Parameter Adaptation in Heuristic Search
– A Population-Based Approach –

Mathias Kern

A thesis submitted for the degree of Philosophiae Doctor

Department of Computer Science
University of Essex

2006

Summary

Heuristic search methods have been applied to a wide variety of optimisation problems. A central element of these algorithms' success is the correct choice of values for their control parameters. To tune these settings, the use of specialists' knowledge and experience are often required.

In this thesis, we first formalise the problem of parameter adaptation in heuristic search. Thereafter, we propose an automated mechanism, i.e. a method that reduces the strong dependency on experts, for choosing the best performing algorithm among several heuristic search approaches and optimising its parameters. The novel Multiple Algorithms' Parameter Adaptation Algorithm (MAPAA) is based on Population-Based Incremental Learning, a method that combines the concepts of Competitive Learning and Genetic Algorithms. Addressing specific characteristics of the adaptation scenario, we extend the basic approach to deal with more versatile search spaces and to run successfully for small populations and few generational cycles. All newly introduced techniques are analysed in detail, and the efficiency and robustness of the MAPAA is studied and proven in several applications.

Acknowledgements

I would like to express my deepest gratitude to my supervisor Professor Edward Tsang. He has been a remarkable mentor. Only his guidance, encouragement and seemingly infinite patience have made my research work possible.

My sincere thanks also go Dr John Ford, Dr Paul Scott, Dr Qingfu Zhang and many other people within the computer science department for their inspiration and help.

Furthermore, I am heavily indebted to Dr Nader Azarmi, Dr Chris Voudouris and Dr Raphael Dorne from BT's Intelligent Systems Research Center for their manifold support.

Last, but by no means least, I would like to thank my friends, especially Tim, my parents, my sister Ati and my partner Claudia. You all made this challenging time so much easier and so much more fun.

This research project was partially funded by EPSRC and BT (case studentship ML843777 / CT320826).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Heuristic search methods form an important class of algorithms for tackling difficult optimisation problems. The efficiency of these widely used techniques often depends on the correct choice of values for their control parameters. Relying on expert knowledge and experience, such parameters are usually set by hand so far. This thesis will present a population-based approach, the Multiple Algorithms' Parameter Adaptation Algorithm, for the automated tuning of parameters in heuristic search algorithms.

In the introductory chapter, we briefly explain the concepts of optimisation problems and heuristic search. Thereafter, the Parameter Adaptation Problem is introduced and defined, and its characteristics are discussed. The chapter concludes with an overview of this thesis.

1.1 Optimisation Problems

Optimisation tasks are encountered in many areas of human life. Finding the shortest way to and from work, arranging furniture effectively, or looking for the best offers in a supermarket are typical examples in our everyday lives. Although these tasks may look basic at first, this is only due to the small problem sizes usually encountered. Larger problems require a dramatically increased effort. In the business world, many decision processes constitute or include optimisation. Maximising profit in the stock market, applying labour in a factory efficiently, or developing reliable and convenient timetables in a transport company – all these are very difficult optimisation tasks.

Motivated by the need to solve such real life problems, researchers have extensively studied the field of optimisation. According to [Russell and Norvig, 2003], the general optimisation problem can be defined as follows:

Definition 1 (Optimisation Problem)**.** *An optimisation problem* Φ *is the problem of finding the best solution* s^{best} *from the set of all possible solutions, the solution space* $\mathbb{S}$*, according to an objective function* $f : \mathbb{S} \rightarrow \mathbb{R}$*. In case of a minimisation problem, the aim is to find* $s^{best} \in \mathbb{S}$ *such that*

$$\forall s \in \mathbb{S} : \ f(s^{best}) \leq f(s).$$

Analogically, the aim in case of a maximisation problem is to find $s^{best} \in \mathbb{S}$

such that

$$\forall s \in \mathbb{S} : \ f(s^{best}) \geq f(s).$$

Research has focused on a number of abstracted problem classes. The well-known examples given below provide just a small insight into the diversity and complexity of such scenarios:

- Vehicle Routing Problem (VRP): The VRP is the task of managing a fleet of vehicles, based at a number of depots, to serve customers by delivering and picking up goods. Constraints like distance, capacity limits of the lorries, and restrictions on when customers can be served often have to be taken into account. The objective is usually to minimise the distance and/or time the vehicles have to travel. For further details, refer to section 6.3.1.

- Job Shop Scheduling Problem (JSSP): In a JSSP, a number of jobs have to be completed. Each job consists of a sequence of actions which have to be processed in a given order. Each of these actions take a certain time and require the application of resources. One possible objective is to find a valid schedule, i.e. a schedule that meets all order and resource constraints, with minimal overall time for the completion of all jobs. A detailed discussion is given in section 6.4.1.

Most optimisation problems, including both aforementioned examples, belong to the class of nondeterministic polynomial (NP) problems. The class

of polynomial problems, i.e. the class of problems that can be solved by polynomial time algorithms, is called P. Although never proven, the vast majority of scientists believe that P≠NP and thus believe that no polynomial time algorithms exist for NP problems. The following example illustrates the related "combinatorial explosion" problem :

Example 1.1. We consider a basic VRP with one vehicle, one depot, and n customers. The task is to find the shortest lorry route, beginning and ending in the depot, that visits all n clients. There are $n-$factorial such routes. Table 1.1 shows the relationship between the problem size n and the number of all possible solutions $n!$. In addition, the time to generate all routes is given in the third column, assuming 10^9 possibilities can be computed per second. As the table clarifies, it is possible to create all possible solutions

Number of Customers	Number of Routes	Computation Time
1	1	$1.0 \cdot 10^{-9}$ seconds
5	120	$1.2 \cdot 10^{-7}$ seconds
10	3628800	$3.6 \cdot 10^{-3}$ seconds
12	$4.8 \cdot 10^{8}$	0.5 seconds
15	$1.3 \cdot 10^{12}$	21.8 minutes
17	$3.6 \cdot 10^{14}$	4.2 days
20	$2.4 \cdot 10^{18}$	77.1 years
50	$3.0 \cdot 10^{64}$	$9.6 \cdot 10^{47}$ years
100	$9.3 \cdot 10^{157}$	$3.0 \cdot 10^{141}$ years

Table 1.1: Relation between problem size, solution space size and required computation time for a VRP

for smaller VRPs and thus to find the optimal solution through a complete search, but as the number of customers grows the computation time of this approach grows extremely quickly. For example, it takes 77 years to find the

best schedule for only 20 customers.

Although sophisticated search methods like branch-and-bound [Balas and Toth, 1985] and advances in computer speed have helped to push up the boundary of practical computability, the underlying problem of exponential growth, often referred to as the "combinatorial explosion", remains for NP problems. It is impossible to apply complete solution algorithms, i.e. methods that guarantee optimality, for larger problem sizes as all known solution algorithms for NP problems show non-polynomial complexity. Rather one has to apply approximation methods, which are the subject of this thesis.

1.2 Heuristic Search

One important class of algorithms to tackle optimisation problems, as described in the last section, are heuristic search methods. The general concept can be defined as follows:

Definition 2 (Heuristic Search Algorithm)**.** *A search algorithm A for tackling optimisation problems that applies a heuristic to search through promising solutions in order to find a good solution is called heuristic search algorithm.*

As the definition points out, a heuristic search algorithm does not system-

atically search through the entire set of possible solutions. Rather a selection mechanism, the heuristic, is applied to choose only certain, apparently promising, solutions. This restriction allows the bypass of the "combinatorial explosion" problem. Unfortunately, a solution produced by a heuristic search algorithm is not guaranteed to be an optimal one. Heuristic search methods have to sacrifice completeness for the benefit of practical usability. Some of the best-known heuristic search methods are Genetic Algorithms [Holland, 1975], Tabu Search [Glover, 1989] and Simulated Annealing [Kirkpatrick et al., 1983, Ingber, 1989].

The central component of every heuristic search method is its heuristic. This strategy allows the incorporation of additional knowledge. The search becomes informed: it uses heuristics to determine what solutions to look for and what solutions to ignore. Therefore, the quality of the heuristics is crucial for the success of the search.

1.3 The Parameter Setting Problem

In most heuristic search methods, the behaviour of the heuristics can be modified via a number of parameters. As these parameters determine the exact functioning of the whole algorithm, the right choice of values for these parameters is often central for a good search performance. This choice is non-trivial and usually requires expert knowledge and experience of the users. This thesis proposes an automated mechanism for this task, the Multiple

Algorithms' Parameter Adaptation Algorithm, which will be introduced in the following two chapters.

First of all, we have to formalise the problem of choosing good parameter values for heuristic search methods based on the definitions of optimisation problems and heuristic search as previously introduced. In this section, we define the Parameter Setting Problem. We begin by providing definitions for parameter domains, the parameter space, and the parameter settings.

Definition 3 (Parameter Domain)**.** *$\mathcal{P}$ is a parameter of the heuristic search algorithm A. The set $\mathbb{D}$ of all possible values for $\mathcal{P}$ is called the domain of this parameter.*

Definition 4 (Parameter Space)**.** *The heuristic search algorithm A has n parameters $\mathcal{P}_1, \ldots, \mathcal{P}_n$ with the domains $\mathbb{D}_1, \ldots, \mathbb{D}_n$. The cross product $\mathbb{P} = \mathbb{D}_1 \times \ldots \times \mathbb{D}_n$ is called the parameter space of A.*

Definition 5 (Parameter Setting)**.** *A point π from the parameter space $\mathbb{P}$ of a heuristic search algorithm A is called a parameter setting for A. Such a point is a vector $(v_1, \ldots, v_n)$ where v_i is a member of the domain of the i-th parameter of A, $i = 1, \ldots, n$.*

With $A(\pi, \Phi)$ we will denote, from now on, the solution $\vec{s} \in \mathbb{S}$ produced by the algorithm A with its parameters set according to π when applied to the optimisation problem Φ. The term $f(A(\pi, \Phi))$ describes therefore the objective function value of the solution produced by A for Φ when using the

parameter setting π.

The parameter setting defined above has central importance for this thesis. It represents a complete assignment of specific values to the parameters of a heuristic search algorithm. The behaviour and thus the performance of a heuristic search can and usually does depend strongly on this specific assignment. The task of finding a good, hopefully optimal parameter setting among the potentially high number of such settings, i.e. from the parameter space, is thus essential.

Even basic heuristic search methods can possess a number of parameters which make the tuning of the algorithm difficult. Examples are given and discussed in chapter 6. In practise, hybrid methods, i.e. methods that combine ideas from several optimisation techniques, are often used. Such hybrid algorithms often have even more parameters. The resulting parameter spaces are thus potentially very large. Therefore, the following Parameter Setting Problem can be very difficult:

Definition 6 (Parameter Setting Problem (PSP))**.** *A heuristic search algorithm A with the parameter space $\mathbb{P}$ is applied to solve k optimisation problems $\vec{\Phi} = (\Phi_1, \ldots, \Phi_k)$ with the objective function f to minimise. The Parameter Setting Problem $PSP(A, f, \vec{\Phi})$ is the task of finding the optimal parameter setting $\pi^{best} \in \mathbb{P}$ such that*

$$\forall \pi \in \mathbb{P} : \ \frac{1}{k} \sum_{i=1}^{k} f(A(\pi^{best}, \Phi_i)) \leq \frac{1}{k} \sum_{i=1}^{k} f(A(\pi, \Phi_i)).$$

Naturally, the PSP itself is NP by nature, and only approximations can realistically be found.

1.4 The Parameter Adaptation Problem

In the Parameter Setting Problem, all optimisation problems are known. In practise, however, one often has to adapt the parameters of a heuristic search algorithm without knowing the exact optimisation problems it will get applied to. Choices can only be based on results and experiences from the application of the search to past problems. The next definition summarises this task:

Definition 7 (Parameter Adaptation Problem (PAP))**.** *A heuristic search algorithm A is applied to solve optimisation problems with the objective function f. The Parameter Adaptation Problem $PAP(A, f, \vec{\Phi}, \vec{\Phi}')$ is the task of finding the optimal parameter setting of A for a sequence of optimisation problems $\vec{\Phi}' = (\Phi'_1, \ldots, \Phi'_{k'})$ based only on information gathered by applying A to the optimisation problems $\vec{\Phi} = (\Phi_1, \ldots, \Phi_k)$. $\vec{\Phi}$ is called the vector of sample problems, $\vec{\Phi}'$ is named the vector of out-of-sample problems.*

To solve the problem defined above, one cannot apply the heuristic search to the optimisation problems from the out-of-sample vector. The only knowledge available is the knowledge about results from the problems of the sample vector. Only after one has found a solution is it possible to study its quality

and hopefully verify its efficiency by applying the heuristic search with the found parameter setting to the problems from the out-of-sample vector.

If the sample and out-of-sample problems are very different, then no method can find a good setting for $\vec{\Phi}'$. Rather we are interested in applications where $\vec{\Phi}$ provides a reasonable good sample of $\vec{\Phi}'$, so that it is possible to draw conclusions: if a parameter setting works well for the sample problems, then it should also deliver good results for the out-of-sample vector.

The PAP involves just one heuristic search algorithm. But often one has a number of heuristic searches available and wants to choose the best one along with its optimal choice of parameter values. The next definition discusses this task.

Definition 8 (Multiple Algorithms' Parameter Adaptation Problem (MAPAP))**.** *The m heuristic search algorithms $\vec{A} = (A_1, \ldots, A_m)$ are applied to solve optimisation problems with the objective function f to minimise. The Multiple Algorithms' Parameter Adaptation Problem $MAPAP$ $(\vec{A}, f, \vec{\Phi}, \vec{\Phi}')$ is the task of finding the heuristic search algorithm A^{best} and the corresponding optimal parameter setting π^{best} for the vector of out-of-sample problems $\vec{\Phi}' = (\Phi'_1, \ldots, \Phi'_{k'})$ such that the average*

$$\frac{1}{k'} \sum_{i=1}^{k'} f(A^{best}(\pi^{best}, \Phi'_i))$$

is minimal among the m algorithms A_j and their optimal parameter settings π_j $(j = 1, \ldots, m)$. In analogy to the PAP, this task must be based only on

information gathered by applying the heuristic search algorithms to the vector of sample problems $\vec{\Phi} = (\Phi_1, \ldots, \Phi_k)$.

With regard to complexity, a MAPAP with m algorithms can easily be divided into m single PAPs. Therefore its complexity is constricted by m times the complexity of the hardest PAP, which yields, as m is a constant, the same complexity as this PAP. In other words, $O(MAPAP)$ is determined by the hardest of the corresponding $O(PAP)$.

1.5 Characteristics of the Parameter Adaptation Problem

The Parameter Adaptation Problem shows the following characteristics that make it difficult to solve:

C1 - Black box: Both the PAP and the MAPAP address heuristic search methods as black boxes. Although the number of parameters and the respective domains are known, no additional knowledge about the heuristic search algorithm, its function, or the meaning and the interaction of its parameters are assumed. While experts in the field of heuristic search can use their experience to quickly decide which parameters are critical with regard to performance, and can narrow down parameter domains to promising values only, our problem for-

mulation intentionally excludes such extra information. On one hand, this makes the search for good parameters choices less informed and as such harder and more difficult. But on the other hand, the user is not required to know how to tune the parameters. As experts in heuristic search are a scarce resource, an algorithm for the automated solving of the (MA)PAP would be helpful and valuable as it requires the user to have just basic knowledge.

C2 - Limited computation time: Solution methods for the Parameter Adaptation Problem and its variations need to evaluate the quality of different parameter settings. To do so, heuristic search algorithms which use these settings get applied to optimisation problems. Such operations can be and usually are computationally expensive. The number of heuristic search operations that can be performed is thus limited. This is particularly true in case of a dynamic scenario: the time span in which a parameter setting can be optimised before the heuristic search method gets applied to the next problem can be very short.

C3 - Noise in the results: Heuristic search methods usually include a random element. The significance of the result of applying a heuristic search algorithm with a specific parameter setting to a single problem is limited. Rather multiple applications of the search method to different problems are required in order to be able to reliably evaluate the quality of a chosen parameter setting.

C4 - Unknown nature of the out-of-sample problems: As required

in the definition of the Parameter Adaptation Problem, only information gathered by applying the heuristic search to the sample problems can be used for finding good parameter settings for the out-of-sample problems. That means that the search is based on information about $\vec{\Phi}$ only, but the success of the search is judged based on information about $\vec{\Phi}'$. Even with known relations, similarities, and dissimilarities between the two data sets, one has to deal with a level of uncertainty. Section 2.4 discusses this characteristic in the context of the No Free Lunch Theorem.

1.6 Overview of the Thesis

Having introduced the problem of parameter adaptation in heuristic search in this chapter, we look at a solution method for this task in the remainder of the thesis. This work is structured as follows:

- Chapter 2 is a review of existing parameter adaptation approaches and discusses the implications of the No Free Lunch Theorem.
- In chapter 3, we study Population-Based Incremental Learning as an optimisation approach that successfully combines the concepts of Genetic Algorithms and Competitive Learning.
- Thereafter, we propose a technique for tackling the Multiple Algorithms' Parameter Adaptation Problem. This method, which we re-

fer to as the Multiple Algorithms' Parameter Adaptation Algorithm, is an extended and specialised version of the basic population-based approach.

- The aim of chapter 5 is the experimental examination and analysis of the newly introduced algorithm.

- The efficiency and robustness of the Multiple Algorithms' Parameter Adaptation Algorithm is tested and demonstrated in chapter 6. It presents the results of applying our optimisation technique to various heuristic search scenarios.

- The final chapter summarises the central aspects of this thesis, discusses the main conclusions and looks at future research directions.

Chapter 2

Existing Approaches for Parameter Tuning and the No Free Lunch Theorem

In the second chapter, we review existing optimisation approaches that can be used for the tuning of parameters. We discuss two general optimisation approaches: the alternating variable search and a method evolved from the Design of Experiments methodology. Furthermore, we choose Genetic Algorithms as a well-known example of heuristic search and provide an overview of various parameter optimisation approaches for this class of algorithms. The chapter concludes with a discussion of the No Free Lunch Theorem and its implications to the Parameter Adaptation Problem.

2.1 Alternating Variable Search

Alternating variable search [Korel, 1990, Torczon, 1997], also known as alternating directions, axial relaxation and local variation, is a general approach to parameter adaptation/optimisation. One of its first applications on a digital computer is reported by [Davidon, 1991]. He describes how this search method was used to determine values of theoretical parameters that best fit experimental data:

> "They varied one theoretical parameter at a time by steps of the same magnitude, and when no such increase or decrease in any one parameter further improved the fit to the experimental data, they halved the step size and repeated the process ..."

In general terms, this simple local search method aims to optimise a vector of variables according to a given objective function. The initial vector of values is chosen at random. Thereafter, each single input variable, i.e. each individual element of the vector, is tested in the exploratory phase. One bye one, variables are probed by increasing or decreasing their values by a certain amount, the step size. If either step leads to an improved objective score, the value of the corresponding variable is updated and the search continues with this new value. This means that the neighbourhoods of all variables are successively probed. If a complete iteration of probing all variables leads to an objective score improvement, then the exploratory process is repeated. However, if such a complete cycle does not yield an improvement, then the

step size is reduced, e.g. halved, and the exploration proceeds with the modified step length. The search procedure continues until a termination criterion is met, e.g. until the step length falls below a specified threshold.

Several variations of the alternating variable search are possible. While the standard algorithm uses the same step length for all variables, other realisations employ different, variable-specific step sizes. Another fundamental aspect of this optimisation routine is the order in which the variables are considered. The basic procedure uses a fixed variable order while others variants chose the variable sequence randomly for each exploratory step.

2.2 Design of Experiments

Design of Experiments (DOE) is a general, systematic approach for planning and conducting experimental work. The main aim is to establish how a number of input variables to a system or process affect its output. [Montgomery, 2005] identifies the following seven steps for designing experiments:

1. *Problem statement*: This statement clearly describes the problem scenario and lists all objectives of the experiment. The three main types of objectives are screening, optimisation and testing for robustness.

2. *Selection of the response variable*: The response variable is the output of the system or process which the experimenter wants to observe.

3. *Choice of factors, ranges and levels*: Together with the response variable, the experimenter has to define the input variables or factors that may influence the performance of the system or process. After selecting which of these factors will be studied in the experiment, the ranges over which the inputs will be varied have to be chosen. From these ranges, exact levels have to be picked at which experiments will be conducted.

4. *Choice of experimental design*: The type of problem, the experimental objectives and additional considerations like available sample sizes influence the exact choice of the experimental design, i.e. how to conduct the experiments in a structured and effective way to achieve the objectives. Such an experimental plan determines how to set and modify the test factors in each experimental run. Various different types of experimental designs of varying complexity and difficulty exist. [Montgomery, 2005, Wu and Hamada, 2000, Cox and Reid, 2000] all describe and analyse manifold design approaches in great detail.

5. *Performing the experiment*: This stage involves the execution and careful monitoring of the experiments.

6. *Statistical analysis of the experimental data*: The data gained during the experimentation phase are analysed with the help of statistical methods. These methods "provide guidelines as to the reliability and validity of the results."

7. *Conclusions*: After analysing the experimental data, conclusions are drawn and potential recommendations are made. This step often in-

volves graphical representations of the main results.

This universal DOE methodology can be used as a starting point for the development and realisation of an experimental setup for tuning the parameters of a heuristic search method. The problem statement describes the challenge of choosing well-performing values for the parameters of this search method. Furthermore, this initial statement names the class of optimisation problems to which the heuristic search is applied.

The response variable, i.e. the monitored output, is the objective score of the heuristic search method. The performance of the heuristic search system and thus the objective is potentially influenced by a number of search parameters. These parameters form the factors that we wish to vary in the experiment.

One of the most common experimental designs for such optimisation scenarios with multiple input variables is the application of the method of steepest descent[1] on a response surface. The experiments are designed in a way that the functional relationship between the inputs and the output, i.e. the response surface, can be approximated with a low-order polynomial. The method of steepest descent can then be used "for moving sequentially in the direction of the maximum [decrease] in the response". The experiments are performed along the path of steepest descent until no further response decreases are monitored. The procedure can be continued by establishing a new model of the response surface and determining a new steepest descent path.

[1]We consider here only minimisation problems.

The rich mathematical theory behind the DOE approach analyses aspects such as, for instance, how to vary input variables simultaneously to maximise the information gained while minimising the experimental resources required, how to detect interdependencies between inputs or how to deal with noise in the observed output. For further information on these aspects and a thorough discussion of various experimental designs, [Montgomery, 2005] is strongly recommended.

2.3 Parameter Adaptation in Genetic Algorithms

Among heuristic search algorithms, Genetic Algorithms (GAs) in particular have attracted a significant amount of research interest with regard to the question of how to set and tune their parameters. While the motivation behind and the concept of GAs will be discussed in more depth in chapter 3, this section provides an overview of existing approaches for determining good values for the parameters of GAs.

Both [da Graca Lobo, 2000] and [Eiben et al., 1999] present surveys of research into how the various parameters of GAs affect their performance. The former classifies this research work as either empirical studies, facetwise theoretical studies or parameter adaptation techniques, and our remarks follow his overview.

