/0.958	0.802/0.884	0.639/0.796
S, C, O, SOP	0.380/0.595	0.566/0.768	0.621/0.792	0.896/0.953	0.985/0.995	0.810/0.899	0.589/0.698
S, C, O, WSOP	0.380/0.543	0.566/0.766	0.627/0.795	0.889/0.950	0.940/0.979	0.810/0.893	0.618/0.709
S, C, O, COFFEE	0.186/0.542	0.570/0.776	0.554/0.737	0.845/0.934	0.890/0.958	0.800/0.882	0.630/0.794
S, NC, O, SOP	0.380/0.596	0.567/0.767	0.531/0.755	0.883/0.947	0.988/0.996	0.810/0.897	0.620/0.711
S, NC, O, WSOP	0.380/0.541	0.568/0.769	0.529/0.752	0.894/0.953	0.940/0.979	0.810/0.892	0.630/0.715
S, NC, O, COFFEE	0.207/0.528	0.570/0.776	0.527/0.729	0.859/0.939	0.892/0.959	0.800/0.882	0.631/0.794
S, C, NO, SOP	0.380/0.594	0.567/0.767	0.608/0.787	0.909/0.959	0.985/0.995	0.810/0.899	0.585/0.696
S, C, NO, WSOP	0.380/0.548	0.567/0.768	0.616/0.800	0.903/0.957	0.940/0.979	0.810/0.892	0.615/0.707
S, C, NO, COFFEE	0.183/0.525	0.570/0.776	0.559/0.736	0.834/0.929	0.892/0.959	0.800/0.882	0.629/0.796
S, NC, NO, SOP	0.380/0.594	0.568/0.766	0.547/0.763	0.890/0.950	0.988/0.996	0.810/0.898	0.605/0.705
S, NC, NO, WSOP	0.380/0.550	0.568/0.767	0.581/0.782	0.878/0.945	0.940/0.979	0.810/0.892	0.624/0.712
S, NC, NO, COFFEE	0.218/0.552	0.570/0.776	0.518/0.726	0.816/0.921	0.893/0.959	0.800/0.882	0.633/0.795

Finally, the best configuration found in the previous experiments was compared to the alignments obtained from SAGA using default parameter settings (R,C,O,SOP), ClustalW and T-Coffee [13]. Table 3 shows the CS and SPS values for each of the seven testcases using one of the tested alignment programs. In five out of seven cases, SAGA with the best-found configuration outperformed the other alignment programs wrt. the BAliBASE evaluation measures (three times using CS and five times using SPS). For the remaining two cases, SAGA using random initialization obtained slightly higher scores than using the ClustalW seed for *1pfc*. However, the differences are not statistically significant. SAGA was not able to improve the *451c* problem. In fact, the resulting alignment had a worse score compared to the ClustalW seed. This case shows that an improvement of the SOP score does not always correlate with an improvement of the alignment quality, i.e., a better CS or SPS value. In conclusion, using SAGA with the new parameter settings outperformed SAGA using default values for five of the seven tested problems, indicating the importance of population seeding when the number of evaluations are fixed at 200000.

Table 3. BAliBASE evaluation measures on tested benchmarks using different MSA programs. *SAGA(seeding)* is SAGA with the best-found settings and ClustalW seeding (see previous table) and *SAGA(random)* is SAGA using default parameter settings and random initialization. All SAGA results represent mean values obtained from 30 runs

	MSA program			
Data set	SAGA (seeding) CS/SPS	SAGA (random) CS/SPS	ClustalW CS/SPS	T-Coffee CS/SPS
1idy	**0.380**/**0.595**	**0.380**/0.565	**0.380**/0.588	0.000/0.150
1aboA	**0.566**/**0.768**	0.484/0.721	0.540/0.755	0.150/0.570
kinase	0.621/**0.792**	0.353/0.606	**0.630**/0.788	0.600/0.769
1hfh	**0.896**/**0.953**	0.878/0.944	0.770/0.900	0.870/0.938
1pfc	0.985/0.995	**0.991**/**0.997**	0.750/0.903	0.940/0.979
1pii	0.810/**0.899**	0.805/0.895	0.740/0.855	**0.820**/**0.899**
451c	0.589/0.698	0.622/0.746	**0.700**/0.755	0.640/**0.786**