Empirical studies subject GAs to performance test through the variation of their parameter values. A well-known exponent is the work conducted by [De Jong, 1975] in which the influence of four parameters, namely population size, generation gap, crossover and mutation probability, is studied. Using five different function optimisation scenarios, De Jong systematically varies these four parameters, analyses the results and thus establishes guiding principles for well-performing, robust parameter choices. [Schaffer et al., 1989] build on this work, extend it to ten test functions and present much more rigorous and exhaustive experimental work. The authors give an update on the proposed standard GA parameter setting but also recognise their uncertainty about whether these results remain valid beyond their test scenarios. Common to both these studies is that they systematically examine the effects of different parameter combinations on the performance of a standard GA.

Facetwise theoretical studies, the second category, analyse GAs by analysing one or two of their parameters in isolation while purposely ignoring all other parameters. Examples include the theoretical study and comparison of different selection schemes ([Goldberg et al., 1992]), the theoretical analysis of the mutation mechanism and the mutation probability ([Muehlenbein, 1992, Baeck, 1993]), and the theoretical investigation of population sizes in GAs ([Goldberg, 1989, Harik et al., 1999]).

[da Graca Lobo, 2000] sees parameter adaptation approaches as techniques that adapt GA parameters as the search progresses. This third category can be further subdivided into centralised methods, decentralised methods and

meta-GAs.

Centralised methods have a central learning component, e.g. a set of predefined learning rules, which is responsible for the adaptation of the GA parameters. [Cavicchio, 1970], as an example, describes an adaptation mechanism that tunes mutation and crossover rates of GAs over time. This parameter adjustment scheme leads to significance performance gains in the experiments reported. [Davis, 1989] and [Julstrom, 1995] study similar approaches which use reward and penalty systems to adapt the rates of the genetic operators. [Smith and Smuda, 1995] look at a different aspect of GAs, the population size. Their algorithm automatically adjusts the population size as the search progresses.

Decentralised methods for parameter adaptation in GAs, on the other hand, do not employ central learning mechanisms but rather encode the parameters which have to be adjusted in the individuals of the population themselves. This idea was already proposed by [Bagley, 1967]. Although such self-adaptation mechanisms are mainly discussed in the context of evolution strategies, [Smith and Fogarty, 1996] suggest that these techniques could be also successfully employed in the GA domain.

Meta-GAs are (higher level) GAs that are applied to adapt and tune the parameters of other (lower level) GAs. In other words, meta-GAs are optimisation routines which adjust the parameter settings of other optimisation algorithms. The first to suggest this approach was [Weinberg, 1970]. [Mercer and Sampson, 1978] report significant performance gains with their

realisation of such a meta-GA. A further exponent of this class of parameter adaptation methods is the algorithm described by [Grefenstette, 1986]. He uses a meta-GA to tune six parameters of a GA and compares his findings with the results presented by [De Jong, 1975]. More recent studies include [Tongchim and Chongstitvatana, 2002] who use a coarse-grained, parallel GA as their meta-strategy.

This section looked at a selection of techniques proposed for tuning parameters of GAs. This selection can neither be exhaustive nor very detailed in the context of this thesis. For more in-depth discussions of this research work, the interested reader is referred to the original publications or the surveys conducted by [da Graca Lobo, 2000, Eiben et al., 1999].

2.4 The No Free Lunch Theorem

The No Free Lunch Theorem, taking its name from the saying "there ain't no such thing as a free lunch", states [Wolpert and Macready, 1995]:

> "...all algorithms that search for an extremum of a cost function perform exactly the same, when averaged over all possible cost functions. In particular, if algorithm A outperforms algorithm B on some cost functions, then loosely speaking there must exist exactly as many other functions where B outperforms A."

[Wolpert and Macready, 1997] furthermore comment that

> "...for any algorithm, any elevated performance over one class of problems is exactly paid for in performance over another class."

This theorem has profound implications for the task of tuning parameters of heuristic search methods. Firstly, each heuristic search algorithm will perform, on average over all possible problems, as well as any other heuristic search algorithm. Secondly, a heuristic search will perform equally well with all parameter choices over the set of all mathematically possible problems.

This raises the question why so much research effort is put into the development and improvement of heuristic search algorithms, and why it can be beneficial to optimise the parameter choices for such methods. [Ho and Pepyne, 2002] provide the answer: "a general-purpose universal optimisation strategy is theoretically impossible, and the only way one strategy can outperform another is if it is specialized to the specific problem under consideration". In praxis, heuristic search methods are often only applied to very specific classes of optimisation problems, and by thus limiting the considered problem domains it is possible to establish better performing search methods and better parameter settings.

The definition of the (Multiple Algorithm's) Parameter Adaptation Problem in section 1.4 describes a black-box scenario in which parameters of heuristic search methods are adapted using sample optimisation problem instances. The quality of the parameter choices, however, is judged by applying the heuristic search methods to different sets of out-of-sample problem instances. The No Free Lunch Theorem explains that it is not possible to effectively

tune the heuristic search parameters for all potential sets of out-of-sample problems. Only when the sample and the out-of-sample instances are drawn from the same class of problems it is possible to effectively adapt the parameters of the heuristic search method. If this is not the case, performance improvements for the sample data do not necessarily translate into performance gains for the out-of-sample data. As an example, a heuristic search method with optimised parameters for random Vehicle Routing Problems does not necessarily perform equally well for Vehicle Routing Problems with clustered, non-random data. We therefore assume in this thesis that the sample and out-of-sample problem instances are always from the same class of problems. All experiments reported in chapters 5 and 6 satisfy this condition.

2.5 Summary

This chapter presented a selection of existing ideas and techniques for the tuning of parameters. Firstly, the general approach of alternating variable search was introduced and described. Secondly, the methodology of the Design of Experiments was discussed. Based on this structured approach for the design and organisation of experimental work, a possible experimental setup for the problem of parameter adaptation in heuristic search was briefly outlined. In a third step, we chose Genetic Algorithms as an example of heuristic search and reported manifold research efforts with regard to parameter optimisation. These approaches were classified as empirical studies, facetwise theoretical studies or parameter adaptation techniques. Finally, the No Free

Lunch Theorem and its implications for the parameter adaptation scenario were studied. This theorem states that it is theoretically impossible to find a "general-purpose universal optimisation strategy" [Ho and Pepyne, 2002] and that, as a consequence, each optimisation algorithm can outperform other optimisation strategies only for specific problem classes.

Chapter 3

Population-Based Incremental Learning

[Baluja, 1994] describes Population-Based Incremental Learning as "a method for integrating genetic search based function optimisation and competitive learning". The combination of these two concepts make it a successful and versatile optimisation approach. The resolution algorithm for the Multiple Algorithms' Parameter Adaptation Problem, which is presented in the next chapter, is based on a novel, generalised formulation of the Population-Based Incremental Learning algorithm. Before we introduce this new method, which we call Generalised Population-Based Incremental Learning, in the later part of this chapter, basic Population-Based Incremental Learning is discussed in the context of Estimation of Distribution Algorithms, Genetic Algorithms and Competitive Learning.

3.1 Population-Based Incremental Learning

Population-Based Incremental Learning, henceforth referred to as PBIL, is an algorithm for tackling optimisation problems. The method belongs to the class of Estimation of Distribution Algorithms, and it combines ideas both from Genetic Algorithms and Competitive Learning. In this section, we therefore summarise the concept of Estimation of Distribution Algorithms, introduce Genetic Algorithms and Competitive Learning closely following [Baluja, 1994], and describe and discuss the standard PBIL algorithm.

3.1.1 Estimation of Distribution Algorithms

Evolutionary Algorithms are population-based methods for solving optimisation and search problems. Well-known exponents of this algorithm class are Genetic Algorithms, which we look at in section 3.1.2, and Evolutionary Programming. These traditional techniques "maintain and successively improve a collection of potential solutions", and "employ crossover and mutation as variation operators to create the elements of the next generation" [Zhang and Mühlenbein, 2004].

Estimation of Distribution Algorithms (EDAs), a more recent addition to the category of Evolutionary Algorithms, follow a different approach. These methods statistically analyse the population and extract data about its composition. Based on this information, a model about the distribution of promising solutions is built, or, in other words, these algorithms estimate

the probability distribution of promising points in the search space. Instead of applying "traditional crossover or mutation", the constructed distribution model is used for sampling new elements for the next generation. Through this repeated process of generating a new population based on a distribution model, and the subsequent modification of the distribution model, the quality of the maintained population is improved over time.

Many different implementations of EDAs have been proposed. Instances include, among others, Population-Based Incremental Learning [Baluja, 1994], the Bayesian Optimisation Algorithm [Pelikan et al., 1999], the Univariate and Bivariate Marginal Distribution Algorithms [Pelikan and Mühlenbein, 1998, Pelikan and Mühlenbein, 1999], and the Factorised Distribution Algorithm [Mühlenbein and Mahnig, 1999].

3.1.2 Genetic Algorithms

Genetic Algorithms (GAs), pioneered by [Holland, 1975], are a class of optimisation methods that take their motivation from the processes of natural selection and genetic recombination. These principles, highly successful in evolution, can be found in abstracted form in every GA.

In contrast to optimisation methods such as Simulated Annealing [Kirkpatrick et al., 1983, Ingber, 1989] and Tabu Search [Glover, 1989], which are based on the idea of improving a single solution in small modification steps, GAs always maintain a whole group of potential solutions

called a population. Each population member, referred to as an individual or chromosome, is represented as a string of fixed length over a (finite) alphabet. In its basic, and most common, form this alphabet consists of just zeros and ones; hence the chromosomes, in this case, are binary vectors.

The population develops over a number of generations. While individuals of the first generation are created randomly, individuals of subsequent cycles are determined by recombining members of the current population. This recombination process, which usually involves two parental chromosomes, allows the merging of information and characteristics of the parent individuals into a new offspring. In addition, the offspring chromosome may get slightly altered by chance through mutation, i.e. some of the values in its vector representation might get changed randomly. The following example illustrates both a standard recombination operator, one point crossover, and a basic mutation operator.

Example 3.1. In this example, each chromosome is represented as a binary vector of length 8. The following two individuals act as parents:

```
parent 1:   11101111
parent 2:   00100001
```

A one point crossover operator, applied at position 5, combines these parents by taking the first 5 bit values from parent 1 and the remaining 3 bit values from parent two. By chance, a mutation operator might change the final bit

of the resulting offspring:

```
            offspring:   11101001
mutated offspring:   11101000
```

The choice of parents to produce offspring chromosomes for the next generation is neither at random nor arbitrary. Rather the individuals of the current population are in competition with each other. This competition, the modelling of the principle of natural selection, is based on the objective function evaluations of the individuals called their fitness. The parent selection is a probabilistic process, i.e. individuals with higher fitness values are more likely to be chosen. Therefore, individuals with a higher fitness are more likely to pass their characteristics on to offsprings in the next generation. Of course, such an offspring is by no means guaranteed to be fitter than its parents. But the recombination mechanism permanently forces the population to explore new combinations and variations. Together with selective pressure, this leads to the evolution of the population towards fitter individuals.

GAs have been successfully applied in many different problem scenarios. Examples include Vehicle Routing [Machado et al., 2002], Job Shop Scheduling [Ono et al., 1996] and Circuit Layout [Mazumder and Rudnick, 1998].

3.1.3 Competitive Learning

The most well-known application of Competitive Learning (CL) is within the field of artificial neural networks [Zell, 1994]. [Kohonen, 1995] lists, among others, Self-Organising Maps and Learning Vector Quantization as examples of CL networks. In such networks, all input cells receive the same input signal. Through interaction within the neural network, the output cells compete for activation. Only the neuron with the highest output signal is allowed to fire, and is said to represent the current input. Through learning, i.e. modifying the interaction within the network, output cells are trained to represent different clusters of the input signal space. The learning process is called Competitive Learning.

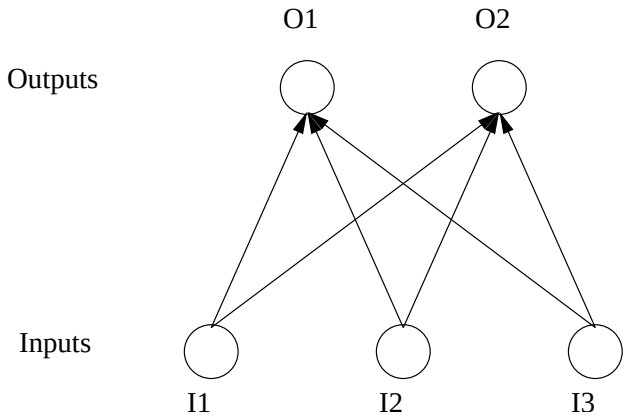

Figure 3.1: A simple Competitive Learning network

In figure 3.1, a simple CL network with three input neurons and two output neurons is given. Each of the input cells is connected to each of the output cells, with a weight attached to each of these connections. The input units can only receive binary signals $I_i \in \{0, 1\}$; hence the input space is $\{0, 1\}^3$.

The activation O_i of output unit i can be computed as

$$O_i = \sum_j w_{ij} \cdot I_i$$

where w_{ij} is the weight of the connection between input neuron i and output neuron j. While the weights are initially determined randomly, they are modified during learning. The winning neuron, i.e. the output neuron with the highest activation, is allowed to modify the weights of its connections in order to represent the current input better:

$$\Delta w_{ij} = \epsilon \cdot (I_i - w_{ij}), \tag{3.1}$$

with ϵ being the learning rate. The learning phase consists of presenting different input signals to the network, and subsequently modifying the weights according to the described learning rule. After this learning process, the weights of an output neuron can be seen as a prototype vector for a cluster represented by this neuron. The network has learned to cluster the input space.

3.1.4 Population-Based Incremental Learning

The PBIL algorithm, as it is presented next, assumes an optimisation problem with the solution space $\mathbb{S} = \{0,1\}^L$. Hence each candidate solution is a binary vector $\vec{s} \in \mathbb{S}$ of length L. The task is to find the solution $\vec{s}^{best}$ with the optimal objective evaluation $f(\vec{s}^{best})$. If not otherwise stated, a

minimisation scenario is assumed in this thesis.

In analogy to GAs, PBIL works with a population of candidate solutions which evolve over a number of generations. But in contrast to GAs, members of the current population are not used in a recombination scheme, rather the "algorithm attempts to create a probability vector from which samples can be drawn to produce the next generation's population. ... In a manner similar to the training of a competitive learning network, the values in the probability vector are shifted towards representing those in high evaluation vectors." [Baluja, 1994] This probability vector $\vec{P} = (P_1, \ldots, P_L)$ holds the probabilities P_i of samples having the bit value one at position i.

In figure 3.2, the pseudo code of the PBIL algorithm is given, followed by the description of its working principle.

PBIL Algorithm

1 $\vec{s}^{best} = \emptyset$
2 FOR $l = 1, \ldots, L$ DO $P_l = 0.5$
3 FOR $g = 1, \ldots, G$ DO
4 FOR $i = 1, \ldots, M$ DO
5 $\vec{s}_i = sampling(\vec{P})$
6 $e_i = f(\vec{s}_i)$
7 $min = argmin_{i=1}^{M} e_i$
8 FOR $l = 1, \ldots, L$ DO
9 $P_l = P_l \cdot (1 - \epsilon_1) + \vec{s}_{min,l} \cdot \epsilon_1$
10 IF $random((0, 1]) < \mu$ THEN
11 $P_l = P_l \cdot (1 - \epsilon_2) + random(\{0, 1\}) \cdot \epsilon_2$
12 $\vec{s}^{best} = minimum(\vec{s}^{best}, \vec{s}_{min})$
13 RETURN $\vec{s}^{best}$

Figure 3.2: The Population-Based Incremental Learning algorithm

In the initialisation phase, the best found solution $\vec{s}^{best}$ is set to $\emptyset$ (line 1)

as no such solution is known yet. Furthermore, all probabilities in $\vec{P}$ are set to 0.5 (line 2) because no information about values of solution vectors with good evaluations are available at this point.

The second phase of the PBIL algorithm consists of the loop over G generational cycles (lines 3 to 12). The first step within this main loop is the sampling of M new candidate solutions $\vec{s}_i$ which get evaluated as e_i, $i = 1, \ldots, M$ (lines 4 to 6). After determining the index min of the best among the M evaluations (line 7), learning takes place. All L probabilities in the probability vector $\vec{P}$ are shifted towards more accurately representing the bit values in the best solution $\vec{s}_{min}$ of the current population (line 9). The learning formula

$$P_l = P_l \cdot (1 - \epsilon_1) + \vec{s}_{min,l} \cdot \epsilon_1,$$

where ϵ_1 is the learning rate and $\vec{s}_{min,l}$ the l-th bit value of $\vec{s}_{min}$, can be rewritten as

$$\Delta P_l = \epsilon_1 \cdot (\vec{s}_{min,l} - P_l). \tag{3.2}$$

This equation resembles the learning rule 3.1 for CL networks. If $\vec{s}_{min}$ has a one at position l, then P_l gets increased by this rule, otherwise it gets decreased. In a second updating step, all probabilities in $\vec{P}$ get "mutated" with mutation probability μ (lines 10 and 11). That means each of these probabilities may get moved by chance slightly towards either 0 or 1 by amount ϵ_2. The final step of the main loop is the replacement of $\vec{s}^{best}$ if the current population includes a solution with a better (smaller) evaluation (line 12).

The PBIL algorithm finishes by returning the best encountered solution $\vec{s}^{best}$ (line 13). A list summarising the complete notation used in the algorithm can be found in appendix A.3.

The main idea of PBIL is to determine the distribution of bit values in solutions with good evaluations. As this distribution is used for the sampling of each new generation, the search is concentrated on areas of the solution space which previously led to promising solutions. It is hoped that these regions yield even better solutions. After each generation, the search is shifted towards the best solution of this generation. This leads to a dynamic system of permanently (re-)focusing the search while still, probabilistically allowing the exploration of the entire search space.

3.2 Further Developments of Population-Based Incremental Learning

3.2.1 Continuous Search Spaces

The basic PBIL algorithm is designed for binary search spaces. Two extensions have been proposed to extend its applicability to continuous domains.

[Servet et al., 1997] suggest to use a single probability P to determine values from an interval $[a, b]$. This probability is interpreted as $P(X > \frac{a+b}{2})$. This means a value is uniformly chosen from $(\frac{a+b}{2}, b]$ with probability P, and from

$[a, \frac{a+b}{2}]$ with probability $1 - P$. The learning rule for updating this likelihood is the same as in the boolean case: if the best individual takes a value from $(\frac{a+b}{2}, b]$, then P is moved towards 1, otherwise its is moved towards 0. If, after some learning, P is specific enough, i.e. $P < 0.1$ or $P > 0.9$, the search is re-focused on the corresponding sub-interval $[a, \frac{a+b}{2}]$ or $(\frac{a+b}{2}, b]$, respectively, and P is newly initialised to 0.5. This mechanism allows the search to focus on sub-intervals that contain the best individual in most cases. However, this approach has serious limitations. Once a sub-interval has been discarded, no further values can be drawn from this region. Furthermore, if both sub-intervals contain promising values to the same degree, re-focusing is not possible and the search remains purely random over the whole interval.

[Sebag and Ducoulombier, 1998] discuss how to use a normal distribution $N(X, \sigma)$ over an interval $[a, b]$ for modelling the distribution of good values. They propose mechanisms for updating the mean X and the variance σ through incremental learning. The reported experimental results are not outstanding. Although the assumption of a normal distribution of good values over a range $[a, b]$ can be justified in many cases, this is not necessarily true in general. Therefore, the applicability of this concept is limited.

3.2.2 Multiple Populations

Using multiple populations rather than a single population has been shown to be an effective mechanism for Evolutionary Algorithms in dynamic optimisation problems [Branke et al., 2000]. [Yang and Yao, 2003] take this sug-

gestion and propose two novel PBIL variants.

Parallel Population-Based Incremental Learning, in its simplest form, is a modification of PBIL which uses two probability vectors $\vec{P}_1$ and $\vec{P}_2$ instead of just one. While all values in the first vector are initialised to 0.5, the values in the second vector are randomly initialised. Both $\vec{P}_1$ and $\vec{P}_2$ "are sampled and updated independently". Initially, half the population in each cycle is sampled from each probability vector. These sample sizes are then adapted iteratively to allow the better performing of the two vectors to generate more samples in latter cycles.

The second enhancement of PBIL discussed is Dual Population-Based Incremental Learning. This variant uses two dual probability vectors $\vec{P} = (P_1, \ldots, P_L)$ and $\vec{P}' = (1 - P_1, \ldots, 1 - P_L)$. These vectors are "symmetric with respect to the central point" $(0.5, \ldots, 0.5)$ of the search space. In contrast to Parallel PBIL where both probability vectors are updated independently during the search process, Dual PBIL applies the default PBIL learning mechanism only to $\vec{P}$ as $\vec{P}'$ changes automatically to maintain the dualism. Both variants agree in all other aspects. The introduction of the dualism principle into PBIL is motivated with an increased diversity of the samples and a higher robustness against a changing environment.

[Yang and Yao, 2003] compare the performance of a standard GA, PBIL, Parallel PBIL and Dual PBIL for stationary knapsack problems and series of dynamic knapsack problems. In the stationary case, all three PBIL variants outperform a standard GA with PBIL showing the best performance. The

authors conclude that "introducing extra probability vector into PBIL may not be beneficial" for stationary problems. In dynamic scenarios with significant changes, however, Dual PBIL shows the best performance and often achieves high performance improvements compared to PBIL. The dualism of the probability vectors leads to an increased adaptability.

3.2.3 Learning Rate Correctives

The learning rate is a fundamental parameter of the PBIL algorithm. As [Shapiro, 2002, Shapiro, 2003] illustrates in two example problem scenarios, it must be sensitively scaled with regard to the problem size. This sensitivity of the learning rate is caused by the drift phenomenon: even on a flat landscape, the amount of movement of the probabilities[1] P_i away from 0.5, the expected value, is larger than the movement towards 0.5. Therefore, PBIL gets attracted to corners of the corresponding hypercube. [Shapiro, 2002] shows that the learning rate must be chosen to be "very small to prevent premature convergence" and proposes two new mechanisms which help to overcome this sensitivity.

Detailed Balance Population-Based Incremental Learning introduces detailed balance in PBIL, a condition central to the convergence of variables to a desired equilibrium distribution in Markov processes[2]. The modified algorithm uses a rejection approach in its learning mechanism. During the learning

[1] Probability vector $\vec{P} = (P_1, \ldots, P_L)$

[2] The interested reader is referred to [Neal, 1993] for a detailed review of such processes and related Markov chain Monte Carlo methods.

process, i.e. during the modification of probabilities P_i from the probability vector $\vec{P}$, moves away from hypercube corners are always accepted while moves towards such corners can be rejected. Shapiro's results for two case studies indicate that Detailed Balance PBIL can successfully counter the problems caused by drift and thus the algorithm works well for a much wider range of learning rates.

The second proposed corrective to control drift is probabilistic mutation. A "mutation-like parameter m" is introduced into the PBIL learning rule (line 9 in figure 3.2):

$$P_l^{new} = \frac{P_l + \epsilon_1(\vec{s}_{min,l} - P_l) + m}{1 + 2m}.$$

This leads to a mechanism analogous to mutation in classical GAs. [Shapiro, 2003] discusses conditions for well-performing choices of mutation rate m and learning rate ϵ_1 and illustrates the effectiveness of probabilistic mutation for needle-in-a-haystack problems.

3.3 Applications of Population-Based Incremental Learning

PBIL has been successfully applied to a wide range of problems, some of which are listed below:

- In his original work, [Baluja, 1994] applies PBIL to Job Shop Schedul-

ing, Traveling Salesman, Bin Packing, Standard Numerical Optimisation and Strong Local Optima Problems. He concludes that the "simple population-based incremental learner performs better than a standard genetic algorithm in the problems empirically compared".

- [Sukthankar, 1997] uses PBIL to tune the parameters of a distributed reasoning system for an intelligent vehicle. The experiments presented demonstrate the "ability of evolutionary algorithms to automatically configure a collection of these modules".

- [Sałustowicz and Schmidhuber, 1997] introduces Probabilistic Incremental Program Evolution (PIPE). This approach for automatic program synthesis combines PBIL and "tree-coded programs". PBIL is used to stochastically generate better-performing programs.

- [Southey and Karray, 1999] show in simulations how a PBIL variant outperforms an implementation of a genetic algorithm in evolving controllers based on neural networks for robotic agents.

- [Gosling, 2003] studies Simple Supply Chain Model problems scenarios, analyses various middlemen strategies for such problems and discusses the efficiency of PBIL in optimising parameters of these strategies.

3.4 Generalised Population-Based Incremental Learning

The PBIL algorithm in its basic form makes several assumptions about the nature of the handled optimisation problem which obviously narrows its applicability. In this section, we address these restrictions and propose a novel, more general formulation of the method that overcomes these problems, which we call Generalised Population-Based Incremental Learning (GPBIL).

Standard PBIL shows the following restrictions:

1. The algorithm can handle just optimisation problems with binary solution vectors. If one extends the solution space to $\mathbb{N}^L$ or even $\mathbb{R}^L$, one has to heavily modify the application of the probability vector and the corresponding learning mechanism.

2. The algorithm assumes that every candidate solution has exactly one unique evaluation. In case of noisy results or differing results for differing scenarios, which the MAPAP shows as stated in characteristic C3 in section 1.5, one has to introduce mechanisms to allow the accumulation of a number of evaluations. Furthermore, the decision about which of two solutions is actually better becomes more difficult.

3. The algorithm in its most basic form just learns from the best population member. Although [Baluja, 1994] also discusses other learning schemes, including learning from negative examples, various further

learning mechanisms are possible.

Regarding the first point, we assume the solution space

$$\mathbb{S} = \mathbb{D}_1 \times \ldots \times \mathbb{D}_n \tag{3.3}$$

with $\mathbb{D}_i$ being a subset of the real numbers. $\mathbb{D}_i$ is called the domain for the i-th element of any solution vector. In order to model a distribution of good values over such a domain, we apply the concept of probability distribution functions of real valued random variables to this set[3]. $\mathcal{F}_{\mathbb{S}}$ is the joint probability distribution function over the entire solution space. It is worth mentioning that the probability vector in the basic PBIL algorithm is nothing more than a joint probability distribution over $\{0,1\}^L$. This joint probability distribution can firstly be used to sample points from the solution space, and secondly can be updated and modified through learning. Of course, the learning mechanism becomes much more complex in this case, and can, with regard to restriction three, take many different shapes. As the proposed GPBIL method is formulated in a generic way, the exact nature of the learning scheme is left open. A specific implementation is presented in the next chapter in the context of the MAPAP.

To overcome the problems mentioned due to the second restriction above, each population member can now have an attached history of evaluation

[3]Appendix B provides a brief overview over the mathematical principles of probability distributions.

results. Hence, an individual I can be represented as a tuple[4]

$$I = \langle \vec{s}, \vec{e} \rangle \tag{3.4}$$

with $\vec{s} \in \mathbb{S}$ and $\vec{e} = (e_1, \ldots, e_G)$ a vector of G evaluation results. An individual can thus potentially hold evaluation results for all G generational cycles. If $\mathbb{E}$ represents the space of all possible evaluation results, the space of all possible individuals $\mathbb{I}$ can be written as $\mathbb{I} = \mathbb{S} \times \mathbb{E}^G$. Adding an evaluation value to an individual in generation g is symbolised by

$$\langle \vec{s}, (e_1, \ldots, e_g, \ldots, e_G) \rangle \biguplus_g e'_g = \langle \vec{s}, (e_1, \ldots, e'_g, \ldots, e_G) \rangle. \tag{3.5}$$

In order to allow the accumulation of different evaluation results, individuals must get the chance of multiple evaluations. Therefore, GPBIL makes it possible for individuals to get passed on to subsequent generations. Which individuals will get this survival chance is determined by a selection mechanism.

Having made these introductory remarks, the pseudo code for GPBIL is presented in figure 3.3. A list containing notational comments can be found in appendix A.4.

In common with the standard PBIL method, both the best individual I_{best} and the joint probability distribution $\mathcal{F}_{\mathbb{S}}$ get initialised in the first stage of the algorithm (lines 1 and 2). More specifically, the function *initialisation*, given

[4]The notation $\langle \ldots, \ldots \rangle$ is used to distinguish individuals clearly.

GPBIL Algorithm

1 $I^{best} = \langle \vec{s}_{initial}, \emptyset \rangle$
2 $\mathcal{F}_{\mathbb{S}} = initialisation(\mathbb{S})$
3 $N = M$
4 FOR $g = 1, \ldots, G$ DO
5 FOR $i = M - N + 1, \ldots, M$ DO $I_i = \langle sampling(\mathcal{F}_{\mathbb{S}}), \emptyset \rangle$
6 FOR $i = 1, \ldots, M$ DO $I_i = I_i \biguplus_g evaluation(I_i, g)$
7 $I^{best} = I^{best} \biguplus_g evaluation(I^{best}, g)$
8 $\mathcal{F}_{\mathbb{S}} = learning(\mathcal{F}_{\mathbb{S}}, I^{best}, \vec{I}, g)$
9 $I^{best} = optimum(I^{best}, \vec{I})$
10 $\vec{I} = selection(I^{best}, \vec{I})$
11 $N = M - size(\vec{I})$
12 RETURN I^{best}

Figure 3.3: The Generalised Population-Based Incremental Learning algorithm

the solution space as a parameter, returns the initial probability distribution function over this space. If $\tilde{\mathbb{S}}$ is the space of all possible solution spaces, and $\mathbb{F}$ is the space of all possible probability distribution functions, then one can write

$$initialisation : \tilde{\mathbb{S}} \rightarrow \mathbb{F}. \tag{3.6}$$

In addition, the number of individuals to be sampled N is initially set to the population size M (line 3).

The main part of the algorithm consists again of a loop over G generational cycles (lines 4 to 11). Firstly, N newly sampled individuals get added to the population so that the population size is again M (line 5). To achieve this, the function *sampling* returns a solution $\vec{s} \in \mathbb{S}$

$$sampling : \mathbb{F} \rightarrow \mathbb{S} \tag{3.7}$$

which gets combined with an (initially empty) evaluation history[5].

In a second step, all individuals of the population as well as the best found individual I^{best} get evaluated; the results are added to the respective evaluation histories (lines 6 and 7). The evaluation is performed through the *evaluation* function which, presented with an individual and the current cycle number, returns a value from the evaluation space:

$$evaluation : \mathbb{I} \times \{1, \ldots, G\} \rightarrow \mathbb{E}. \tag{3.8}$$

The joint probability distribution $\mathcal{F}_{\mathbb{S}}$ then gets modified through learning (line 8)

$$learning : \mathbb{F} \times \mathbb{I} \times \mathbb{I}^M \times \{1, \ldots, G\} \rightarrow \mathbb{F}, \tag{3.9}$$

which is based on comparing the most recent performances of the M population members with the performance of the best individual and modifying the probability distribution function accordingly.

Next, a test is performed to determine whether an individual from the population can replace the best found individual I^{best} (line 9):

$$optimum : \mathbb{I} \times \mathbb{I}^M \rightarrow \mathbb{I}. \tag{3.10}$$

The main loop finishes with the selection of population members which are

[5]Note that this is not a function in the strict mathematical sense because the returned results are random but follow the given probability distribution.

passed on to the next cycle (line 10):

$$selection : \mathbb{I} \times \mathbb{I}^M \rightarrow \mathbb{I}^M, \tag{3.11}$$

and the subsequent re-computation of N (line 11).

The GPBIL algorithm terminates returning the best encountered individual I^{best} as the result (line 12).

Note that the functions *initialisation*, *sampling*, *learning*, *optimum*, and *selection* are given without any further specification. Their meaning and task has been described, but no additional details are provided. This allows the algorithm to be formulated in a very broad, generic way.

3.5 Summary

In this chapter, we discussed Population-Based Incremental Learning (PBIL). It was shown how this powerful optimisation strategy combines concepts of Genetic Algorithms and Competitive Learning. Central to its function is the maintenance of a model about the distribution of promising solutions. This model serves as a source for the generation of new candidate solutions that are in competition with each other. The most successful candidates are used as feedback and thus help to improve the distribution model itself. Through a repetition of this process, the method improves the quality of the distribution model over time, and therefore learns to generate better candidate solutions.

However, PBIL has several limitations that narrow its applicability. To overcome these shortcomings, we proposed a novel, generalised formulation that we named Generalised Population-Based Incremental Learning (GPBIL). This new method allows more general solution spaces, more complex evaluations which take noise into account, and provides room for manifold learning schemes. In contrast to PBIL, the generalised version is not a fully specified algorithm as such but rather a generic framework. Several of its components are left unspecified to allow for tailored realisations in different problem scenarios. As this thesis is concerned with the problem of parameter adaptation in heuristic search, an application and complete implementation of GPBIL for this case is presented in the next chapter.

Chapter 4

Multiple Algorithms' Parameter Adaptation

In this chapter, we propose an algorithm based on Generalised Population-Based Incremental Learning for solving the Multiple Algorithms' Parameter Adaptation Problem. In order to define this new method, which we refer to as the Multiple Algorithms' Parameter Adaptation Algorithm (MAPAA), we specify and describe all generic components of the Generalised Population-Based Incremental Learning scheme. However, no empirical analysis is provided here as the next chapter is aimed at the study of the suggested technique through experimentation.

The composition of this chapter is, therefore, as follows. After a brief recapitulation of the Multiple Algorithms' Parameter Adaptation Problem, this

problem is then modelled in a suitable way so that it can be used with the generalised formulation of Population-Based Incremental Learning. The modelling process includes the definition of a solution space, the description of an evaluation space, and a discussion about the employed probability distribution functions. Thereafter, we characterise all the methods for initialising the joint probability distribution, for the sampling and evaluation of individuals, for applying learning processes, and for the tasks of updating the best solution and selecting surviving individuals. Pseudo code is provided for each of these functions. The chapter closes with a summary that lists all parameters and limitations of the proposed algorithm.

4.1 The Problem

Revisiting the definition of the Multiple Algorithms' Parameter Adaptation Problem (MAPAP) from section 1.4, the following information is available to every solution method:

m	number of heuristic search algorithms
$\vec{A} = (A_1, \ldots, A_m)$	vector of heuristic search algorithms
l_i	number of parameters of algorithm A_i
$\mathbb{D}_{i,j}$	domain of j-th parameter of algorithm A_i
$\vec{\Phi} = (\Phi_1, \ldots, \Phi_k)$	vector of k sample optimisation problems
f	objective function of optimisation problems

(4.1)

Given these data, the task is to find the algorithm A^{best} among the m methods, together with its optimal parameter setting π^{best}, that performs best for the out-of-sample problems $\vec{\Phi}' = (\Phi'_1, \ldots, \Phi'_{k'})$. Any findings must only be based on information gathered by applying the heuristic search methods to the sample problems $\vec{\Phi}$; the problems in $\vec{\Phi}'$ are not available during this process.

4.2 The Solution Space

In order to model the MAPAP in a suitable way under Generalised Population-Based Incremental Learning (GPBIL), one has to define an appropriate solution space[1] over which a joint probability distribution can be maintained and updated. A point from this solution space has to provide the information to answer two questions: which of the heuristic search algorithms has been chosen, and what are the selected parameter values for this algorithm. This can be achieved by defining the solution space as

$$\mathbb{S} = \{1, \ldots, m\} \times \mathbb{D}_{1,1} \times \cdots \times \mathbb{D}_{1,l_1} \times \cdots \times \mathbb{D}_{m,1} \times \cdots \times \mathbb{D}_{m,l_m},$$

rewritten as

$$\begin{aligned} &\mathbb{S} = \mathbb{D}_1 \times \cdots \times \mathbb{D}_n \\ &\text{with} \quad \mathbb{D}_1 = \{1, \ldots, m\},\ \mathbb{D}_2 = \mathbb{D}_{1,1},\ \ldots,\ \mathbb{D}_n = \mathbb{D}_{m,l_m}. \end{aligned} \tag{4.2a}$$

[1] We refer here to the solution space of the MAPAP, not the solution space of the optimisation problems the algorithms A_i are supposed to solve.

A solution $\vec{s} \in \mathbb{S}$ is thus a real-valued vector of length n, where

$$n = 1 + \sum_{i=1}^{m} l_i. \tag{4.2b}$$

Such a solution vector holds, in fact, information about parameter choices for all m heuristic search methods.

4.3 The Evaluation Space

As the vector of sample optimisation problems, $\vec{\Phi}$, provides k possible candidate problems, a policy has to be devised in order to decide when to use which optimisation problem. The scheme applied here is to use problem Φ_g for the evaluation of individuals in generation g. This way, new evaluation results are available in every cycle. An individual $I = \langle \vec{s}, \vec{e} \rangle$ encodes m parameter settings for the algorithms A_1 to A_m. There are, potentially, m different objective values with regard to one optimisation problem. Therefore, we define the evaluation space as

$$\mathbb{E} = \mathbb{R}^m. \tag{4.3}$$

A consequence of this equation is that an individual's history is a point in the space $(\mathbb{R}^m)^G$. That means that this memory can hold evaluation results collected for all m heuristic search methods over all G generational cycles. However, not all these results might be necessary, available or computable, so that history entries can be left blank, symbolised by $\emptyset$.

4.4 Notes about Individuals

Each individual I from the population is a pair $\langle \vec{s}, \vec{e} \rangle$ of a candidate solution $\vec{s} = (s_1, \ldots, s_n)$ from the solution space $\mathbb{S}$ and an evaluation history $\vec{e}$ from the space $(\mathbb{E})^G$. Although such an individual holds parameter choices for all m algorithms A_1 to A_m, only the parameter setting for the encoded algorithm, namely the algorithm with the index s_1, is of importance and is ever used. This also limits the evaluation history to the effect that it only stores results about this one algorithm, all other fields are left blank.

The only exception is the special individual $I^{best} = \langle \vec{s}^{\,best}, \vec{e}^{\,best} \rangle$. The m parameter settings,which are encoded by this individual, represent the best settings that have been encountered for the algorithms A_1 to A_m. In addition, it provides the information about the best among these m algorithms which is the method indexed by s_1^{best}. The history $\vec{e}^{\,best}$ contains evaluations achieved by all m algorithms as a result.

The first step of the MAPAA is the initialisation of I^{best} by pairing an initial candidate solution $\vec{s}^{\,initial}$ with an empty history ($\emptyset$) (line 1 in the GPBIL figure 3.3). If advance information about reasonable parameter settings for the heuristic search methods are available than those can be incorporated into $\vec{s}^{\,initial}$. Otherwise, one can choose $\vec{s}^{\,initial}$ as a random point from $\mathbb{S}$.

4.5 The Joint Probability Distribution over the Solution Space

As stated in equation 4.2, the solution space takes the form $\mathbb{S} = \mathbb{D}_1 \times \cdots \times \mathbb{D}_n$. Although GPBIL principally allows any subset of the real numbers for these domains, we limit these sets here to two types: $\mathbb{D}_i$, $i = 1, \ldots, n$, can be either a finite set, or the domain can be an interval of the real numbers. By defining probability distribution functions $\mathcal{F}_{\mathbb{D}_i} : \mathbb{R} \to [0, 1]$ for marginal random variables over the ranges $\mathbb{D}_i$, we define the joint probability distribution $\mathcal{F}_{\mathbb{S}}$ over $\mathbb{S}$ by

$$\mathcal{F}_{\mathbb{S}} = \mathcal{F}_{\mathbb{D}_1} \cdot \ldots \cdot \mathcal{F}_{\mathbb{D}_n}. \tag{4.4}$$

Appendix B.1 briefly summarises the mathematical concepts behind discrete and continous random variables, and the corresponding probability distribution functions(PDFs).

4.5.1 Finite Domain

If domain $\mathbb{D}_i$ is a finite set with u_i elements

$$\mathbb{D}_i = \{d_{i,1}, \ldots, d_{i,u_i}\}, \tag{4.5a}$$

then the probability distribution over this set, according to standard probability theory, is characterised by the u_i probabilities[2]

$$P_{i,1}, \ldots, P_{i,u_i} \tag{4.5b}$$

representing the likelihoods of the corresponding elements to be chosen. That means GPBIL can model the probability distribution function $\mathcal{F}_{\mathbb{D}_i}$ over $\mathbb{D}_i = \{d_{i,1}, \ldots, d_{i,u_i}\}$, which is given as

$$\mathcal{F}_{\mathbb{D}_i}(x) = \sum_{d_{i,j}<x} P_{i,j}, \quad x \in \mathbb{R}, \tag{4.5c}$$

by u_i real numbers.

4.5.2 Interval Domain

In the case where domain $\mathbb{D}_i$ is a real-valued interval

$$\mathbb{D}_i = [a_i, b_i], \tag{4.6a}$$

the probability distribution over $\mathbb{D}_i$ can, potentially, be a very complex function. In general, it is possible to just approximate such a continous function to an extent.

Previous strategies to model distributions over continuous domains were briefly discussed in section 3.2.1. The two techniques mentioned there have

[2]In fact, only $u_i - 1$ of these probabilities are required.

serious limitations. They both rely on the assumption that the search can be focused on a single region within $[a_i, b_i]$, and as such they are not efficient if the interval contains multiple promising areas. Furthermore, neither scheme includes negative feedback, and they both rely on a high number of generations for learning.

As the number of learning steps is limited in our case, we propose a new technique that incorporates positive and negative learning and that is capable of approximating more versatile distributions. An example is provided in section 5.1.2.

Our approach limits the probability distribution function to a certain kind of step function which we define with the help of a one-dimensional Self-Organising Map (SOM). The application of this neural network, a summary of which can be found in appendix C, is motivated by its use of a simple learning mechanism that results in a concentration of neurons in areas with high positive feedback. The learning mechanism is described in section 4.10.2.

We assume an one-dimensional SOM with u_i neurons with weights

$$a_i \leq W_{i,1} < \cdots < W_{i,u_i} \leq b_i. \tag{4.6b}$$

These u_i neurons divide the interval $[a_i, b_i]$ into u_i sub-intervals $[a_i, \frac{W_{i,1}+W_{i,2}}{2})$, $[\frac{W_{i,1}+W_{i,2}}{2}, \frac{W_{i,2}+W_{i,3}}{2})$, $\ldots$, $[\frac{W_{i,u_i-2}+W_{i,u_i-1}}{2}, \frac{W_{i,u_i-1}+W_{i,u_i}}{2})$, and $[\frac{W_{i,u_i-1}+W_{i,u_i}}{2}, b_i]$, where the j-th sub-interval contains all values that are closest to the j-th neuron weight. The idea is that each of the u_i neurons is equally likely to be

chosen, i.e. the probability that a random value comes from any of the u_i sub-interval is $\frac{1}{u_i}$. Furthermore, all values from the same sub-interval should be equally probable. Consequently, the area under the resulting density function for each of the u_i sub-intervals must be $\frac{1}{u_i}$, and the density function must be constant within each sub-interval, or, more specifically must take the value $\frac{1}{u_i} \cdot \frac{1}{\text{interval length}}$. The probability density function d_i is therefore given by

$$d_i(x) = \begin{cases} 0 & \text{if } x \notin [a_i, b_i] \\ \frac{1}{u_i} \cdot \frac{2}{W_{i,1}+W_{i,2}-2a_i} & \text{if } x \in [a_i, \frac{W_{i,1}+W_{i,2}}{2}) \\ \frac{1}{u_i} \cdot \frac{2}{W_{i,3}-W_{i,1}} & \text{if } x \in [\frac{W_{i,1}+W_{i,2}}{2}, \frac{W_{i,2}+W_{i,3}}{2}) \\ \dots & \\ \frac{1}{u_i} \cdot \frac{2}{W_{i,u_i}-W_{i,u_i-2}} & \text{if } x \in [\frac{W_{i,u_i-2}+W_{i,u_i-1}}{2}, \frac{W_{i,u_i-1}+W_{i,u_i}}{2}) \\ \frac{1}{u_i} \cdot \frac{2}{2b_i-W_{i,u_i}-W_{i,u_i-1}} & \text{if } x \in [\frac{W_{i,u_i-1}+W_{i,u_i}}{2}, b_i] \end{cases} . \quad (4.6c)$$

As usual, the probability distribution function is defined as

$$\mathcal{F}_{\mathbb{D}_i}(x) = \int_{-\infty}^{x} d_i(t)dt, \quad x \in \mathbb{R}, \quad (4.6d)$$

in this case.

The closer (the weights of) the neurons are, the narrower the described sub-intervals are and, as a consequence, the density function is larger in these sub-intervals. In other words, a concentration of neurons in a certain region of $[a_i, b_i]$ leads to higher density function values in this region, whereas a region

with few neurons shows lower density function values. The next example clarifies these characteristics.