5 Discussion

In this paper, we investigated different aspects of SAGA, such as (i) scheduling operators to obtain optimal usage compared to uniform selection of all available operators, (ii) seeding the initial population with a ClustalW derived alignment compared to random initialization only, and finally (iii) using crossover operators compared to using mutation-based operators only.

Overall, the experimental results showed that operator scheduling (OS) did not perform any better than simply choosing the operators uniformly (see table 2). This observation differs from the general opinion that OS is very useful when

solving optimization problems with many variation operators [6]. Preliminary experiments trying out other OS strategies confirmed the reported results. In all cases uniform selection of operators was as good or better than the tested methods. A possible explanation of why OS does not increase the performance is that repeated usage of an operator that is believed to be 'good' does not necessarily result in a stepwise improvement of alignment quality. The main reason seems to be the stochastic nature of the variation operators which usually randomly select the sequences and/or gap-regions to be modified. However, more in-depth investigations on the search-spaces induced by the MSA scoring schemes are needed to fully understand why OS does not work very well on this particular problem.

Moreover, our study showed that in most cases the seeding approach was able to obtain similar or better scores than the random initialization approach (using the same parameter settings and number of evaluations). Again, this observation is in contrast to the original study by Notredame and Higgins [6] where seeding has been avoided because it was believed to bias the search to local optima. Furthermore, using random initialization without seeding typically required twice as many fitness evaluations as the seeding approach to obtain similar fitness and CS/SPS values (see supplementary material for more details).

Generally, the use of crossover did not improve the performance of SAGA on the tested data sets. On the *kinase* and *1hfh* test cases using crossover operators resulted in slightly better CS/SPS scores whereas omitting crossover improved the results on the *1aboA* test case. Whether to use crossover or not seems to depend on the actual data set in question. More experiments on other BAliBASE data sets might give some guidelines on when to apply or omit crossover operators.

Finally, the comparison to other MSA programs showed that SAGA is able to improve the initial alignment provided by ClustalW in five out of 7 cases (see table 3). Typically, the runtime needed by SAGA was between 40-240 seconds on a 1.8GHz Pentium-IV PC (run-time data is provided in the supplementary material) making it a valuable tool as an alignment improver. Furthermore, SAGA using the seeding approach is able to obtain the overall best scores in five out of seven cases compared to ClustalW, T-Coffee, and SAGA using default random initialization.

In conclusion, seeding SAGA with ClustalW solutions improved the convergence properties of SAGA (see convergence graphs in the supplementary material) thus lowering the total number of fitness evaluations needed to obtain alignments of good quality (wrt. CS and SPS measures). The results also indicate that the benefits of using OS and crossover operators are questionable. Repeating the investigations described in this study on all the BAliBASE data sets will provide us with a better indication on whether the use of OS and crossover operators are needed in general.

6 Supplementary Material

Additional experimental results (fitness and runtime tables, convergence graphs, histograms comparing various SAGA configurations etc.) are publically available from `http://www.daimi.au.dk/~thomsen/evobio2004/`.

Acknowledgments

The authors would like to thank the Danish Research Council for financial support. The authors also thank Jakob Vesterstrøm for valuable comments on draft versions of the manuscript.