Example 4.1. We assume the domain $\mathbb{D} = [0, 1]$, and a SOM with four weights $W_1 = 0.35$, $W_2 = 0.45$, $W_3 = 0.55$, and $W_4 = 0.85$. The sub-interval belonging to W_3, for instance, reaches from the median of W_2 and W_3, $\frac{0.45+0.55}{2} = 0.5$, to the median of W_3 and W_4, $\frac{0.55+0.85}{2} = 0.7$, and is hence $[0.5, 0.7)$. As there are four such sub-intervals, the area under the density function over $[0.5, 0.7)$ must be $\frac{1}{4}$, and hence the density function must yield the constant value $\frac{1}{4} \cdot \frac{1}{0.7-0.5} = 1.25$ over this sub-interval. The complete density function is thus

$$d(x) = \begin{cases} 0 & \text{if } x \notin [0.0, 1.0] \\ 0.625 & \text{if } x \in [0.0, 0.4) \\ 2.5 & \text{if } x \in [0.4, 0.5) \\ 1.25 & \text{if } x \in [0.5, 0.7) \\ 0.833 & \text{if } x \in [0.7, 1.0] \end{cases} .$$

This is shown in figure 4.1. The graph illustrates that the concentration of neurons at 0.35, 0.45 and 0.55 leads to the highest density function value of 2.5 in the interval $[0.4, 0.5)$. On the other hand, the low density function value 0.625 in $[0.0, 0.4)$ indicates a low concentration of neurons around this area. □

To summarise, whether the domain $\mathbb{D}_i$ is a finite set or an interval, we can in

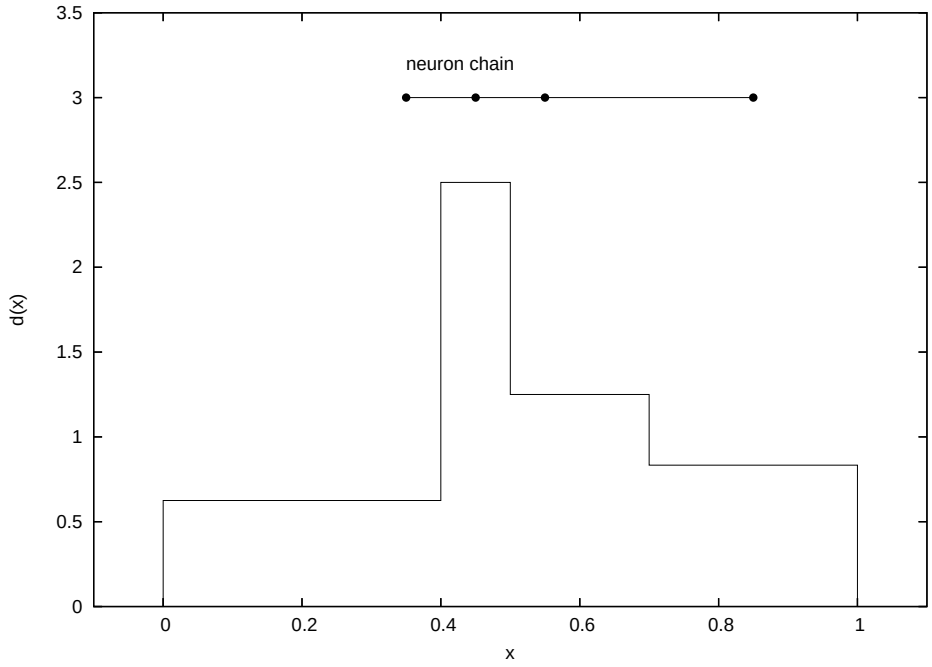

Figure 4.1: Example density function

both cases describe a probability distribution over $\mathbb{D}_i$ with u_i real numbers, that are either interpreted as element probabilities

$$\mathcal{F}_{\mathbb{D}_i} = \mathcal{F}(P_{i,1}, \ldots, P_{i,u_i}) \tag{4.7a}$$

or neuron weights

$$\mathcal{F}_{\mathbb{D}_i} = \mathcal{F}(W_{i,1}, \ldots, W_{i,u_i}). \tag{4.7b}$$

4.6 Auxiliary Functions

At this point, we introduce several auxiliary functions which help to simplify definitions and descriptions in the latter part of this chapter. Thereafter, an

example is provided in order to clarify the meaning of these functions and to demonstrate their application.

- For a given individual $I = \langle \vec{s}, \vec{e} \rangle = \langle (s_1, \ldots, s_n), \vec{e} \rangle$ and a natural number $i \in \{1, \ldots, n\}$, the function v returns the value from domain $\mathbb{D}_i$ which is encoded by I:

$$\begin{aligned} &v : \mathbb{I} \times \{1, \ldots, n\} \to \mathbb{R} \\ &v\big(\langle (s_1, \ldots, s_n), \vec{e} \rangle, i\big) = s_i \end{aligned} \tag{4.8}$$

  As $\mathbb{D}_1$ encodes the indices of the algorithms A_1 to A_m, we can write shorter:

$$\begin{aligned} &alg : \mathbb{I} \to \{1, \ldots, m\} \\ &alg(I) = v(I, 1). \end{aligned} \tag{4.9}$$

- If we are interested in the algorithm the i-th domain $\mathbb{D}_i$ belongs to, we can use the following function:

$$\begin{aligned} &alg : \{2, \ldots, n\} \to \{1, ..., m\} \\ &alg(i) = j \quad \Longleftrightarrow \quad 1 + \sum_{k=1}^{j-1} l_k < i \leq 1 + \sum_{k=1}^{j} l_k \end{aligned} \tag{4.10}$$

- The mapping π yields the parameter setting encoded by an individual

I for algorithm A_i:

$$\begin{aligned} &\pi : \mathbb{I} \times \{1, \ldots, m\} \to \bigcup_{i=1}^{m} \mathbb{R}^{l_i} \\ &\pi(I, i) = \big(v(I, p+1), \ldots, v(I, p+l_i)\big) \quad \text{with } p = 1 + \sum_{j=1}^{j<i} l_j. \end{aligned} \tag{4.11a}$$

With no algorithm index specified, the represented algorithm $alg(I)$ is assumed:

$$\pi(I) = \pi\big(I, alg(I)\big). \tag{4.11b}$$

- The auxiliary function h, which returns for a given individual the stored evaluation result for the i-th algorithm in generation g, is defined as follows:

$$\begin{aligned} &h : \mathbb{I} \times \{1, \ldots, m\} \times \{1, \ldots, G\} \to \mathbb{R} \\ &h\Big(\langle \vec{s}, \big((e_{1,1}, \ldots, e_{1,m}), \ldots, (e_{G,1}, \ldots, e_{G,m})\big)\rangle, i, g\Big) = e_{g,i} \end{aligned} \tag{4.12a}$$

By omitting the algorithm index i, we presume algorithm $A_{alg(I)}$:

$$h(I, g) = h(I, alg(I), g). \tag{4.12b}$$

Example 4.2. Two algorithms ($m = 2$) A_1 and A_2 with two parameters each ($l_1 = l_2 = 2$) are given. Parameters $\mathcal{P}_{1,1}$ and $\mathcal{P}_{1,2}$ of A_1 can take values from the domains $\mathbb{D}_{1,1} = \{2, 4, 6\}$ and $\mathbb{D}_{1,2} = [0, 1]$, respectively, while the values for the parameters of A_2 are chosen from $\mathbb{D}_{2,1} = [0, 100]$ and $\mathbb{D}_{2,2} = \{-3, -5\}$, respectively.

The parameter space for algorithm A_1 is $\mathbb{P}_1 = \{2, 4, 6\} \times [0, 1]$, while the parameter space for A_2 is given by $\mathbb{P}_2 = [0, 100] \times \{-3, -5\}$. Hence the resulting solution space is $\mathbb{S} = \{1, 2\} \times \{2, 4, 6\} \times [0, 1] \times [0, 100] \times \{-3, -5\}$ with $n = 1 + 2 + 2 = 5$. The evaluation space takes the form $\mathbb{E} = \mathbb{R}^2$. With a total number of cycles of $G = 3$, the histories of individuals come from the space $\mathbb{E}^G = (\mathbb{R}^2)^3$.

The individual $I = \langle (1, 4, 0.57, 37.1, -3), \big((10, \emptyset), (9, \emptyset), (11, \emptyset)\big) \rangle$ is a point in the individual space $\mathbb{I} = \mathbb{S} \times \mathbb{E}^G$. The space here is $\{1, 2\} \times \{2, 4, 6\} \times [0, 1] \times [0, 100] \times \{-3, -5\} \times (\mathbb{R}^2)^3$.

I describes the choice of algorithm A_1 with the parameter values 4 and 0.57. This is reflected in $alg(I) = 1$ and $\pi(I) = \pi(I, 1) = (4, 0.57)$. $\pi(I, 2)$, on the other hand, yields $(37.1, -3)$, i.e. I encodes the parameter values 37.1 and -3 for A_2.

The value of $v(I, 3)$ is 0.57 as the third element of the solution part of the individual contains this number. As this value comes from the domain $\mathbb{D}_3 = [0, 1]$ which belongs to the second parameter of algorithm A_1, the term $alg(3)$ is evaluated as 1, the index of this algorithm.

The history of I holds only evaluation results for the first algorithm A_1. As an example, the stored evaluation for this algorithm in cycle three is $h(I, 1, 3) = 11$. Zero results, as in $h(I, 2, 2) = \emptyset$, indicate that no results for the second algorithm have been collected. □

4.7 Initialising the Joint Probability Distributions

During the initialisation of the joint probability distribution (line 2 in the GPBIL figure 3.3), all marginal probability distribution functions over the domains get initialised. As no prior knowledge is available at this point, these distributions get instantiated as equal distributions. More specifically, for $i = 1, \ldots, n$ and $j = 1, \ldots, u_i$, we choose

$$P_{i,j} = \frac{1}{u_i} \tag{4.13}$$

when $\mathbb{D}_i$ is a finite set. If the i-th domain is an interval $[a_i, b_i]$, we apply a neuron chain with L neurons

$$u_i = L, \tag{4.14a}$$

where L is a parameter of the MAPAA and must be predefined. We then initialise the neuron weights according to the rule

$$W_{i,j} = a_i + \frac{2j-1}{2} \cdot \frac{b_i - a_i}{u_i}. \tag{4.14b}$$

This yields the density function

$$d_i(x) = \begin{cases} 0 & \text{if } x \notin [a_i, b_i] \\ \frac{1}{b_i - a_i} & \text{if } x \in [a_i, b_i] \end{cases} \tag{4.14c}$$

and hence an equal distribution over $[a_i, b_i]$.

The function $initialisation : \tilde{\mathbb{S}} \rightarrow \mathbb{F}^n$, which maps the solution space onto a joint probability distribution function, therefore has the following pseudo code:

Function $initialisation(\mathbb{D}_1 \times \cdots \times \mathbb{D}_n)$
1 FOR $i = 1, \ldots, n$ DO
2 IF $\mathbb{D}_i$ is finite domain THEN
3 $u_i = size(\mathbb{D}_i)$
4 FOR $j = 1, \ldots, u_i$ DO $P_{i,j} = \frac{1}{u_i}$
5 $\mathcal{F}_{\mathbb{D}_i} = \mathcal{F}(P_{i,1}, \ldots, P_{i,u_i})$
6 ELSE
7 $u_i = L$
8 FOR $j = 1, \ldots, u_i$ DO $W_{i,j} = a_i + \frac{2j-1}{2} \cdot \frac{b_i - a_i}{u_i}$
9 $\mathcal{F}_{\mathbb{D}_i} = \mathcal{F}(W_{i,1}, \ldots, W_{i,u_i})$
10 RETURN $\mathcal{F}_{\mathbb{D}_1} \cdot \ldots \cdot \mathcal{F}_{\mathbb{D}_n}$

Figure 4.2: Pseudo code of the *initialisation* function

4.8 Sampling from the Joint Probability Distribution

During the sampling process (line 5 in the GPBIL figure 3.3), a solution vector $\vec{s} \in \mathbb{S}$ is constructed by choosing an element from each domain $\mathbb{D}_i$, $i = 1, \ldots, n$, according to the probability distributions $\mathcal{F}_{\mathbb{D}_i}$ over these domains. The mutation rate μ, a parameter of the MAPAA, provides a small probability that this choice is purely random, i.e. a value according to the equal distribution $\mathcal{F}^{equal}_{\mathbb{D}_i}$ over $\mathbb{D}_i$ is selected. The function $sampling : \mathbb{F}^n \rightarrow \mathbb{S}$ is presented as a code fragment in figure 4.3.

Function $sampling(\mathcal{F}_{\mathbb{D}_1} \cdot \ldots \cdot \mathcal{F}_{\mathbb{D}_n})$
1 FOR $i = 1, \ldots, n$ DO
2 IF $random([0,1]) < \mu$ THEN $s_i = random(\mathcal{F}_{\mathbb{D}_i}^{equal})$
3 ELSE $s_i = random(\mathcal{F}_{\mathbb{D}_i})$
4 RETURN $(s_1, \ldots, s_n)$

Figure 4.3: Pseudo code of the *sampling* function

4.9 Evaluating Individuals

The pseudo code of the evaluation function $evaluation : \mathbb{I} \times \mathbb{N} \rightarrow \mathbb{E}$, which is used in lines 6 and 7 in the GPBIL figure 3.3, can be found in figure 4.4. Presented with an individual I and the current cycle number g, this function returns a vector of m real numbers. Only in the case of the best individual I^{best} are evaluation results from all m algorithms required. For all other individuals I, only the single objective function value for algorithm $A_{alg(I)}$, which is the algorithm described by I, is of importance and is in fact calculated.

Function $evaluation(I, g)$
1 IF $I \neq I^{best}$ THEN RETURN $\Big(\emptyset, \ldots, \emptyset, f\big(A_{alg(I)}(\pi(I), \Phi_g)\big), \emptyset, \ldots, \emptyset\Big)$
2 ELSE RETURN $\Big(f\big(A_1(\pi(I,1), \Phi_g)\big), \ldots, f\big(A_m(\pi(I,m), \Phi_g)\big)\Big)$[3]

Figure 4.4: Pseudo code of the *evaluation* function

[3] As a reminder, $f\big(A_i(\pi(I,i), \Phi_g)\big)$ is the objective function value of the solution produced by algorithm A_i with the parameter setting $\pi(I,i)$ when applied to the optimisation problem Φ_g.

4.10 Updating of the Joint Probability Distribution

The updating of the joint probability distribution $\mathcal{F}_{\mathbb{S}}$ in line 8 of the GPBIL figure 3.3 is essential for GPBIL in order to learn what are favourable and less favourable choices from the domains $\mathbb{D}_i$, $i = 1, \ldots, n$. As these domains can take two different shapes, we first introduce the basic learning rules for both finite and interval domains. Thereafter, the complete learning scheme is described.

4.10.1 Learning over Finite Domains

Following the learning rule from standard Population-Based Incremental Learning (PBIL), positive feedback from a value $d_{i,k} \in \mathbb{D}_i = \{d_{i,1}, \ldots, d_{i,u_i}\}$ leads to an increased likelihood of this value:

$$P_{i,k}^{new} = 1 \cdot \epsilon + P_{i,k}^{old} \cdot (1 - \epsilon) = P_{i,k}^{old} + (1 - P_{i,k}^{old}) \cdot \epsilon$$

In this formula, $\epsilon \in (0, 1)$ is the learning rate. All other probabilities $P_{i,j}$ with $j \neq k$ are decreased according to

$$P_{i,j}^{new} = P_{i,j}^{old} \cdot \frac{1 - P_{i,k}^{new}}{1 - P_{i,k}^{old}}.$$

The rule for positive learning over a finite set domain can be thus comprising as

$$P_{i,j}^{new} = \begin{cases} P_{i,j}^{old} + (1 - P_{i,j}^{old}) \cdot \epsilon & \text{if } j = k \\ P_{i,j}^{old} \cdot \frac{1-P_{i,k}^{new}}{1-P_{i,k}^{old}} & \text{if } j \neq k \end{cases}. \tag{4.15}$$

It is worth noting that this rule maintains a number of characteristics. Firstly, all probabilities are from $[0, 1]$. Secondly, all probabilities sum up to 1

$$\begin{aligned} \sum_{j=1}^{u_i} P_{i,j}^{new} &= P_{i,k}^{new} + \sum_{j=1, j\neq k}^{u_i} P_{i,j}^{new} \\ &= P_{i,k}^{new} + \sum_{j=1, j\neq k}^{u_i} P_{i,j}^{old} \cdot \frac{1 - P_{i,k}^{new}}{1 - P_{i,k}^{old}} \\ &= P_{i,k}^{new} + \frac{1 - P_{i,k}^{new}}{1 - P_{i,k}^{old}} \cdot \sum_{j=1, j\neq k}^{u_i} P_{i,j}^{old} \\ &= P_{i,k}^{new} + \frac{1 - P_{i,k}^{new}}{1 - P_{i,k}^{old}} \cdot (1 - P_{i,k}^{old}) \\ &= 1. \end{aligned}$$

Thirdly, the ratio of two probabilities P_{i,j_1} and P_{i,j_2} with $j_1 \neq k$ and $j_2 \neq k$ is unchanged ($P_{i,j_2}^{old} > 0$ is assumed):

$$\frac{P_{i,j_1}^{new}}{P_{i,j_2}^{new}} = \frac{P_{i,j_1}^{old} \cdot \frac{1-P_{i,k}^{new}}{1-P_{i,k}^{old}}}{P_{i,j_2}^{old} \cdot \frac{1-P_{i,k}^{new}}{1-P_{i,k}^{old}}} = \frac{P_{i,j_1}^{old}}{P_{i,j_2}^{old}}.$$

The rule for negative learning, based on the formula

$$P_{i,k}^{new} = 0 \cdot \epsilon + P_{i,k}^{old} \cdot (1 - \epsilon) = P_{i,k}^{old} + (0 - P_{i,k}^{old}) \cdot \epsilon,$$

can be derived analogically:

$$P_{i,j}^{new} = \begin{cases} P_{i,j}^{old} + (0 - P_{i,j}^{old}) \cdot \epsilon & \text{if } j = k \\ P_{i,j}^{old} \cdot \frac{1 - P_{i,k}^{new}}{1 - P_{i,k}^{old}} & \text{if } j \neq k \end{cases}. \tag{4.16}$$

Example 4.3. We assume the domain $\{1, 2, 3\}$ with the probabilities $P_1 = 0.5$, $P_2 = 0.3$ and $P_3 = 0.2$. Positive feedback for the value 1 with a learning rate of 0.2 leads to updated probabilities of $P_1^{new} = 0.5 + (1 - 0.5) \cdot 0.2 = 0.6$, $P_2^{new} = 0.3 \cdot \frac{1-0.6}{1-0.5} = 0.24$ and $P_3^{new} = 0.2 \cdot \frac{1-0.6}{1-0.5} = 0.16$. The sum of these new likelihoods is $0.6 + 0.24 + 0.16 = 1$. If, on the other hand, negative learning for the value 1 is assumed, again with a learning rate of 0.2, then the resulting probabilities would be $P_1^{new} = 0.5 + (0 - 0.5) \cdot 0.2 = 0.4$, $P_2^{new} = 0.3 \cdot \frac{1-0.4}{1-0.5} = 0.36$ and $P_3^{new} = 0.2 \cdot \frac{1-0.4}{1-0.5} = 0.24$, which sum up to 1 as well: $0.4 + 0.36 + 0.24 = 1$. □

4.10.2 Learning over Interval Domains

If $\mathbb{D}_i$ is an interval $[a_i, b_i]$, then the PDF over this domain is given by the u_i neuron weights $a_i \leq W_{i,1} < \cdots < W_{i,u_i} \leq b_i$. The rule applied for modifying these weights is the learning rule for Self-Organising Maps. Positive feedback from a value $X \in [a_i, b_i]$ leads firstly to the determination of the best

matching weight $W_{i,k}$

$$k = argmin_{j=1}^{u_i}(|X - W_{i,j}|)^4, \tag{4.17a}$$

and secondly to a weights update according to

$$W_{i,j}^{new} = W_{i,j}^{old} + \epsilon \cdot h(k,j) \cdot (X - W_{i,j}^{old}).$$

$\epsilon \in [0,1]$ is the rate of positive learning, while $h(k,j)$ represents the neighbourhood kernel. If a cylinder neighbourhood kernel is of size δ then

$$h(k,j) = \begin{cases} 1 & \text{if } |k-j| \leq \delta \\ 0 & \text{else} \end{cases}$$

is applied and the learning rule can be formulated as

$$W_{i,j}^{new} = \begin{cases} W_{i,j}^{old} + \epsilon \cdot (X - W_{i,j}^{old}) & \text{if } |k-j| \leq d \\ W_{i,j}^{old} & \text{else} \end{cases}. \tag{4.17b}$$

The standard SOM concept does not include the notion of negative feedback, i.e. negative learning. But to our algorithm, the ability to use negative experience is important. Therefore, a rule for negative learning in a SOM is introduced. Its basic idea is the movement of neurons away from a negative input X within a certain neighbourhood around it, in contrast to neuron

[4] $argmin_i(x_i)$ returns the index i for which x_i is minimal.

movement towards X in case of positive learning. After determining the closest neuron

$$k = argmin_{j=1}^{u_i}(|X - W_{i,j}|), \tag{4.18a}$$

the neuron weights get modified according to

$$W_{i,j}^{new} = \begin{cases} W_{i,j}^{old} + \epsilon \cdot (W_{i,k-\delta}^{old} - W_{i,j}^{old}) & \text{if } |k-j| \leq \delta \text{ and } W_{i,j}^{old} \leq X \\ W_{i,j}^{old} + \epsilon \cdot (W_{i,k+\delta}^{old} - W_{i,j}^{old}) & \text{if } |k-j| \leq \delta \text{ and } W_{i,j}^{old} > X \cdot \\ W_{i,j}^{old} & \text{else} \end{cases} \tag{4.18b}$$

If the weights $W_{i,k-\delta}^{old}$ or $W_{i,k+\delta}^{old}$ in the formula above are not defined, we use $W_{i,k-\delta}^{old} = a_i$ and $W_{i,k+\delta}^{old} = b_i$, respectively.

The following example illustrates the effects of both positive and negative learning.

Example 4.4. The domain $[0.05, 0.95]$ is given with 9 evenly distributed weights $W_1 = 0.1, W_2 = 0.2, \ldots, W_9 = 0.9$. Positive feedback from the value 0.48 results in $k = 5$ as $W_5 = 0.5$ is the best matching, i.e. nearest, weight. With a learning rate of 0.5 and a neighbourhood size of 2, only the weights W_4, W_5 and W_6 get modified by moving them towards 0.48: $W_4^{new} = 0.4 + 0.5 \cdot (0.48 - 0.4) = 0.44$, $W_5^{new} = 0.5 + 0.5 \cdot (0.48 - 0.5) = 0.49$ and $W_6^{new} = 0.6 + 0.5 \cdot (0.48 - 0.6) = 0.54$.

But if $X = 0.48$ is negative feedback, then the weight W_4 gets shifted towards

W_3, and W_5 and W_6 get moved towards W_7: $W_4^{new} = 0.4 + 0.5 \cdot (0.3 - 0.4) = 0.35$, $W_5^{new} = 0.5 + 0.5 \cdot (0.7 - 0.5) = 0.6$ and $W_6^{new} = 0.6 + 0.5 \cdot (0.7 - 0.6) = 0.65$. Figure 4.5 shows the concentration of neurons around $X = 0.48$ after positive learning, and their away movement from this value after negative feedback. □

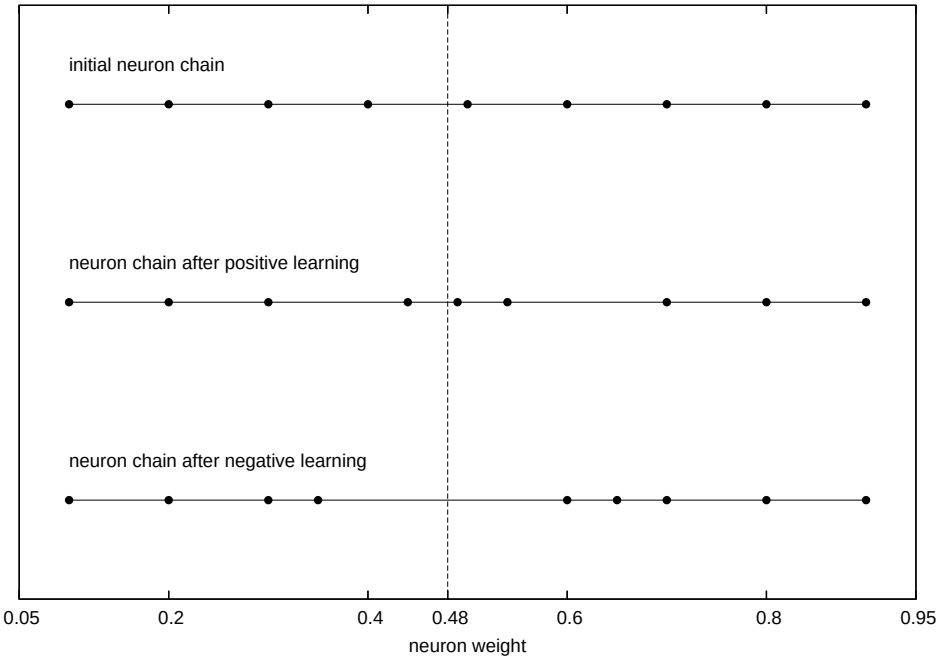

Figure 4.5: Neuron weights after positive and negative learning

4.10.3 The Learning Scheme

Having discussed the basic rules for positive and negative learning for both finite and interval domains, we are now in a position to describe the entire learning mechanism as applied in the MAPAA. This strategy is presented in three steps with the corresponding functions building upon each other.

The Learning Function *learning*1

Objective: The aim of function *learning*1 is the abstraction of a single learning step, i.e. a single update of a single PDF, regardless of whether positive or negative learning over a finite or interval domain is performed.

Description: Four input parameters are required to allow the updating of a single PDF by learning from a single value: a PDF $\mathcal{F}_{\mathbb{D}_i}$ that is to be updated, a value $X \in \mathbb{D}_i$ from which to learn, a weight $\omega \in [0, 1]$ influencing the amount of learning, and a flag $b \in \{0, 1\}$ indicating positive ($b = 1$) or negative ($b = 0$) learning. The function returns the updated probability distribution as its result.

The pseudo code, which is shown in figure 4.6, combines the four learning rules discussed in equations 4.15, 4.16, 4.17 and 4.18. It contains six MAPAA parameters that must be predefined. The learning rates for positive and negative learning over finite and interval domains are given by ϵ^+_{fin}, ϵ^-_{fin}, ϵ^+_{int} and ϵ^-_{int}. In the case of an interval domains, the neighbourhhood kernel sizes δ^+ for positive and δ^- for negative feedback must be provided as well.

The Learning Function *learning*2

Objective: With the second learning function we describe a mechanism for the repeated modification of a single probability distribution function through learning from a group of individuals.

Function $learning1(\mathcal{F}_{\mathbb{D}_i}, X, \omega, b)$

1 IF $\mathbb{D}_i$ is finite domain THEN

2 $\quad X \equiv d_{i,k}$

3 $\quad$ IF b is 1 THEN $P_{i,k}^{new} = P_{i,k}^{old} + (1 - P_{i,k}^{old}) \cdot \epsilon_{fin}^{+} \cdot \omega$

4 $\quad$ ELSE $P_{i,k}^{new} = P_{i,k}^{old} + (0 - P_{i,k}^{old}) \cdot \epsilon_{fin}^{-} \cdot \omega$

5 $\quad$ FOR $j = 1, \ldots, u_i, j \neq k$ DO $\quad P_{i,j}^{new} = P_{i,j}^{old} \cdot \frac{1 - P_{i,k}^{new}}{1 - P_{i,k}^{old}}$

6 $\quad$ RETURN $\mathcal{F}(P_{i,1}^{new}, \ldots, P_{i,u_i}^{new})$

7 ELSE

8 $\quad k = argmin_{j=1}^{u_i}(|X - W_{i,j}|)$

9 $\quad$ IF b is 1 THEN FOR $j = 1, \ldots, u_i$ DO

10 $\quad\quad W_{i,j}^{new} = \begin{cases} W_{i,j}^{old} + \epsilon_{int}^{+} \cdot \omega \cdot (X - W_{i,j}^{old}) & \text{if } |k - j| \leq \delta^{+} \\ W_{i,j}^{old} & \text{else} \end{cases}$

11 $\quad$ ELSE FOR $j = 1, \ldots, u_i$ DO

12 $\quad\quad W_{i,j}^{new} = \begin{cases} W_{i,j}^{old} + \epsilon_{int}^{-} \cdot \omega \cdot (W_{i,k-\delta^{-}}^{old} - W_{i,j}^{old}) & \text{if } |k - j| \leq \delta^{-},\ W_{i,j}^{old} \leq X \\ W_{i,j}^{old} + \epsilon_{int}^{-} \cdot \omega \cdot (W_{i,k+\delta^{-}}^{old} - W_{i,j}^{old}) & \text{if } |k - j| \leq \delta^{-},\ W_{i,j}^{old} > X \\ W_{i,j}^{old} & \text{else} \end{cases}$

13 $\quad$ RETURN $\mathcal{F}(W_{i,1}^{new}, \ldots, W_{i,u_i}^{new})$

Figure 4.6: Pseudo code of the *learning*1 function

Description: The function *learning*2, shown in figure 4.7, takes four input parameters: a single PDF $\mathcal{F}_{\mathbb{D}_i}$ which has to be updated, a real-valued reference evaluation e, a vector $\vec{I} = (I_1, \ldots, I_l)$ of l individuals from which to learn, and the current cycle number g. Its return value is the modified PDF $\mathcal{F}_{\mathbb{D}_i}$.

In the first part of this method (lines 1 and 2), the population[6] $\vec{I}$ gets partitioned into two groups: those individuals, $\vec{I^{+}}$, that show a better evaluation[7] than the reference evaluation e in cycle g, and those individuals, $\vec{I^{-}}$, that did

[5] *ascending_sort* sorts the elements of a set in ascending order and returns them as a vector.

[6] This is not necessarily the whole MAPAA population $(I_1, \ldots, I_M)$.

[7] As we assume minimisation problems, we mean a smaller evaluation result.

Function $learning2\big(\mathcal{F}_{\mathbb{D}_i}, e, (I_1, \ldots, I_l), g\big)$
1 $\vec{I^+} = ascending\_sort\big(\{I_j | h(I_j, g) < e\}\big)$[5]
2 $\vec{I^-} = ascending\_sort\big(\{I_j | h(I_j, g) \geq e\}\big)$
3 FOR $j = size(\vec{I^-}), \ldots, 1$ DO
4 $\quad \mathcal{F}_{\mathbb{D}_i} = learning1(\mathcal{F}_{\mathbb{D}_i}, v(I_j^-, i), \frac{j}{size(\vec{I^-})}, 0)$
5 FOR $j = size(\vec{I^+}), \ldots, 1$ DO
6 $\quad \mathcal{F}_{\mathbb{D}_i} = learning1(\mathcal{F}_{\mathbb{D}_i}, v(I_j^+, i), \frac{size(\vec{I^+})+1-j}{size(\vec{I^+})}, 1)$
7 RETURN $\mathcal{F}_{\mathbb{D}_i}$

Figure 4.7: Pseudo code for the *learning*2 function

not perform better than e. Thereafter in the second phase, negative learning from the individuals in $\vec{I^-}$ (lines 3 and 4) and positive learning from the members of $\vec{I^+}$ (lines 5 and 6) is performed. Both learning processes are linearly weighted: the better/worse the evaluation of a population member is, the higher the weight is and thus the effect of the positive/negative feedback. If there are, for instance, three individuals with a better performance than the reference value e, then a weight of $\frac{1}{3}$ is used for the third best individual, the weight $\frac{2}{3}$ is applied for the second best, and learning from the best population member is performed with a maximum weight of $\frac{1}{1}$.

Since an individual $I_j \in \{I_1 \ldots, I_l\}$ encodes $v(I_j, i)$ as its choice from domain $\mathbb{D}_i$, this value is used as the source for positive feedback (if I_j belongs to $\vec{I^+}$) or negative feedback (if I_j belongs to $\vec{I^-}$) because we aim to modify the PDF $\mathcal{F}_{\mathbb{D}_i}$ over $\mathbb{D}_i$.

The Complete Learning Mechanism *learning*

Objective: Based on the the abstraction of basic learning steps through the functions *learning*1 and *learning*2, we present the complete learning mechanism *learning*, as used in line 8 in the GPBIL figure 3.3, in a compact form.

Description: The learning scheme is based on a simple idea. Population members that outperform the best so far encountered individual see the likelihoods of the parameter choices they represent increased, while underperforming individuals see a probability decrease of encoded choices. This strategy leads, over time, to a concentration on values that often feature in better candidate solutions.

The function $learning : \mathbb{F}^n \times \mathbb{I} \times \mathbb{I}^M \times \{1, \ldots, G\} \rightarrow \mathbb{F}^n$, the pseudo code of which is listed in figure 4.8, manages the modification of the joint probability distribution by learning from results obtained by population members in the latest generational cycle. Therefore, the following input data are required: the joint probability distribution $\mathcal{F}_{\mathbb{S}} = \mathcal{F}_{\mathbb{D}_1} \cdot \ldots \cdot \mathcal{F}_{\mathbb{D}_n}$ over the solution space $\mathbb{S}$, the best individual so far encountered I^{best}, the current population $\vec{I} = (I_1, \ldots, I_M)$ and the current generation number g. It returns the updated joint probability distribution $\mathcal{F}_{\mathbb{S}}$.

The working principle of the *learning* mechanism is simple: it modifies the n PDFs $\mathcal{F}_{\mathbb{D}_i}$ one by one. The PDF over the first domain $\mathbb{D}_1$ describes the

[8] *vec* transforms a set into a vector.

Function $learning(\mathcal{F}_{\mathbb{D}_1} \cdot \ldots \cdot \mathcal{F}_{\mathbb{D}_n}, I^{best}, (I_1, \ldots, I_M), g)$
1 $\mathcal{F}_{\mathbb{D}_1} = learning2(\mathcal{F}_{\mathbb{D}_1}, h(I^{best}, g), \vec{I}, g)$
2 FOR $i = 2, \ldots, n$ DO
3 $\quad \vec{I}' = vec(\{I_j | alg(I_j) = alg(i)\})$[8]
4 $\quad \mathcal{F}_{\mathbb{D}_i} = learning2(\mathcal{F}_{\mathbb{D}_i}, h(I^{best}, alg(i), g), \vec{I}', g)$
5 RETURN $\mathcal{F}_{\mathbb{D}_1} \cdot \ldots \cdot \mathcal{F}_{\mathbb{D}_n}$

Figure 4.8: Pseudo code of the *learning* function

distribution of good and less good methods among the algorithms A_1 to A_m. It is updated by comparing the evaluation results achieved by algorithms present in the current population with the reference evaluation $h(I^{best}, g)$, which is the evaluation of the best found algorithm in cycle g (line 1). Algorithms improving on this reference evaluation become more probable, i.e. see their likelihood in $\mathcal{F}_{\mathbb{D}_1}$ increased, while algorithms demonstrating a worse performance become less likely.

The set $\mathbb{D}_i$, $i \in \{2, \ldots, n\}$, is the domain of a parameter belonging to algorithm $A_{alg(i)}$. Only some of the individuals $I_j \in \vec{I}$ represent this algorithm and thus have evaluation results stored for it (line 3). Only these population members can be used for the modification of $\mathbb{D}_i$ through learning (line 4). The evaluation result of method $A_{alg(i)}$ with its best parameter setting $\pi(I^{best}, alg(i))$ in cycle g, namely $h(I^{best}, alg(i), g)$, serves as the reference performance as we aim to improve this setting.

4.11 The Optimum Function

The m parameter settings $\pi(I^{best}, 1)$ to $\pi(I^{best}, m)$ describe the best encountered parameter choices for the algorithms A_1 to A_m. In addition, $alg(I^{best})$ provides the information about the best among these algorithms. Near the end of each generational cycle (line 9 in the GPBIL figure 3.3), the population is analysed in order to detect whether improvements to the above choices can be found.

The decision about whether an algorithm A_i with parameter setting π_i performs better than A_j with parameter choices π_j is non-trivial. This question cannot be answered by comparing the achieved solutions for just a single optimisation problem as heuristic search algorithms usually comprise an element of noise, as stated under C3 in section 1.5. Rather we have to compare evaluation results achieved for several optimisation problems $\Phi_1, \Phi_2, \Phi_3, \ldots$, i.e. compare

$$f\big(A_1(\pi_1, \Phi_1)\big),\ f\big(A_1(\pi_1, \Phi_2)\big),\ f\big(A_1(\pi_1, \Phi_3)\big),\ \ldots$$

with

$$f\big(A_2(\pi_2, \Phi_1)\big),\ f\big(A_2(\pi_2, \Phi_2)\big),\ f\big(A_2(\pi_2, \Phi_3)\big),\ \ldots.$$

A smaller and thus better mean of one data series is an insufficient indicator for its superiority. As absolute evaluation results can vary highly, a few outliers can change the result dramatically. Instead of looking at the absolute differences of the evaluation results, we propose to look at the relative dis-

tinctions. Assuming that all evaluation results are positive, the data series of ratios

$$\frac{f\big(A_1(\pi_1,\Phi_1)\big)}{f\big(A_2(\pi_2,\Phi_1)\big)},\ \frac{f\big(A_1(\pi_1,\Phi_2)\big)}{f\big(A_2(\pi_2,\Phi_2)\big)},\ \ldots,$$

normalised as

$$\frac{f\big(A_1(\pi_1,\Phi_1)\big)-f\big(A_2(\pi_2,\Phi_1)\big)}{f\big(A_2(\pi_2,\Phi_1)\big)},\ \frac{f\big(A_1(\pi_1,\Phi_2)\big)-f\big(A_2(\pi_2,\Phi_2)\big)}{f\big(A_2(\pi_2,\Phi_2)\big)},\ \ldots, \tag{4.19}$$

is a better indication of which of the two algorithms performs better. A negative ratio shows the superiority of the first algorithm, while a positive result points to a better performance of the second algorithm. For example, a ratio of -0.02 indicates a 2% better evaluation of A_1. As a result, a negative mean of the data series in equation 4.19 suggests that, on average, A_1 performs better than A_2, and vice versa.

But an important question remains unanswered: is a detected superiority of one algorithm over another really significant, i.e. can we statistically prove that the found difference is not due to pure chance? The one-sample and one-tailed t-test, a statistical method which is discussed in more detail in appendix B.2, is the tool we use to answer this question. Given the data used in equation 4.19, it can decide, with a certain level of confidence, whether the mean is significantly smaller than zero, and thus that A_1 is significantly better than A_2, or not. The main prerequisite for the application of the t-test is that the analysed data must follow a distribution similar to the normal distribution. As we later show empirically, this is usually the case in practice.

The realisation of the described approach within GPBIL is illustrated in figures 4.9 and 4.10. The first of the two functions, *ratio*, takes two individuals and two algorithm indices, and thus two parameter settings with the corresponding algorithms, as input. From the available evaluation results history, it compiles a vector containing the ratios described in formula 4.19:

Function $ratio(I_1, i_1, I_2, i_2)$
1 RETURN $vec\left(\{\frac{h(I_1,i_1,g)-h(I_2,i_2,g)}{h(I_2,i_2,g)} | g \in \{1,\ldots,G\} \text{ and ratio exist}\}\right)$

Figure 4.9: Pseudo code of the *ratio* function

The main function $optimum : \mathbb{I} \times \mathbb{I}^M \rightarrow \mathbb{I}$ loops through all individuals from the population and decides in each case, with the help of the t-test, whether the represented algorithm with the represented parameter setting demonstrates a significant improvement over the best parameter setting for this algorithm found so far, or even over the best overall algorithm found so far. If significant improvements can be found, I^{best} is updated in order to accommodate the new results.

Function $optimum(I^{best}, (I_1, \ldots, I_M))$
1 FOR $i = 1, \ldots, M$ DO
2 IF $ttest\Big(ratio\big(I_i, alg(I_i), I^{best}, alg(I_i)\big), \alpha, min_{ttest}\Big)$ THEN
3 UPDATE I^{best} such that $\pi\big(I^{best}, alg(I_i)\big) = \pi(I_i)$
4 IF $ttest\Big(ratio\big(I_i, alg(I_i), I^{best}, alg(I^{best})\big), \alpha, min_{ttest}\Big)$ THEN
5 UPDATE I^{best} such that $\pi\big(I^{best}, alg(I_i)\big) = \pi(I_i)$
6 UPDATE I^{best} such that $alg(I^{best}) = alg(I_i)$
7 RETURN I^{best}

Figure 4.10: Pseudo code of the *optimum* function

Our implementation of the t-test is controlled by two parameters that must be supplied. Firstly, the test can only detect significant differences if the

studied sample contains at least min_{ttest} entries. Secondly, a significance level α (see appendix B.2) must be provided in order to control the level of confidence we want to have in the results.

4.12 The Selection Process

During the selection process, individuals of the current population are selected for being passed on to the next generation (line 10 in figure 3.3). Survival can only be justified if these solutions are promising, i.e. if it is believed that they have the potential to replace the best individual so far encountered in the future.

The criterion used here is mean performance: only population members I_i from $\vec{I}$ whose evaluations are, on average, better than the evaluations of the best found individual I^{best}, are selected for the next generation. This decision is again based on the data that were outlined in equation 4.19: a negative mean leads to survival while a non-negative mean results in exclusion. The corresponding pseudo code for $selection : \mathbb{I} \times \mathbb{I}^M \rightarrow \mathbb{I}^M$ is listed in figure 4.11.

Function $selection(I^{best}, (I_1, \ldots, I_M))$
1 RETURN $vec\Big(I_i | mean\big(ratio\big(I_i, alg(I_i), I^{best}, alg(I^{best})\big)\big) < 0\Big)$

Figure 4.11: Pseudo code of the *selection* function

4.13 Summary

The Multiple Algorithms' Parameter Adaptation Algorithm (MAPAA) was introduced in this chapter as a method for tackling the Multiple Algorithms' Parameter Adaptation Problem (MAPAP). The technique was presented as an instantiation of the generalised formulation of Population-Based Incremental Learning. Definitions for all generic, and thus still unspecified, components of the framework method were provided in order to establish a fully described solution algorithm. The purpose of this chapter was purely to define the MAPAA. No empirical analysis was presented as we focus on experimental means of studying the proposed algorithm in chapter 5.

The MAPAA as defined shows a number of restrictions:

1. Parameter domains are limited to sub-sets of the real numbers that are either finite in their cardinality or are intervals.

2. Without loss of generality, the handled optimisation problems are assumed to be minimisation tasks with positive objective function values.

3. The defined joint probability distribution function is restricted to the product of unconditional probability distribution functions.

4. The ratios of the objective function results of any two heuristic search algorithms studied must follow a distribution which resembles a normal distribution.

Furthermore, the MAPAA is controlled by a number of parameters that must be user-defined. These parameters are listed in table 4.1.

MAPAA Parameter	Symbol
number of generational cycles	G
population size	M
mutation rate	μ
positive learning rate, finite domain	ϵ^{+}_{fin}
negative learning rate, finite domain	ϵ^{-}_{fin}
number of neurons for the SOM model	L
positive learning rate, interval domain	ϵ^{+}_{int}
negative learning rate, interval domain	ϵ^{-}_{int}
neighbourhood kernel size, positive learning	δ^{+}
neighbourhood kernel size, negative learning	δ^{-}
significance level of the t-test	α
minimal number of sample data required by t-test	min_{ttest}

Table 4.1: Parameters of the Multiple Algorithms' Parameter Adaptation Algorithm

Chapter 5

Experimental Study of the Multiple Algorithms' Parameter Adaptation Algorithm

The Multiple Algorithms' Parameter Adaptation Algorithm was formally introduced and described in the previous chapter. The aim of this chapter is the experimental examination and analysis of this method. It is divided into two main parts. In the first half, experiments with the models for estimating probability distributions over both finite and interval domains are presented. These focus on the intensifying and diversifying effects of positive and negative learning, respectively. In the second half of this chapter, the Multiple

Algorithms' Parameter Adaptation Algorithm itself is studied in more detail. Experiments are conducted to determine good learning schemes, to study the effect of mutation, to look at the resource distribution if the adaptation method is applied to various algorithms simultaneously, and to show that the application of the t-test is justified. The chapter concludes with a summary of the results found.

5.1 Experiments with Probability Distributions over Finite and Interval Domains

5.1.1 Experiments I and II: Learning over Finite Domains

In the first two experiments, we study how positive and negative learning works over a finite domain. The considered domain is the set $\mathbb{D} = \{1, \ldots, 10\}$ in both cases. Ten probabilities $P_1, \ldots, P_{10}$ are used to model a probability distribution function $\mathcal{F}$ over $\mathbb{D}$, as was outlined in section 4.5.1. The learning rules for updating these probabilities were introduced in equations 4.15 and 4.16. They are, rewritten in a simplified form,

$$P_i^{new} = \begin{cases} P_i^{old} + (1 - P_i^{old}) \cdot \epsilon & \text{if } j = k \\ P_i^{old} \cdot \frac{1-P_k^{new}}{1-P_k^{old}} & \text{if } j \neq k \end{cases}$$

in the case of positive learning, and

$$P_i^{new} = \begin{cases} P_i^{old} + (0 - P_i^{old}) \cdot \epsilon & \text{if } j = k \\ P_i^{old} \cdot \frac{1-P_k^{new}}{1-P_k^{old}} & \text{if } j \neq k \end{cases}$$

for negative learning. In both formulas, $k \in \mathbb{D}$ is the value from which to learn, and ϵ symbolises the applied learning rate.

Experiment I: Positive Learning over Finite Domains

Objective of the experiment: The aim of this experiment is to clarify how the application of the rule for positive learning over a finite domain leads to the estimation of an unknown probability distribution by only using sample data drawn from this distribution.

Detailed description of the experiment:

The task of the first experiment is to learn to estimate the distribution $\mathcal{F}^{target}$ over the domain $\mathbb{D} = \{1, \ldots, 10\}$, which is given by the ten probabilities $P_1^{target} = \frac{1}{55}, P_2^{target} = \frac{2}{55}, \ldots, P_{10}^{target} = \frac{10}{55}$. As initially no information is available about this distribution, its estimation $\mathcal{F}$ gets initialised as a uniform distribution, i.e. as $P_1 = P_2 = \ldots = P_{10} = \frac{1}{10}$. This estimation distribution then gets modified via the rule for positive learning by repeatedly presenting sample data k drawn from $\mathcal{F}^{target}$. A small learning rate of $\epsilon = 0.02$ is used in this experiment.