References

1. Gupta, S., J.D. Kececioglu, J., Schaffer, A.: Improving the practical space and time efficiency of the shortest-paths approach to sum-of-pairs multiple sequence alignment. J. Comp. Bio. **2** (1995) 459–472
2. Wang, L., Jiang, T.: On the complexity of multiple sequence alignment. Journal of Computational Biology **1** (1994) 337–348
3. Thompson, J., Higgins, D., Gibson, T.: Clustal w: improving the sensitivity of progressive multiple sequence alignment through sequence weighting. Nucleic Acids Research **22** (1994) 4673–4680
4. Feng, D., Doolittle, R.: Progressive sequence alignment as a prerequisite to correct phylogenetic trees. J. Mol. Evol. **25** (1987) 351–360
5. Kim, J., Pramanik, S., Chung, M.: Multiple sequence alignment using simulated annealing. Computer Applications in the Biosciences (CABIOS) **10** (1994) 419–426
6. Notredame, C., Higgins, D.: Saga: Sequence alignment by genetic algorithm. Nucleic Acids Research **24** (1996) 1515–1524
7. Chellapilla, K., Fogel, G.B.: Multiple sequence alignment using evolutionary programming. In: Proceedings of the First Congress of Evolutionary Computation (CEC-1999). (1999) 445–452
8. Thomsen, R., Fogel, G., Krink, T.: Improvement of clustal-derived sequence alignments with evolutionary algorithms. In: Proceedings of the Fifth Congress on Evolutionary Computation (CEC-2003). (2003)
9. Thomsen, R., Fogel, G., Krink, T.: A clustal alignment improver using evolutionary algorithms. In: Proceedings of the Fourth Congress on Evolutionary Computation (CEC-2002). Volume 1. (2002) 121–126
10. Thompson, J., Plewniak, F., Poch, O.: Balibase: a benchmark alignment database for the evaluation of multiple alignment programs. Bioinformatics **15** (1999) 87–88
11. Davis, L.: Adapting operator probabilities in genetic algorithms. In: Proceedings of the Third International Conference on Genetic Algorithms (ICGA III). (1989) 61–69
12. Notredame, C., Holm, L., Higgins, D.: Coffee: An objective function for multiple sequence alignments. Bioinformatics **14** (1998) 407–422
13. Notredame, C., Higgins, D., Heringa, J.: T-coffee: A novel method for fast and accurate multiple sequence alignment. J. Mol. Biol. **302** (2000) 205–217

Bibliography

[1] P. J. Angeline. Adaptive and self-adaptive evolutionary computations. In M. Palaniswami and Y. Attikiouzel, editors, *Computational Intelligence: A Dynamic Systems Perspective*, pages 152–163. IEEE Press, 1995.

[2] W. Banzhaf. The "molecular" traveling salesman. *Biological Cybernetics*, 64:7–14, 1990.

[3] T. Bäck. Evolution strategies: An alternative evolutionary algorithm. In J. Alliot, E. Lutton, E. Ronald, M. Schoenhauer, and D. Snyers, editors, *Artificial Evolution*, pages 3–20. Springer, 1996.

[4] T. Bäck, F. Hoffmeister, and H.-P. Schwefel. A survey of evolution strategies. In R. Belew and L. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 2–9, San Mateo, CA, 1991. Morgan Kaufman.

[5] W. Boomsma. Using adaptive operator scheduling on problem domains with an operator manifold: Applications to the travelling salesman problem. In *Proceedings of the 2003 Congress on Evolutionary Computation (CEC-2003)*, volume 2, pages 1274–1279, 2003.

[6] L. Cai, D. Juedes, and E. Liakhovitch. Evolutionary computation techniques for multiple sequence alignment. In *Proceedings of the Second Congress on Evolutionary Computation (CEC-2000)*, pages 829–835, 2000.

[7] H. Carrillo and D. Lipman. The multiple sequence alignment problem in biology. *SIAM journal of applied mathematics*, 48:1073–1082, 1988.

[8] R. Cary and G. Stormo. Graph-theoretic approach to rna modeling using comparative data. In *Proceedings 3rd International Conf. on Intelligent Systems for Molecular Biology*, pages 75–80, 1995.

[9] K. Chellapilla and G. B. Fogel. Multiple sequence alignment using evolutionary programming. In *Proceedings of the First Congress of Evolutionary Computation (CEC-1999)*, pages 445–452, 1999.

[10] G. Croes. A method for solving the traveling salesman problems. *Operations Research*, 6:791–812, 1958.

[11] L. Davis. Applying adaptive algorithms to epistatic domains. In *Proceedings of the International Joint Conference on Artificial Intelligence*, volume 1, pages 161–163, 1985.

[12] L. Davis. Adapting operator probabilities in genetic algorithms. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, San Mateo, CA, 1989. Morgan Kaufman.

[13] M. Dayhoff, R. Schwartz, and B. Orcutt. A model of evolutionary change in proteins. *Atlas of protein sequence and structure*, 5:345–352, 1978.

[14] K. A. De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* PhD thesis, University of Michigan, Ann Arbor, MI, 1975. Dissertation Abstracts International 36(10), 5140B, University Microfilms Number 76-9381.

[15] A. E. Eiben, I. G. Sprinkhuizen-Kuyper, and B. A. Thijssen. Competing crossovers in an adaptive ga framework. In *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, pages 787–792, 1998.

[16] D. Fogel. A parallel processing approach to a multiple travelling salesman problem using evolutionary programming. In *Proceedings of the Fourth annual Symposium on Parallel Processing*, pages 318–326, Fullerton, California, April 1990.

[17] D. B. Fogel. An evolutionary approach to the travelling salesman problem. *Biological Cybernetics*, 60(2):139–144, 1988.

[18] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution.* John Wiley & Sons, New York, 1966.

[19] F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8(1):156–166, 1977.

[20] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13:533–549, 1986.

[21] F. Glover. Tabu search part i. *ORSA Journal on Computing 1*, pages 190–206, 1989.

[22] F. Glover. Tabu search part ii. *ORSA Journal on Computing 2*, pages 4–32, 1990.

[23] D. E. Goldberg and R. Lingle Jr. Alleles, loci, and the traveling salesman problem. In J. J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms and Their Applications.* Lawrence Erlbaum Associates, Publishers, 1985.

[24] Ágoston Endre Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Trans. on Evolutionary Computation*, 3(2):124–141, 1999.

[25] J. J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 16(1):122–128, 1986.

[26] J. J. Grefenstette. *Incorporating problem specific knowledge into genetic algorithms*, pages 42–60. Morgan Kaufmann, 1987.

[27] D. Gusfield. *Algorithms on strings, trees, and sequences: computer science and computational biology.* Cambridge University Press, 1997.

[28] K. Helsgaun. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European J. Oper. Res.*, 126(1):106–130, 2000.

[29] S. Henikoff and J. Henikoff. Amino Acid Substitution Matrices from Protein Blocks. *Proceedings of the National Academy of Sciences*, 89(22):10915–10919, 1992.

[30] R. Hinterding, Z. Michalewicz, and Ágoston Endre Eiben. Adaptation in evolutionary computation: A survey. In *IEEECEP: Proceedings of The IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, 1997.

[31] M. Hirosawa, M. Hoshida, M. Ishikawa, and T. Toya. MASCOT: multiple alignment system for protein sequences based on three- way dynamic programming. *Comput. Appl. Biosci.*, 9(2):161–167, 1993.

[32] J. H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, MI, 1975.

[33] J. Horn, D. E. Goldberg, and K. Deb. Long path problems. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Proceedings of theInternational Conference on Evolutionary Computation, theThirdConference on Parallel Problem Solving from Nature(PPSN III)*, volume 866, pages 149–158, Jerusalem, 9–14 1994. Springer.

[34] T.-T. Horng, C.-M. Lin, B.-J. Liu, and C.-Y. Kao. Using genetic algorithms to solve multiple sequence alignments. In *Proceedings of the Second Genetic and Evolutionary Computation Conference (GECCO-2000)*, pages 883–890, 2000.

[35] M. Ishikawa, T. Toya, M. Hoshida, K. Nitta, A. Ogiwara, and M. Kanehisa. Multiple sequence alignment by parallel simulated annealing. *Comput. Appl. Biosci.*, 9(3):267–273, 1993.