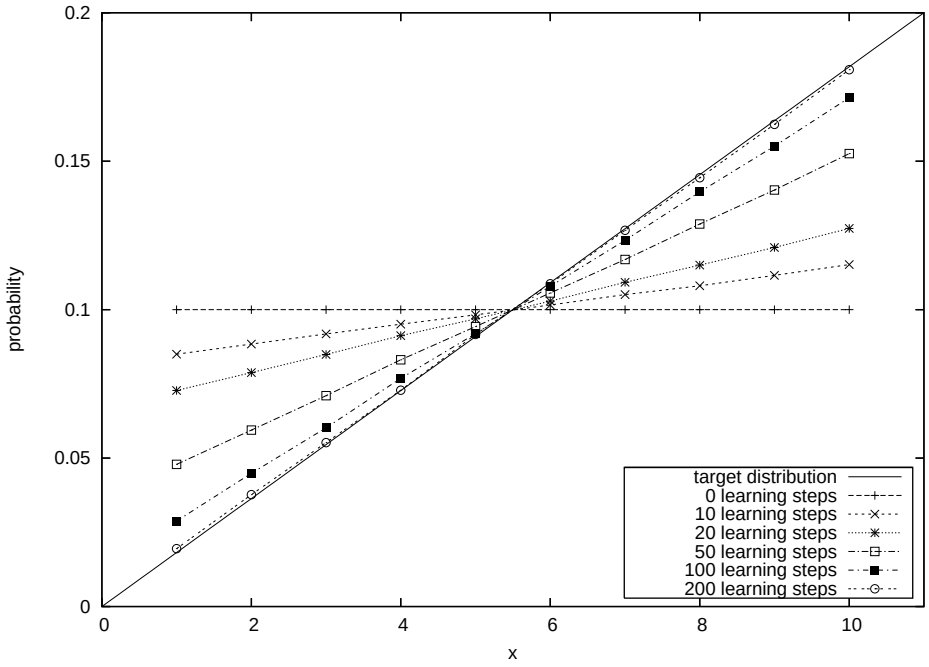

Figure 5.1: The effect of positive learning over a finite domain

Figure 5.1 presents the development of the probability distribution $\mathcal{F}$ over time (number of learning steps). For each value i from the considered domain, the corresponding probability P_i is shown after 0, 10, 20, 50, 100 and 200 learning steps. Results are averaged over 10,000 runs, and probabilities belonging to the same learning step are connected by a line for clarity. Starting as a uniform distribution, the P_i converge over time towards the values P_i^{tagret}. The more learning steps that take place, the better the estimation of $\mathcal{F}^{target}$ through $\mathcal{F}$. After 200 learning steps there is an almost perfect match. This means that the probabilities P_i have learnt to represent the distribution of the values they learnt from, the P_i^{target}.

Conclusion: The mechanism for positive feedback over finite domains enables us to estimate a target probability distribution function. The longer

the learning process lasts, i.e. the more samples that are provided, the better the estimation. The implication for the Multiple Algorithms' Parameter Adaptation Algorithm (MAPAA) is that it can learn to estimate the distribution of good values for parameters with finite ranges. As a consequence, the search can be intensified around the more promising parameter choices.

Experiment II: Negative Learning over Finite Domains

Objective of the experiment: The focus of the second experiment is on the relevance of negative learning when a probability distribution over a finite domain gets modified both through positive and negative feedback. We show that the rule for negative learning brings a diversifying element to the learning process.

Detailed description of the experiment: The probability distribution function $\mathcal{F}$ over the finite set $\mathbb{D} = \{1, 2, \ldots, 10\}$ is initially given by the ten probabilities $P_1 = \frac{1}{55}, P_2 = \frac{2}{55}, \ldots, P_{10} = \frac{10}{55}$. In this experiment, there is no target distribution to learn. Rather $\mathcal{F}$ itself is used for repeatedly drawing values from $\mathbb{D}$ during the experimental runs. These values are used for learning and thus for updating $\mathcal{F}$. Both forms of learning, positive and negative, are possible. With a certain probability r a learning step is seen as positive feedback, otherwise, i.e. with probability $1 - r$, the rule for negative feedback is applied. Results are averaged over 10,000 runs, and rates of 0.02 are used both for positive and negative learning.

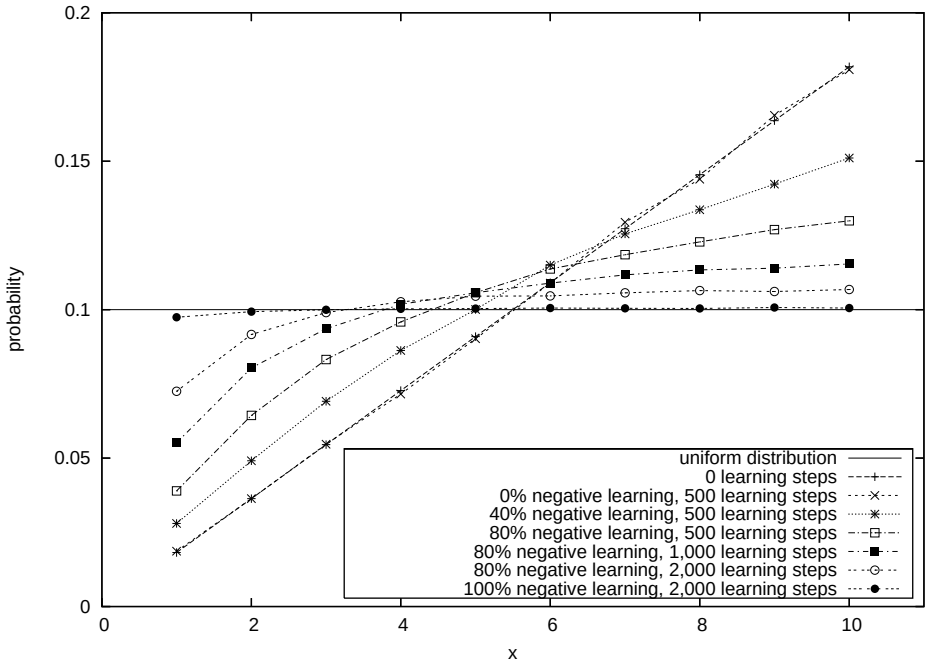

Figure 5.2: The effect of negative learning over a finite domain

The figure 5.2 shows the development of the probability distribution over time for varying ratios of positive and negative learning. The graphics illustrate that, without negative learning and thus only positive learning, the probability distribution does not change on average as it is constantly reinforced: the distribution of values after 500, purely positive, learning steps is congruent to the initial distribution at 0 cycles. However, with 60% positive learning and thus a 40% chance of negative learning, a small change is noticeable after 500 cycles. The probabilities move slightly towards the uniform distribution. This effect is more visible after 500 cycles with an 80% probability of negative learning, and progresses as more learning steps are performed. In the extreme case, 100% negative learning leads to an almost uniform distribution after 2,000 learning cycles. This can be interpreted as

an increased diversity of the distribution: the more negative learning and the longer it takes place for, the more the single probabilities even out for a probability distribution over a finite domain, i.e. the more similar the distribution is to the uniform distribution.

Conclusion: The relevance of experiment II for the MAPAA is that negative learning over a finite domain has a diversifying effect on a probability distribution over this domain. In other words, negative learning can be seen as a diversification mechanism within the search.

5.1.2 Experiments III and IV: Learning over Interval Domains

While the experiments in the previous section dealt with learning over finite domains, we look here at the effects and results of positive and negative learning over interval domains. For both experiments, the studied domain $\mathbb{D}$ is the interval $[0, 1]$. In section 4.5.2, we described how a one-dimensional Self-Organising Map (SOM) can be used to model a probability distribution function $\mathcal{F}$ over $\mathbb{D}$. Here we use a chain with a fixed length of $L = 50$ neurons, the weights of which are denoted by $W_1, \ldots, W_{50}$. In equation 4.6c, the relationship between these neuron weights and a density function d, and hence $\mathcal{F}$, was established. The modification of the neuron weights follows

the rules as presented in formulas 4.17 and 4.18. They are, simplified,

$$k = argmin_{i=1}^{L}(|X - W_i|)$$
$$W_i^{new} = \begin{cases} W_i^{old} + \epsilon \cdot (X - W_i^{old}) & \text{if } |k - i| \leq \delta \\ W_i^{old} & \text{else} \end{cases}$$

in the case of positive learning, and

$$k = argmin_{i=1}^{L}(|X - W_i|)$$
$$W_i^{new} = \begin{cases} W_i^{old} + \epsilon \cdot (W_{k-\delta}^{old} - W_i^{old}) & \text{if } |k - i| \leq \delta \text{ and } W_i^{old} \leq X \\ W_i^{old} + \epsilon \cdot (W_{k+\delta}^{old} - W_i^{old}) & \text{if } |k - i| \leq \delta \text{ and } W_i^{old} > X \\ W_i^{old} & \text{else} \end{cases}$$

in the negative case. In both equations, $X \in \mathbb{D}$ is the learning input, ϵ is the learning rate, and δ is the size of the neighbourhood kernel used.

Experiment III: Positive Learning over Interval Domains

Objective of the experiment: The aim of experiment III, the counterpart to experiment I where we looked at positive learning over finite sets, is to show how the application of the rule for positive learning over an interval domain leads to the estimation of an unknown probability distribution through a one-dimensional SOM.

Detailed description of the experiment: The task in this experiment

is to learn to estimate the probability distribution function $\mathcal{F}^{target}$, which is given by its density function

$$d^{target}(x) = \begin{cases} 4 & \text{if } x \in [0.1, 0.2] \\ 6 & \text{if } x \in [0.8, 0.9] \\ 0 & \text{else} \end{cases},$$

through a chain of 50 neurons. Initially, these neurons are uniformly distributed over the interval $[0, 1]$. The one-dimensional SOM is then repeatedly modified during the experiment by learning from values X that are chosen randomly following the distribution $\mathcal{F}^{target}$. In the applied rule for positive learning, we use a neighbourhood kernel size of $\delta = 10$ and the learning rate $\epsilon = 0.05$.

Histograms of the neuron distribution over the domain $[0, 1]$, averaged over 10,000 simulation runs, are presented in figure 5.3. The initially uniform distribution of the neurons changes over time. The more learning steps that are performed, the more the neurons become concentrated around the sub-intervals $[0.1, 0.2]$ and $[0.8, 0.9]$. These are exactly the sub-intervals from which sample data X are drawn. The diagram also shows that the concentration of neurons in the interval $[0.8, 0.9]$ is higher than in $[0.1, 0.2]$.

As described in section 4.5.2, a high neuron concentration in a certain area leads to a high value of the density function in that area. Consequently, the neurons in this example define a density function d which yields high values in the intervals $[0.1, 0.2]$ and $[0.8, 0.9]$, furthermore higher values are reached

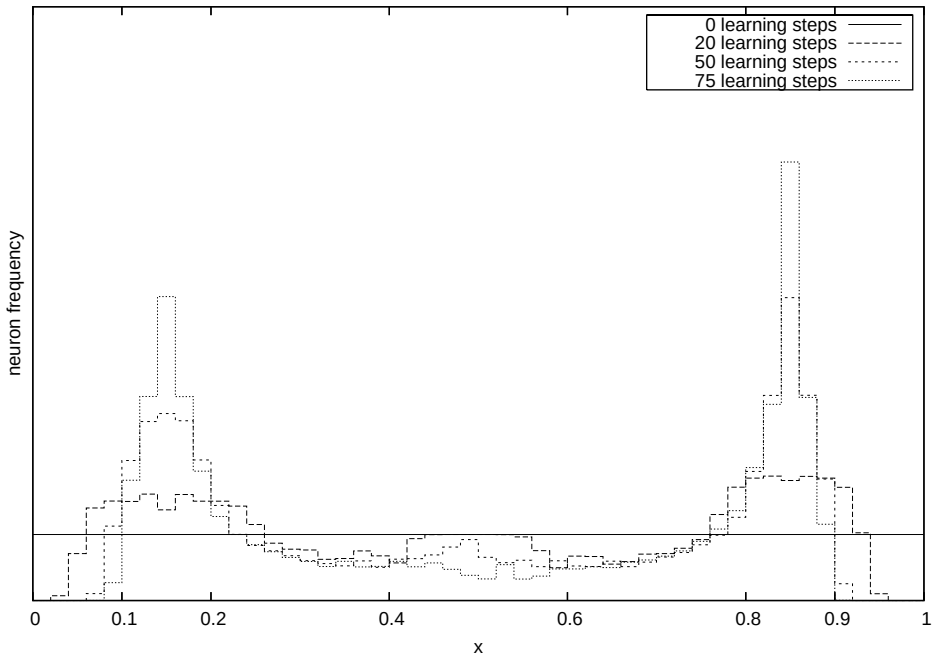

Figure 5.3: Neuron distributions after positive learning over an interval domain

in the latter interval, and much smaller values are reached elsewhere in the domain. These properties are demonstrated in figure 5.4.

The learnt density functions d are by no means exact matches of the target function d^{target}. The estimated density d, by its definition, cannot take the value zero over the domain, just small values. As a consequence, it cannot reach the values 4 and 6 over the whole intervals $[0.1, 0.2]$ and $[0.8, 0.9]$, respectively. Furthermore, high peaks in the middle of the two mentioned sub-intervals as well as comparatively small values at their boundaries are noticeable. The reason for these characteristics is that a learning input leads to the weight modification of multiple neurons. With a high neighbourhood kernel size, as used in this experiment, most of the neurons located around

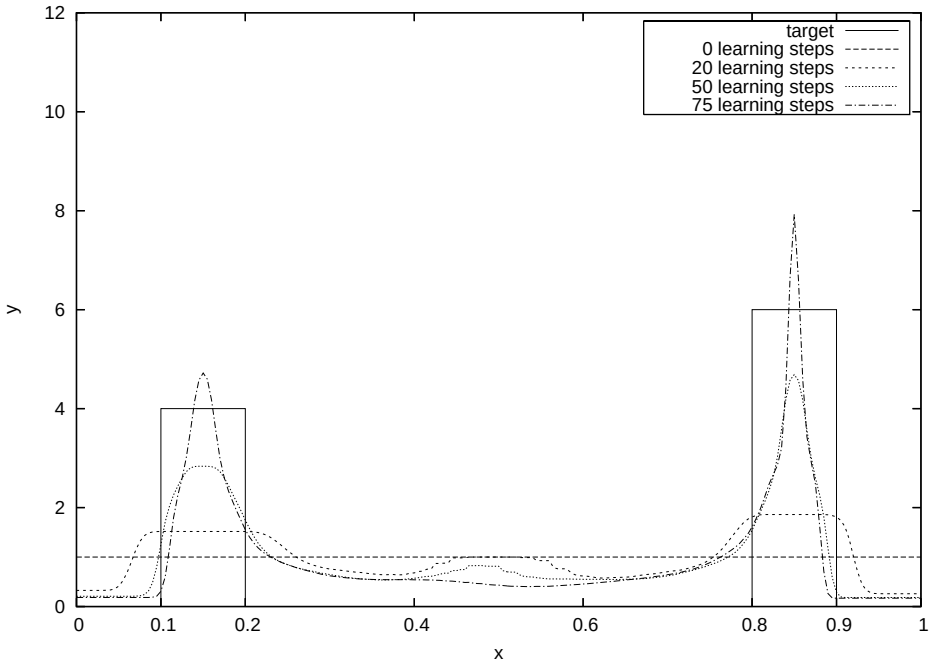

Figure 5.4: Density functions after positive learning over an interval domain

for instance $[0.1, 0.2]$ get affected by learning inputs from this interval. On average, they therefore tend to become concentrated around the middle of this area. In the original SOM algorithm, this phenomenon is tackled by learning rates and neighbourhood kernels that become smaller with time. Such a mechanism allows the neuron chain to settle into a more stable state. However, this mechanism also makes the neuron chain less able to adapt over time, i.e. it loses its ability to adapt quickly to changes in the distribution of input patterns. As a continuous capability to adapt to changes is highly desired in the MAPAA, for instance when the search encounters a new interesting sub-interval and reinforces those values, constant learning rates and neighbourhood kernel sizes are used in this approach.

Conclusion: The mechanism for positive feedback over interval domains

can learn to approximate the probability distribution of presented sample data. Although the learnt approximation is not an exact match of the target function, it is a sufficiently good estimation of the target for our purpose and shares the main characteristics of it. Its applicability and usefulness in the MAPAA will be the object of discussion in section 5.2.4 on experiment VI in this chapter.

Experiment IV: Negative Learning over Interval Domains

Objective of the experiment: With the fourth experiment, we show that the incorporation of negative learning over an interval domain has a diversifying effect. More specifically, we demonstrate that negative learning in a neuron chain leads to a more uniform distribution of the neurons over the domain, and hence to a more even density function over this interval.

Detailed description of the experiment: In this experiment, the $L = 50$ neurons of the neuron chain are initially distributed uniformly over the subinterval $[0.25, 0.75]$ of the domain $\mathbb{D} = [0, 1]$, i.e. they are located at the positions $0.255, 0.265, \ldots, 0745$[1]. The neuron weights therefore describe the

[1]The interval $[0.25, 0.75]$ is partitioned into 50 equal sized sub-intervals $[0.25, 0.26)$, $[0.26, 0.27)$ to $[0, 74, 0.75]$, and the neurons are located in the middle of these ranges.

density function[2]

$$d(x) = \begin{cases} \frac{1}{13} & \text{if } x \in [0.0, 0.26) \\ 2 & \text{if } x \in [0.26, 0.74) \\ \frac{1}{13} & \text{if } x \in [0.74, 1.0] \\ 0 & \text{else} \end{cases},$$

and hence a probability distribution function $\mathcal{F}$. During the experimental runs, values, that are drawn from $[0, 1]$ following $\mathcal{F}$, are used for the modification of the neuron chain. The modification is seen, with a certain percentage, as either positive or negative feedback.

The experiment shows that the application of negative learning forces the neurons to spread out over the interval $[0, 1]$, and has an equalising effect on the probability distribution. The higher the percentage of negative learning, and the more learning steps that are performed, the more the neurons become uniformly distributed over the whole of the domain $[0, 1]$. Figure 5.5, in which the resulting density functions are presented, confirms these observations.

A small, 20% chance of negative learning (and thus 80% positive learning) does not change the initial situation much after 100 learning steps, although even then equalising effects are observable on the borders of $[0.25, 0.75]$. If the negative learning percentage is increased to 60%, the density function values become smaller inside the interval $[0.25, 0.75]$, and larger in the remaining

[2]With regard to the domain $[0, 1]$, the first neuron at 0.255 is closest to all points in $[0, 0.26)$, the second neuron is closest to all points in $[0.26, 0.27)$, the third neuron attracts all neurons in $[0.27, 0.28)$, and so on. As the area under the density function must be $\frac{1}{L} = \frac{1}{50}$ for each sub-interval, the corresponding density function values are $\frac{1}{50} \cdot \frac{1}{0.26-0.0} = \frac{1}{13}$, $\frac{1}{50} \cdot \frac{1}{0.27-0.26} = 2$, $\frac{1}{50} \cdot \frac{1}{0.28-0.27} = 2$, and so on.

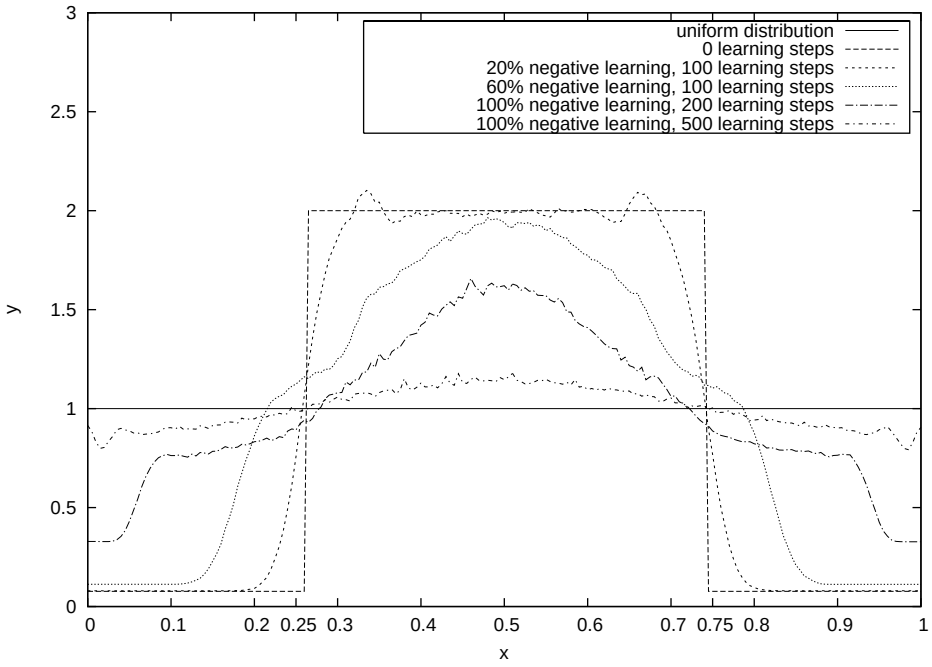

Figure 5.5: Density functions after positive and negative learning over an interval domain

part of $[0, 1]$. In case of pure negative learning, an almost uniform distribution over the domain is reached after 500 learning steps. To sum up, the more and the longer negative learning is performed, the more the estimated density function becomes evened out over the whole interval, i.e. the closer the probability distribution becomes to the uniform distribution.

Conclusion: In analogy to experiment II, the fourth experiment strongly suggests that negative learning over an interval increases diversity. Therefore, its application within the MAPAA can be seen as the addition of a diversification mechanism to the algorithm.

5.2 Experiments with the Multiple Algorithms' Parameter Adaptation Algorithm

5.2.1 Parameters of the Multiple Algorithms' Parameter Adaptation Algorithm

Population Size and Number of Generations

The Multiple Algorithms Parameter Adaptation Problem (MAPAP) demonstrates a number of characteristics that make it particularly difficult to tackle. These attributes were listed and discussed in section 1.5. Besides the black box character of the problem (C1), the difficulties with noise in the results of heuristic search methods (C3) and the unknown nature of the out-of-sample instances (C4), the limitation of available calculation time (C2) is the pivotal property of the MAPAP. The evaluation of individuals can be computationally very expensive. This high demand for computation power restricts the MAPAA both in the population size and the number of generational cycles that can practically be used.

In the original work introducing Population-Based Incremental Learning [Baluja, 1994], experiments were conducted using population sizes of 100 over $2,000$ generations. Those experiments thus required $200,000$ evaluations of population members which was manageable as these computations

were very fast. But in case of the MAPAP such high numbers of evaluations are not possible. As an example, a run of the Tabu Search algorithm for Vehicle Routing Problems, as described in the next chapter, can take up to 10 seconds to produce a reasonably good solution for a single instance on a standard office computer – this corresponds to one evaluation. Consequently, $200,000$ evaluations would take up to $33,333$ minutes, or 23 days.

To guarantee a complete run of the MAPAA in an acceptable and maintainable time frame, the population size as well as the number of generational cycles obviously must be limited to smaller values. In all experiments reported here, the population size M is therefore restricted to

$$M = 12, \tag{5.1}$$

and the number of generations is fixed at

$$G = 200. \tag{5.2}$$

The MAPAA with these settings still requires at least $2,600$ evaluations[3] which may take up to 7 hours in case of the previously described example. This computation time is considered affordable.

[3]In each cycle, all 12 population members and the best encountered individual are evaluated.

Parameters of the t-Test

Two further parameters have to be predefined for the *optimum* function in the MAPAA, as it was outlined in section 4.11. The first parameter α determines the significance level of the t-test, i.e. the level of confidence required for its result to be correct. The second parameter, min_{ttest}, defines the minimum number of data required. Both parameters can be freely modified to meet the user requirements when running the MAPAA. For all experiments reported in this document, these parameters were set to

$$\alpha = 2.5\% \tag{5.3}$$

and

$$min_{ttest} = 10. \tag{5.4}$$

Further Parameters

Four of the MAPAA parameters, as listed in section 4.13, have been discussed and have been set to fixed values. However, the eight remaining parameters are still undefined. They are studied in a number of experiments in this chapter in order to establish good operational settings.

5.2.2 The Testbed Algorithms

In the experiments described in this chapter, heuristic search algorithms are emulated by using artificially created objective functions that are very quickly computable. This allows the realisation of more versatile experiments as well as a better significance through larger test series. Although a much higher number of evaluations would be possible here, the restrictions on population size ($M = 12$) and generational cycles ($G = 200$) are always applied to guarantee a realistic emulation.

We define two pseudo heuristic search algorithms A_1 and A_2, each of which has ten parameters $\mathcal{P}_1, \ldots, \mathcal{P}_{10}$. The parameter domains are equal: $\mathbb{D}_1 = \ldots = \mathbb{D}_{10} = [0, 10]$. The objective functions of both pseudo algorithms depend only on the applied parameter settings, i.e no real optimisation problem instances are required. In addition, the objective functions include a random factor from $[0.95, 1.05]$ in each case to model the effect of noise.

The first objective function f_1 is given by

$$f_1(\mathcal{P}_1, \ldots, \mathcal{P}_{10}) = random(0.95, 1.05) \cdot (100 + \sum_{i=1}^{10} |\mathcal{P}_i - 1|).$$

All parameters contribute independently to the objective function. Without consideration of the random element, it can yield results from the range $[100, 190]$. Obviously parameter values closer to 1 lead, on average, to smaller and hence better objective function values, with $(1, \ldots, 1)$ being the optimal parameter setting. Within the MAPAA, we use $(10, \ldots, 10)$ as the initial

setting.

The second objective function is defined as

$$f_2(\mathcal{P}_1, \ldots, \mathcal{P}_{10}) = random(0.95, 1.05) \cdot (100 + \sum_{i=1}^{10} g(\mathcal{P}_i, \mathcal{P}_{i-1}, \mathcal{P}_{i+1}))$$

with

$$g(\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3)) = \begin{cases} -2 & \text{if } \mathcal{P}_2 \in [6,9] \text{ and } \mathcal{P}_3 \in [6,9] \\ |\mathcal{P}_1 - 1| & \text{else} \end{cases},$$

where we assume $\mathcal{P}_0 \equiv \mathcal{P}_{10}$ and $\mathcal{P}_{11} \equiv \mathcal{P}_1$. In contrast to the first function, the domain here is $[80, 190]$, again not taking the random factor into account. Although the parameter setting $(1, \ldots, 1)$ leads to a good average objective function value of 100, better results of 80 are reached if all parameters are chosen from the interval $[6, 9]$. Again, the parameter setting $(10, \ldots, 10)$ is used as the initial MAPAA solution.

Owing to the random factor in both functions, objective ratios for any two parameter settings follow a distribution that resembles a normal distribution. Therefore, the application of the t-test within our method is justified.

5.2.3 Experiment V: Learning Rates for Finite Domains

Objective of the experiment: In experiment V, we study how different rates for positive and negative learning over finite domains influence the

performance of the MAPAA.

Detailed description of the experiment: The MAPAA is applied to find good parameter settings for A_1 and A_2. The domains of the ten parameters are in each case discretised: $\mathbb{D}_1 = \ldots = \mathbb{D}_{10} = \{0, \ldots, 10\}$. As we are studying learning rates over finite domains, various combinations of positive and negative learning rates, as introduced in section 4.10.3, are considered. In detail, the positive learning rate ϵ^+_{fin} is chosen from the set $\{0.50, 0.30, 0.20, 0.10, 0.05, 0.00\}$ while its negative counterpart ϵ^-_{fin} is selected from the set $\{0.20, 0.10, 0.05, 0.02, 0.01, 0.00\}$. Results are averaged over 1,000 experimental runs, and the chance of mutation during the sampling process is $\mu = 0.05$.

To compare different sets of learning rates, we look at the development of the objective function value of the best found parameter setting over time. We consider a learning rate set better the smaller this value is after 200 generational cycles.

Several observations can be made when applying the MAPAA to algorithm A_1. The combination $\epsilon^+_{fin} = \epsilon^-_{fin} = 0.00$, i.e. a search without learning and thus purely random, yields the worst results. The inclusion of negative learning improves the quality of the solution. The higher the rate of negative learning is, the better the results become. However, these improvements are still comparatively poor.

Much better results are achieved if the MAPAA incorporates positive feed-

back. All tested positive learning rates yield reasonably good results when combined with small rates of negative learning $\epsilon^-_{fin} \leq 0.02$. Among these choices, non-zero values for ϵ^-_{fin} usually yield better results than MAPAA instances relying only on positive feedback. The application of higher ϵ^-_{fin}-values leads to a clear performance decline.

The correlations described above are illustrated in figure 5.6. The graph shows for $\epsilon^+_{fin} = 0.10$ combined with different rates of negative learning the development of the objective function value for the best found parameter setting over time, i.e. over generational cycles in the MAPAA. The combinations $\epsilon^+_{fin} = 0.10/\ \epsilon^-_{fin} = 0.02$ and $\epsilon^+_{fin} = 0.10/\ \epsilon^-_{fin} = 0.00$ outperform all other rates, with the former being marginally better. Increasing the influence of negative feedback to $\epsilon^-_{fin} = 0.05$, $\epsilon^-_{fin} = 0.10$ or even $\epsilon^-_{fin} = 0.20$ leads to a deterioration of the results.

The dominant influence of positive learning is expected. Each parameter contributes independently to the objective function, and the closer a parameter value is to 1 the better. The combination of a higher positive learning rate with a much smaller rate for negative learning leads to a fast concentration of the search on such values.

The picture changes slightly if the MAPAA is applied to find good parameter settings for algorithm A_2. Now, combinations that include higher amounts of negative feedback perform well and often yield better results. In fact, the pair $\epsilon^+_{fin} = 0.20/\ \epsilon^-_{fin} = 0.20$ performs best overall for A_2.

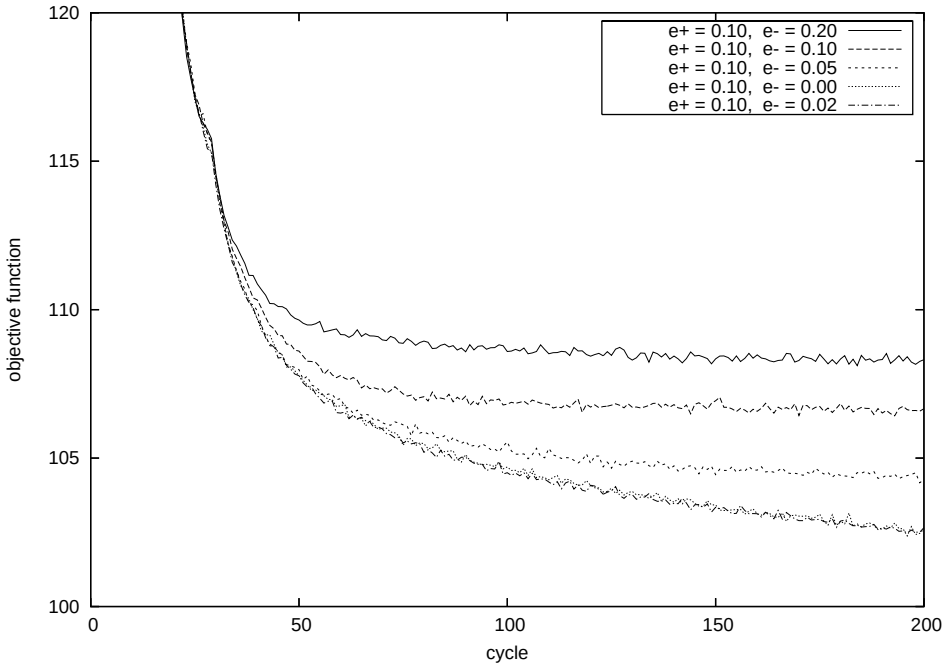

Figure 5.6: Development of f_1 for best parameter setting over time

In figure 5.7, we present the graphs of the objective function f_2 for the same set of learning rates as in the first part of the experiment. A positive learning rate of 0.10 is best paired with a negative learning rate of 0.02, but also rates of $\epsilon^-_{fin} = 0.05$ and $\epsilon^-_{fin} = 0.10$ yield good results. They outperform the configuration without negative learning. A greater focus on negative feedback with $\epsilon^-_{fin} = 0.20$ clearly shows the worst performance.

As in the case for the first algorithm, the absence of positive learning and the reliance on purely negative feedback does not perform well. But interestingly, the worst performance is achieved with the highest rate of positive learning, $\epsilon^+_{fin} = 0.50$, and no negative learning. However, an inclusion of negative learning at a small rate dramatically improves this performance.

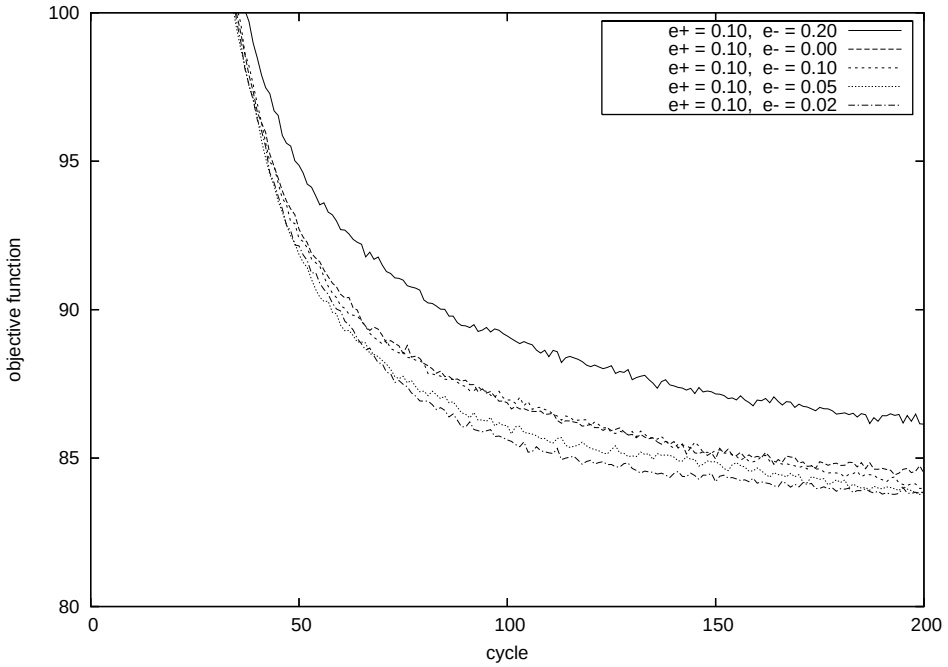

Figure 5.7: Development of f_2 for best parameter setting over time

These findings can be explained as follows. In case of only positive learning with a high rate of 0.5, the search becomes focused on the area around the parameter setting $(1, \ldots, 1)$ as in the experiment for f_1. Without diversification through negative learning, it becomes stuck there and is, in general, not able to efficiently explore other, better regions of the parameter space which are harder to find. If negative learning is included, the performance clearly becomes improved. But very good results are achieved with a more moderate rate of 0.10 for positive learning and a small rate of 0.02 for negative learning, as these choices allow the search to explore the parameter space more evenly without concentrating to quickly on specific areas.

Conclusion: To sum up, there is no straightforward best choice for the rates of learning over finite domains in the MAPAA. Depending on the scenario,

learning rates show different performances. However, experiment V suggests that the combination of a moderate positive learning rate of

$$\epsilon_{fin}^{+} = 0.10 \tag{5.5}$$

and a small negative learning rate of

$$\epsilon_{fin}^{-} = 0.02 \tag{5.6}$$

yields consistently good to very good results. The above values are therefore used in the remainder of experiments in this thesis.

5.2.4 Experiment VI: Learning Rates for Interval Domains

Objective of the experiment: In the sixth experiment, we look at the performance of the MAPAA when different learning rates and neighbourhood kernel sizes are used for interval domains.

Detailed description of the experiment: As in the previous experimental study, the MAPAA is applied to the algorithms A_1 and A_2 in order to determine good choices for the respective ten parameters. For both methods, the parameter ranges are modelled as real-valued intervals, i.e. the domains are implemented as $\mathbb{D}_1 = \ldots = \mathbb{D}_{10} = [0, 10]$. The length of the one-dimensional Self-Organising Maps, which are used for the characteri-

sation of probability distribution functions over these intervals, is fixed to $L = 50$. A multitude of learning rates and neighbourhood kernel sizes (see section 4.10.3) are tested. More specifically, the rate for positive learning ϵ_{int}^{+} is chosen from the set $\{0.5, 0.3, 0.2, 0.1, 0.05\}$ while the negative learning rate ϵ_{int}^{-} comes from the range $\{0.2, 0.1, 0.05, 0.01, 0.0\}$, with the corresponding neighbourhood kernel sizes δ^{+} and δ^{-} selected from $\{15, 10, 7, 4\}$ and $\{15, 10, 7, 5, 2\}$, respectively. All possible combinations are studied with results being averaged over $1,000$ runs. A mutation rate of $\mu = 0.05$ is employed in all cases.

The MAPAA's application to algorithm A_1 shows that positive learning is again decisive for a successful determination of good parameter settings. Generally, instances with the highest rate of positive learning $\epsilon_{int}^{+} = 0.50$ and the largest neighbourhood $\delta^{+} = 15$ yield supreme results. Scaling these values down leads to a deterioration of the performance.

This situation is illustrated in figure 5.8. It shows how the value of f_1 for the best found parameter setting develops over time. In all displayed learning configurations, the negative feedback is kept constant at $\epsilon_{int}^{-} = 0.10$ and $\delta^{-} = 2$. One can clearly observe that the $\epsilon_{int}^{+}/\delta^{+}$ pair 0.50/15 yields the best results, followed by 0.50/10, 0.20/15 and 0.50/7. The combination 0.10/15 yields the worst performance.

With the focus on positive learning, negative feedback plays a lesser role. Among the best performing learning schemes, no clear observations about good values for negative learning are possible. As positive feedback is domi-

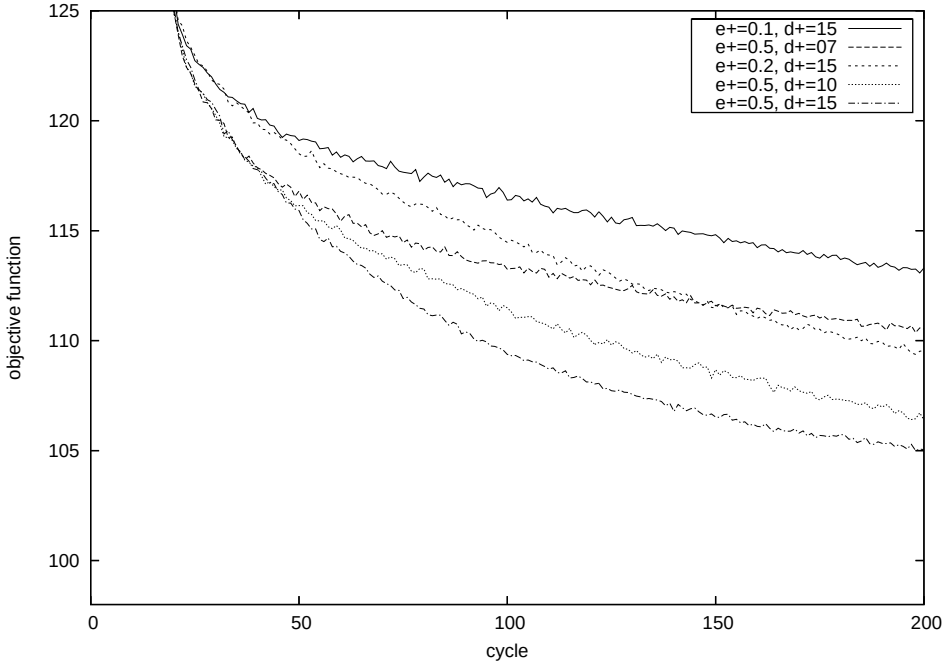

Figure 5.8: Development of f_1 for best parameter setting over time

nant, ϵ_{int}^{-} and δ^{-} seem to have little influence.

In analogy to experiment V, this behaviour is anticipated. As the ten parameters of A_1 contribute independently to the objective, no correlations between different parameters have to be detected and a high amount of positive learning is vital for the MAPAA performance.

The picture for algorithm A_2, however, is different. Here, the best results are achieved with more moderate amounts of positive feedback coupled with small amounts of negative feedback. Neither configurations with the highest degree of positive feedback $\epsilon_{int}^{+} = 0.50/\delta^{+} = 15$ nor schemes with high values of negative feedback, ϵ_{int}^{-}, are among the best learning procedures. The explanation for these observations lies again in the nature of A_2: a slower learning

from positive examples paired with the diversifying effects of small negative feedback allows the MAPAA to detect parameter relations efficiently.

Conclusion: It is not possible to specify an outright best learning scheme for learning over interval domains as the application is important for the performance of a configuration. However, the values

$$L = 50 \tag{5.7}$$

$$\epsilon_{int}^{+} = 0.50 \tag{5.8}$$

$$\delta^{+} = 10 \tag{5.9}$$

$$\epsilon_{int}^{-} = 0.10 \tag{5.10}$$

$$\delta^{-} = 2 \tag{5.11}$$

performed well for both cases presented, as well as in other experiments. Therefore, the MAPAA is used with the aforementioned choices in all future experiments.

5.2.5 Experiment VII: Mutation Rate

Objective of the experiment: The purpose of experiment VII is to look at the quality of solutions produced by the MAPAA for different degrees of mutation.

Detailed description of the experiment: We apply the MAPAA to both algorithms A_1 and A_2. In both cases, the first five parameters $\mathcal{P}_1$ to $\mathcal{P}_5$

are modelled as parameters over finite, discretised domains $\{0, 1, \ldots, 10\}$, while $\mathcal{P}_6$ to $\mathcal{P}_{10}$ have interval domains $[0, 10]$. All parameters of the MAPAA are fixed to the values previously discussed, only the mutation rate μ is varied between 0 and 1. The experiments are repeated $1,000$ times to obtain averaged results.

After running the MAPAA for $G = 200$ generational cycles, we determine the objective values produced by both algorithms with the respective best found parameter settings. In figure 5.9, this measure is shown for A_1 for mutation rates between 0.0 and 0.5. The best results are clearly achieved with mutation rates between 0.05 and 0.15. In this area, the objective values yielded are nearly constant. Smaller and larger amounts of mutation lead to a performance decline. Results for algorithm A_2 are qualitatively similar. There, mutation rates from the interval $[0.05, 0.20]$ produce the best results.

Conclusion: Our experiments demonstrate that choosing a moderate mutation rate between 0.05 and 0.15 yields consistently the best results. Therefore, a constant mutation rate of

$$\mu = 0.05 \tag{5.12}$$

is used for all further experiments reported.

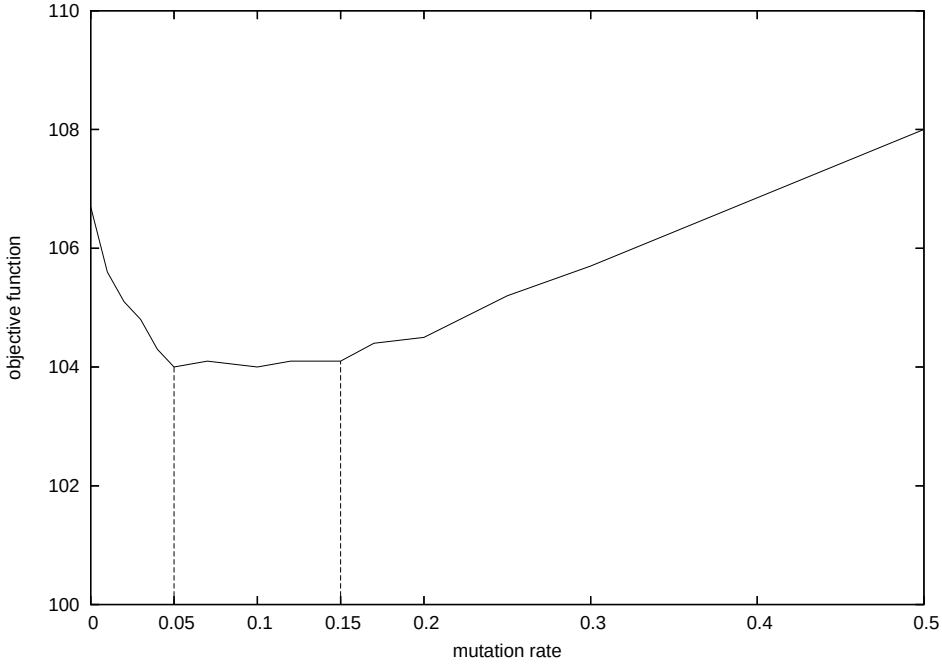

Figure 5.9: Relationship between solution quality and mutation rate

5.2.6 Experiment VIII: Distribution of Computation Resources

Objective of the experiment: In this penultimate experimental scenario, we study the distribution of resources if the MAPAA is applied to optimise the parameters of various algorithms simultaneously.

Detailed description of the experiment: As in scenario VII, this experiment is based on modelling the second algorithm with five finite and five interval parameter domains. However, A_2 was modified to A_2^k to the effect that $f_{A_2^k} = f_2 + k$. In other words, the objective function of A_2^k is the objective function of A_2, f_2, plus a constant k. In $1,000$ experimental runs, the MA-

PAA is applied to pairs of algorithms A_2 and A_2^k with $k \in \{0, 1, 3, 10, 20\}$. The focus of this experiment is on how our adaptation scheme distributes the available computation resources between such pairs of algorithms. The measure used here is the portion of the population that is occupied by the first of the two algorithms. In figure 5.10, the development of this measure is shown over generational cycles.