[36] D. Johnson. Local optimization and the traveling salesman problem. In M. Paterson, editor, *Lecture Notes in Computer Science*, volume 443, pages 446–461. Springer, 1990. Proceedings of the 17th Colloquium on Automata, Languages and Programming.

[37] D. Johnson and L. McGeoch. The Traveling Salesman Problem: A Case Study in Local Optimization. Draft of November 20, 1995. To appear as a chapter in the book *Local Search in Combinatorial Optimization*, E. H. L. Aarts and J. K. Lenstra (eds.), John Wiley and Sons, New York., 1995.

[38] K. A. D. Jong and W. M. Spears. An analysis of the interacting roles of population size and crossover in genetic algorithms. In H. P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature - Proceedings of 1st Workshop, PPSN 1*, volume 496, pages 38–47, Dortmund, Germany, 1-3 1990. Springer-Verlag, Berlin, Germany.

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

[40] J. Kim, S. Pramanik, and M. Chung. Multiple sequence alignment using simulated annealing. *Comput. Appl. Biosci.*, 10(4):419–426, 1994.

[41] T. Krink, P. Rickers, and R. Thomsen. Applying self-organised criticality to evolutionary algorithms. In H.-P. S. Marc Schoenauer, Kalyanmoy Deb, Günter Rudolph, Xin Yao, Evelyne Lutton, Juan Julian Merelo, editor, *Parallel Problem Solving from Nature - PPSN VI 6th International Conference*, Paris, France, 2000. Springer Verlag.

[42] P. Larrañaga, C. Kuijpers, M. Poza, and R. Murga. Decomposing bayesian networks: triangulation of the moral graph with genetic algorithms. *Statistics and Computing*, 7(1):19–34, 1997.

[43] P. Larrañaga, C. M. H. Kuijpers, R. H. Murga, I. Inza, and S. Dizdarevic. Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial Intelligence Review*, 13(2):129–170, 1999.

[44] E. Lawler, J. Lenstra, A. R. Kan, and D. Shmoys, editors. *The Traveling Salesman Problem*. Wiley, 1985.

[45] S. Lin. Computer solutions of the traveling salesman problem. *Bell Systems Journal*, 44:2245–2269, 1965.

[46] D. Lipman, S. Altschul, and J. Kececioglu. A tool for multiple sequence alignment. In *Proceedings of the National Academy of Sciences*, volume 86, pages 4412–4415, 1989.

[47] H. Mühlenbein. Parallel genetic algorithms, population genetics and combinatorial optimization. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, San Mateo, CA, 1989. Morgan Kaufman.

[48] H. Mühlenbein, M. Gorges-Schleuter, and O. Krämer. Evolution algorithms in combinatorial optimization. *Parallel Computing*, 7:65–85, 1988.

[49] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, Berlin, 1992.

[50] Z. Michalewicz and D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer, Berlin, 2000.

[51] Y. Nagata and S. Kobayashi. Edge assembly crossover: A high-power genetic algorithm for the travelling salesman problem. In T. Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA97)*, San Francisco, CA, 1997. Morgan Kaufmann.

[52] C. Notredame and D. Higgins. Saga: Sequence alignment by genetic algorithm. *Nucleic Acids Research*, 24(8):1515–1524, 1996.

[53] C. Notredame, D. Higgins, and J. Heringa. T-coffee: A novel method for fast and accurate multiple sequence alignment. *J. Mol. Biol.*, 302:205–217, 2000.

[54] C. Notredame, L. Holm, and D. Higgins. Coffee: An objective function for multiple sequence alignments. *Bioinformatics*, 14(5):407–422, 1998.

[55] I. M. Oliver, D. J. Smith, and J. R. C. Holland. A study of permutation crossover operators on the travelling salesman problem. In J. J. Grefenstette, editor, *Genetic algorithms and their applications : Proc. of the second Int. Conf. on Genetic Algorithms*, pages 224–230, Hillsdale, NJ, 1987. Lawrence Erlbaum Assoc.

[56] C. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*, chapter 18. Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[57] I. Rechenberg. Cybernetic solution path of an experimental problem. UK, 1965. Royal Aircraft Establishment.

[58] I. Rechenberg. Evolutionsstrategie: Optimierung technischer systeme nach prinzipien der biologischen evolution. Stuttgart, 1973. Frommann-Holzboog.

[59] G. Reinelt. TSPLIB — a traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384, 1991.

[60] N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4:406–425, 1987.

[61] D. Schlierkamp-Voosen and H. Mühlenbein. Strategy adaptation by competing subpopulations. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Parallel Problem Solving from Nature – PPSN III*, pages 199–208, Berlin, 1994. Springer.

[62] J. Setubal and J. Meidanis. *Introduction to computational molecular biology*. International Thomson Publishing, 1997.

[63] S.Lin and B.W.Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21:498–516, 1973.

[64] J. E. Smith and T. C. Fogarty. Operator and parameter adaptation in genetic algorithms. *Soft Computing*, 1(2):81–87, 1997.

[65] W. M. Spears. Adapting crossover in evolutionary algorithms. In J. R. McDonnell, R. G. Reynolds, and D. B. Fogel, editors, *Proc. of the Fourth Annual Conference on Evolutionary Programming*, pages 367–384, Cambridge, MA, 1995. MIT Press.

[66] R. Storn and K. Price. Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, Berkeley, CA, 1995.

[67] R. Storn and K. Price. Differential evolution - a simple and efficient heuristic for global optimisation over continuous spaces. *Journal of Global Optimization*, 11:341–359, 1997.

[68] G. Syswerda. *Schedule optimization using genetic algorithms*, chapter 21, pages 332–349. Van Nostrand Reinhold, 1991.

[69] G. Tao and Z. Michalewicz. Inver-over operator for the TSP. In A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature – PPSN V*, pages 803–812, Berlin, 1998. Springer. Lecture Notes in Computer Science 1498.

[70] J. Thompson, D. Higgins, and T. Gibson. Clustal w: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22:4673–4680, 1994.

[71] J. Thompson, F. Plewniak, and O. Poch. Balibase: a benchmark alignment database for the evaluation of multiple alignment programs. *Bioinformatics*, 15:87–88, 1999.

[72] R. Thomsen, G. Fogel, and T. Krink. A clustal alignment improver using evolutionary algorithms. In *Proceedings of the Fourth Congress on Evolutionary Computation (CEC-2002)*, volume 1, pages 121–126, 2002.

[73] R. Thomsen, G. Fogel, and T. Krink. Improvement of clustal-derived sequence alignment with evolutionary algorithms. In *Proceedings of the 2003 Congress on Evolutionary Computation (CEC-2003)*, 2003.

[74] R. Thomsen and T. Krink. Self-adaptive operator scheduling using the religion-based ea. In *Proceedings of Parallel Problem Solving from Nature VII (PPSN-2002)*, pages 214–223. Springer Verlag, 2002.

[75] H.-K. Tsai, J.-M. Yang, and C.-Y. Kao. Solving traveling salesman problems by combining global and local search mechanisms. In D. B. Fogel, M. A. El-Sharkawi, X. Yao, G. Greenwood, H. Iba, P. Marrow, and M. Shackleton, editors, *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, pages 1290–1295. IEEE Press, 2002.

[76] A. Tuson and P. Ross. Cost based operator rate adaption: An investigation. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature – PPSN IV*, pages 461–469, Berlin, 1996. Springer.

[77] L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *Journal of Computational Biology*, 1:337–348, 1994.

[78] D. Whitley, T. Starkweather, and D. Fuquay. Scheduling problems and traveling salesman: The genetic edge recombination operator. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, San Mateo, CA, 1989. Morgan Kaufman.