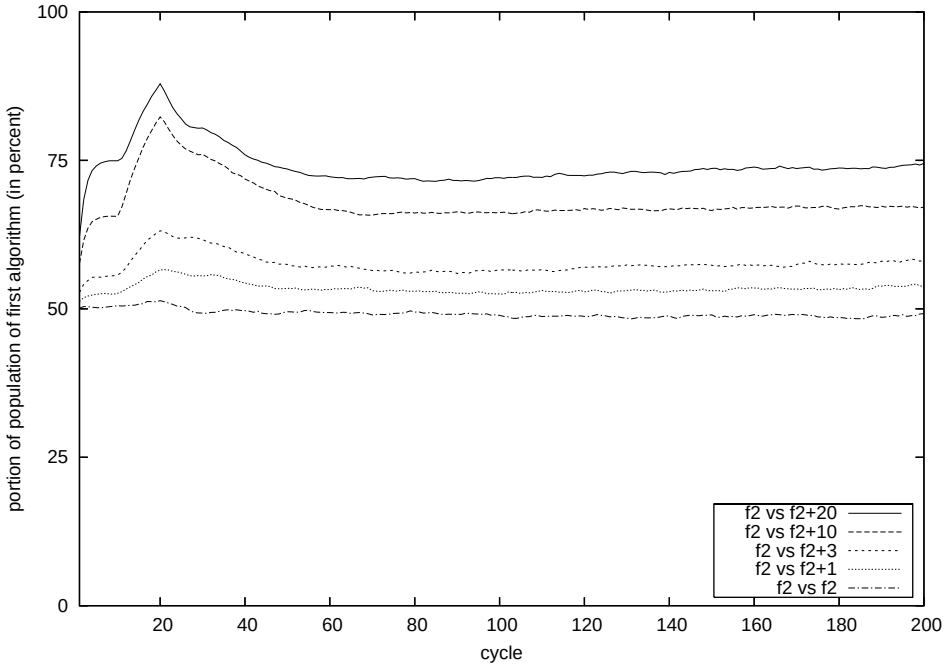

Figure 5.10: Distribution of Computation Resources showing successful algorithms have bigger share of the population

If A_2 competes with A_2^0, the MAPAA distributes its computation resources evenly between the two algorithms. On average, 50% of the population are composed of individuals representing A_2, and hence the remaining 50% represent the second method A_2^0. As both algorithms possess the same objective and hence produce the same results, resources are evenly shared.

But if A_2 competes with A_2^k, $k > 0$, then the former produces better, i.e. smaller, objective results. The larger k is, the larger is the performance superiority of A_2 over A_2^k. Graph 5.10 demonstrates that the MAPAA allocates more resources to the first algorithm, and thus less to the second, the larger the performance gap between the two methods is. As an example, on average 75% of the population represent A_2 and 25% A_2^{20} after cycle 50.

The figure also shows that these performance distributions are constant in the later part of MAPAA runs, in our scenario after cycle 50. In this later phase, an equilibrium between positive and negative feedback leads to an equilibrium in the algorithm distribution. However in the initial 50 cycles, positive learning is much more dominant as it is much easier to find improved parameter settings. This large amount of positive feedback, primarily for A_2, results in a peak of the population's portion that is assigned to the better performing algorithm.

Experiments with more than two competing algorithms produce similar results. Better performing methods get a bigger share of the population and hence computation resources, while less effective methods have less presence in the MAPAA populations.

Conclusion: Experiment VIII shows that our adaptation mechanism is capable of efficiently distributing computation resources between multiple algorithms. Promising, better performing methods are studied more intensely through an allocation of more resources.

5.2.7 Experiment IX: The Statistical Test

Objective of the experiment: The intention of the final experiment in this chapter is to show that the application of the t-test in the MAPAA is justified, i.e. that the data analysed with this statistical test show normal distribution like characteristics.

Detailed description of the experiment: The *optimum* function, as defined in section 4.11, is the point of the MAPAA where a t-test is employed to check whether the statistical data gained support the replacement of the best found parameter setting for an algorithm with the parameter setting represented by an individual within the population. In order to do so, it extracts past evaluation results from the evaluation histories of the individuals and examines a data series (here simplified)

$$\frac{f(A(\pi_1, \Phi_i))}{f(A(\pi_2, \Phi_i))}_4$$

for various different optimisation problems Φ_i, where π_1 represents the parameter setting of the individual from the population and π_2 the corresponding setting of the best individual found so far. If the t-test suggests that this series has a mean value smaller than one, then π_1 replaces the corresponding part in the best found individual.

The derivation of the t-test is based on the assumption that the analysed data are drawn from a normal distribution. However, as

[4] As a reminder, $f(A(\pi, \Phi))$ is the objective function value of the solution produced by algorithm A with the parameter setting π for the optimisation problem Φ.

[Bronstein and Semendjajew, 1991] point out, the t-test is not very sensitive to this condition. Because of its robustness, it can be applied as long as the frequency distribution of the data does not show multiple peaks and is not too skewed. We demonstrate these characteristics for the above described data series.

One experiment with data derived from the application of a Tabu Search algorithm with two different parameter settings[5] to $2,000$ VRP instances is presented here. In figure 5.11, a histogram with 50 bins showing the ratios of the derived objective function values is presented. Furthermore, the data are approximated by a Bezier curve, and a normal distribution with the mean at 0.96 and a standard deviation of 0.024 is shown.

Although both the Shapiro-Wilk W test [Royston, 1992] and the Anderson-Darling test [Stevens and D'Agostino, 1986] indicate that the considered data are not drawn from a normal distribution, it is clearly visible that both the Bezier and the normal distribution curve approximate the frequency distribution of the data well. Both curves have one peak at $x = 0.96$ and are symmetrical.

For this experiment, we can therefore state that the data are drawn from a distribution with normal distribution like characteristics. Further experiments with this and other heuristic search algorithms show similar results.

Conclusion: Our empirical study suggests that the application of the t-test

[5]The two parameter settings are $\pi_1 = (1, 7, 4, 1, 0.05.40, 5)$ and $\pi_2 = (1, 5, 1, 0, 0.00, 20, 0)$. Refer to section 6.3.2 and table 6.4 for details.

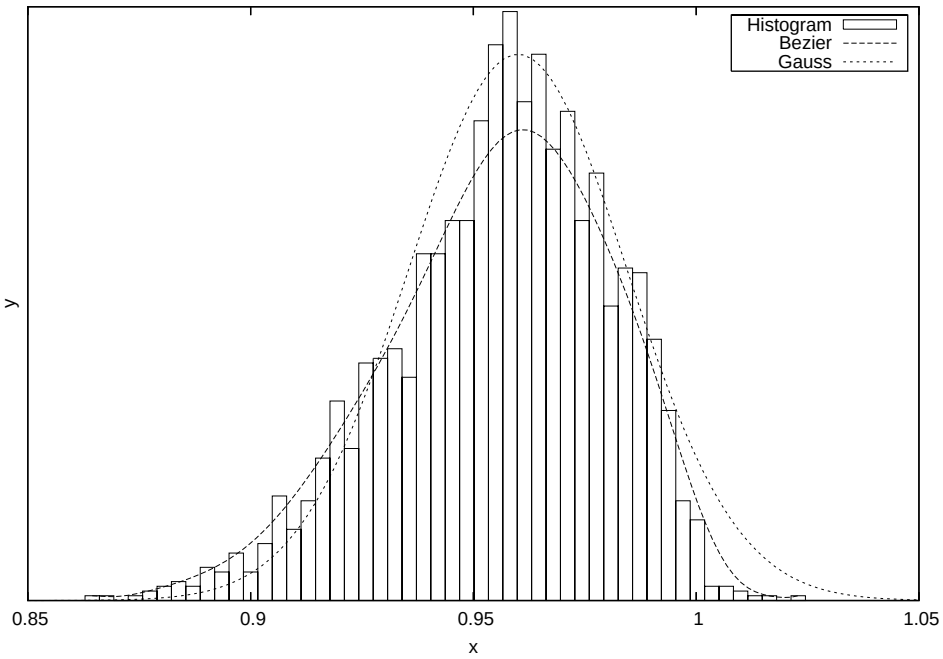

Figure 5.11: Histogram of Objective Function Ratios

in the MAPAA is justified. Because of its robustness, the t-test only requires data from a normal distribution like distribution with a single peak that is not too skewed, and the data analysed in this experiment satisfy these conditions.

5.3 Summary

In this chapter, we examined the Multiple Algorithms' Parameter Adaptation Algorithm (MAPAA) by means of experimentation. On one hand, we demonstrated how the rules for positive learning help to intensify the search around promising solutions. On the other hand, we illustrated how negative

learning adds a diversification mechanism to our approach. Furthermore, experiments for the determination of good learning schemes were carried out. Although it was not possible to find choices that perform constantly best, we were able to select robust, high performance learning rates. In addition, we showed how a moderate amount of mutation helps to improve the performance of the MAPAA. We also studied how our adaptation method balances its computation resources if applied to several algorithms simultaneously. We highlighted that the efficient strategy used distributes more computation time to better performing methods. Finally, we provided experimental proof for the validity of utilising the t-test. Table 5.1 provides an overview of these conclusions.

As a result of our experimental study, we have determined a set of good choices for the parameters of the MAPAA. All further experiments reported in this thesis are based on these values that are summarised in table 5.2.

Exp.	Conclusion	Section
I	Positive learning over finite domains leads to the estimation of a target probability distribution.	5.1.1
II	Negative feedback brings a diversifying element to the learning process over finite domains.	5.1.1
III	Positive learning over interval domains leads to the estimation of a target probability distribution.	5.1.2
IV	Negative feedback brings a diversifying element to the learning process over interval domains.	5.1.2
V	The combination of a moderate positive learning rate $\epsilon^+_{fin} = 0.10$ and a small negative learning rate $\epsilon^-_{fin} = 0.02$ yields consistently good results for finite domains.	5.2.3
VI	A Self-Organising Map with a chain of $L = 50$ neurons, learning rates of $\epsilon^+_{int} = 0.50$ and $\epsilon^-_{int} = 0.10$, and neighbourhood kernels of size $\delta^+ = 10$ and $\delta^- = 2$ performs well as a learning scheme over interval domains.	5.2.4
VII	Choosing a moderate mutation rate between 0.05 and 0.15 yields consistently the best results.	5.2.5
VIII	The adaptation mechanism efficiently allocates more resources to better performing algorithms in case of multiple methods.	5.2.6
IX	The application of the t-test is justified.	5.2.7

Table 5.1: Conclusions of experiments conducted with the Multiple Algorithms' Parameter Adaptation Algorithm

MAPAA Parameter	Symbol	Value
number of generational cycles	G	200
population size	M	12
mutation rate	μ	0.05
positive learning rate, finite domain	ϵ^+_{fin}	0.10
negative learning rate, finite domain	ϵ^-_{fin}	0.02
number of neurons for the SOM model	L	50
positive learning rate, interval domain	ϵ^+_{int}	0.50
negative learning rate, interval domain	ϵ^-_{int}	0.10
neighbourhood kernel size, positive learning	δ^+	10
neighbourhood kernel size, negative learning	δ^-	2
significance level of the t-test	α	2.5%
minimal number of sample data required by t-test	min_{ttest}	10

Table 5.2: Parameter values of the Multiple Algorithms' Parameter Adaptation Algorithm

Chapter 6

Applications of the Multiple Algorithms' Parameter Adaptation Algorithm

The purpose of this chapter is to demonstrate the efficiency and robustness of the Multiple Algorithms' Parameter Adaptation Algorithm (MAPAA). It therefore presents an overview of seven applications. In five of these scenarios, the parameters of a single heuristic search are adapted, while the two remaining applications address cases where the MAPAA is applied to two heuristic search algorithms simultaneously. At the end of this chapter, we examine a MAPAA implementation in the commercial iOpt toolkit and conclude with a summary.

6.1 Preliminary Notes

6.1.1 Implementation Details

All heuristic search methods covered in this chapter are implemented in the JAVA programming language. The experiments reported were performed on Intel processor-based 2.0 GHz personal computers. Depending on the application scenario, the heuristic search algorithms took between 10 and 25 seconds to produce a solution for a single optimisation problem instance.

6.1.2 Size of the Search Space

The aim of the MAPAA is to find good parameter settings from the parameter spaces of the studied heuristic search methods. Therefore, these parameter spaces constitute the search space of our adaptation approach. If such a space takes the shape $\mathbb{D}_1 \times \ldots \times \mathbb{D}_n$, then its size can be calculated by $\prod_{i=1}^{n} |\mathbb{D}_i|$.

Table 6.1 provides an overview of the encountered search space sizes for all seven applications studied in this chapter. Details of these spaces and the respective parameters are discussed in sections 6.2 to 6.4. Here, we assume that continuous domains are discretised with 11 sample values[1]. The table shows that the sizes of the search spaces range from $5,880$ possibilities in Application III to $1,023,660$ in Application VII.

[1] As an example, the interval $[0, 1]$ can be discretised as $\{0.0, 0.1, \ldots, 0.9, 1.0\}$.

Application	Search Space Size
I	$3 \cdot 7 \cdot 6 \cdot 11 \cdot 11 = 15,246$
II	$3 \cdot 7 \cdot 5 \cdot 2 \cdot 11 \cdot 7 \cdot 7 = 113,190$
III	$3 \cdot 7 \cdot 5 \cdot 2 \cdot 2 \cdot 7 \cdot 2 = 5,880$
IV (II+III)	$113,190 + 5,880 = 119,070$
V	$3 \cdot 4 \cdot 5 \cdot 5 \cdot 11 \cdot 11 \cdot 2 \cdot 2 = 145,200$
VI	$3 \cdot 2 \cdot 11 \cdot 10 \cdot 11 \cdot 11 \cdot 11 = 878,460$
VII (V+VI)	$145,200 + 878,460 = 1,023,660$

Table 6.1: Search space sizes in the Applications I to VII

6.1.3 Infeasibility of Exhaustive Search

In all seven applications in this chapter, an exhaustive search (in which each point is tested) for the best parameter setting is usually not possible due to the sheer size of the parameter spaces. Taking into account the noise factor in the results of heuristic search methods[2], each point would have to be tested for a number of different optimisation problem instances in order to get averaged results.

We assume here an averaging over just 10 problem instances. In Application III, with the smallest search space size of $5,880$, a single heuristic search call requires 10 seconds. Hence, an exhaustive search would run for $5,880 \cdot 10 \cdot 10 = 588,000$ seconds, or 7 days. For Application V, where a single heuristic search instance runs for 25 seconds, a complete search would last for $145,200 \cdot 10 \cdot 25 = 36,300,000$ seconds, or 420 days. Even the smallest computation time of one week can rarely be justified and is usually too high. More selective and thus faster search strategies are required. We propose the

[2]See characteristic C3 in section 1.5.

MAPAA as such an alternative strategy.

6.1.4 Running Time of the MAPAA

In all experiments, the MAPAA runs for $G = 200$ generations. In each cycle, all $M = 12$ parameter settings present in its population as well as the best encountered parameter settings for the m considered heuristic search methods get evaluated. In the single algorithm scenarios (Applications I, II, III, V and VI) m equals 1, and in the two multiple algorithm scenarios (Applications IV and VII) m is 2. The MAPAA therefore tests $G \cdot (M + m)$ parameter setting, which adds up to $2,600$ or $2,800$ evaluations.

The running time of the MAPAA predominantly depends on the required computation time for these evaluations. In Applications I to IV, where such a heuristic search takes between 10 to 15 seconds, a single MAPAA run lasts between 7 and 12 hours. For the remaining Applications V, VI and VII, where a heuristic search call is computationally more expensive, a single MAPAA instance can run for up to 20 hours.

Although the computation time resources required by the MAPAA are still high, they are much smaller than those required by a complete search. In our experiments, this improvement in running time was at least of one magnitude[3] and in general of two magnitudes and higher. This makes the computation demands much more manageable and, we believe, affordable. These

[3]A magnitude of one is considered as factor 10.

demand still, however, limit the number of independent MAPAA runs for each of the seven applications covered in this chapter. In the first four scenarios I to IV, results are reported from 20 individual MAPAA runs for each case. For the more time demanding Applications V to VII, the number of MAPAA runs is set to 16.

6.1.5 Fraction of the Search Space explored by the MAPAA

During one MAPAA run, $G \cdot M = 200 \cdot 12 = 2,400$ evaluations are performed for individuals from the population. That means that our parameter adaptation approach can explore at most $2,400$ different parameter settings. But well performing configurations are often repeatedly tested. In fact, a minimal number of $min_{ttest} = 10$ evaluations is required for a setting to be considered as an improvement by the statistical test within the MAPAA. Consequently, significantly less than $2,400$ parameter settings are explored during the search. The usual magnitude is from a few hundred up to one thousand.

To sum up, the MAPAA can explore only a small fraction of the search space. However, the search focuses on promising areas and leads, as we show in this chapter, to good and robust results.

6.1.6 A Measure for the Quality of Solutions

The MAPAA is a solution algorithm for the Multiple Algorithms' Parameter Adaptation Problem (MAPAP)[4]. To briefly recapitulate, the aim is to find the algorithm A^{best} among the algorithms $A_1, \ldots, A_m$, together with the corresponding best parameter setting π^{best}, such that the sum

$$f_{avg} = \frac{1}{k'} \sum_{i=1}^{k'} f(A^{best}(\pi^{best}, \Phi'_i)) \tag{6.1}$$

is minimal. During the solution process, the out-of-sample problems $\vec{\Phi}' = (\Phi'_1, \ldots, \Phi'_{k'})$ are not available, and only information gathered by applying the heuristic search algorithms to sample problems $\vec{\Phi} = (\Phi_1, \ldots, \Phi_k)$ can be used.

The sum given in equation 6.1 is used as the criterion to judge and compare the quality of solutions yielded by the MAPAA. In other words, if our adaptation approach returns algorithm A with parameter setting π as its result, then the average objective function value f_{avg} produced by this specific algorithm configuration for given out-of-sample problems $\vec{\Phi}'$ is the decisive measure to assess the solution quality.

[4]See section 1.4 for the formal definition.

6.1.7 Single Parameter Optimisation

Having defined a measure for comparing the quality of candidate solutions for the MAPAP with equation 6.1, one important question remains unanswered: What is the optimal value for f_{avg} for a given vector of test optimisation problems $\vec{\Phi}'$? As explained earlier, an exhaustive search is, in practical terms, not possible. Moreover, a detailed analysis of the search space landscapes is too big an issue to be addressed in this thesis. We describe therefore a basic statistical approach, which we refer to as Single Parameter Optimisation (SPO), and use its results as reference values to judge the efficiency of the MAPAA.

The SPO approach for tuning the parameters of a heuristic search, and thus improving its performance, is to optimise the parameters one by one. This procedure starts with all parameters set to initial values. One parameter is chosen, either randomly or following a predefined order, and is varied while all other parameters are kept constant. This allows the method to find a good value for this single parameter. Once a good value is found, the initial setting is modified in order to incorporate this setting. This process is repeated for all parameters.

If all parameters contribute independently to the performance of the heuristic search algorithm, then SPO is an effective way to determine good settings. As the quality of a parameter choice does not depend on other parameter values in this case, optimising each parameter individually yields very good results.

However, parameters are often not independent of each other. Rather there are inter-dependencies which influence the quality of the heuristic search. Such links between parameters are often not obvious, and expert knowledge is required to understand and utilise them efficiently. As a consequence, SPO is less effective in such cases. Furthermore, the order parameters are optimised in is often crucial for the quality of the settings found, as is the initial configuration SPO starts with.

For a search space $\mathbb{D}_1 \times \ldots \times \mathbb{D}_n$, SPO examines $|\mathbb{D}_i|$ different settings during the optimisation of the i-th parameter. Overall, it explores $\sum_{i=1}^{n} |\mathbb{D}_i|$ different parameter configurations. To counter the interfering influence of noise, each such setting is tested for a number of optimisation problems. In Applications I, II and III, we average results over 100 problem instances, while 50 instances are used for Applications V and VI[5]. Hence SPO requires between $2,150$ (Application V) and $4,200$ (Application II) evaluations, which is the same magnitude as required by the MAPAA.

6.1.8 Experimental Setup

For each of the seven applications presented in sections 6.2 to 6.4, we provide the following information:

- a description of the class of optimisation problems used,

[5]No SPO runs are required for Applications IV and VII. These are scenarios where two heuristic search methods are adapted simultaneously, and the SPO results are the same as for Applications II/III and V/VI, respectively.

- a specification of the applied heuristic search method (including details about its parameters, its parameter domains and its initial parameter choices),
- the result of an SPO run (which starts with the specified initial parameter setting) for the heuristic search method,
- the results of 16 or 20 independent MAPAA runs (which again use the the specified initial parameter setting) for the heuristic search,
- an analysis and comparison of the achieved results, and
- conclusions drawn.

6.2 An Application for the Travelling Salesman Problem

In this section, we discuss the results produced by the MAPAA when it is applied to a Simulated Annealing algorithm for the Travelling Salesman Problem.

6.2.1 The Travelling Salesman Problem

The Travelling Salesman Problem (TSP) is possibly the most well known optimisation problem. In the classic symmetric TSP, a number of N

cities and their respective, symmetric distances are given. The aim is to find the shortest tour that visits each city exactly once. This NP-hard problem has attracted a huge amount of research interest. Many different optimisation techniques have been applied to this problem scenario including Tabu Search [Zachariasen and Dam, 1995], Simulated Annealing [Boettcher and Percus, 1998, Moldover and Coddington, 1994], Genetic Algorithms [Whitley and Mathias, 1992] and Guided Local Search [Voudouris and Tsang, 1999]. [Johnson and McGeoch, 1997] is a comprehensive survey of solutions methods for the TSP.

The TSPLIB library [Reinelt, 1995] is a collection of TSP instances of various nature. However, as the Multiple Algorithms' Parameter Adaptation Algorithm (MAPAA) requires a large number of problem cases to run, we create random scenarios with the following characteristics:

- There are between 100 and 200 cities to visit.
- Each city is situated in the square $[0, 1] \times [0, 1]$.
- The Euclidean distance is used as the distance measure.

6.2.2 Application I: A Simulated Annealing Method for the TSP

General Concept

Simulated Annealing (SA) belongs to the class of local or neighbourhood search techniques. [Russell and Norvig, 2003] describes these techniques as "algorithms [which] operate using a single current state ... and generally move only to neighbours of that state". The underlying principle is that of improving an initial, often randomly created single solution by modifying it in small, local steps. The set of all solutions that can be reached from the current state by a single modification step is called the neighbourhood of this state. Therefore, local search is the repeated replacement of the current solution by a solution from its neighbourhood, until a termination criterion is satisfied.

The process of selecting a member from the neighbourhood is central for the success of a local search method. A basic approach is hill-climbing, where only improving states from the neighbourhood, i.e. states with better objective values than the current solution, are considered for selection, until the neighbourhood does not contain any more improving states. Hence hill-climbers risk the search to get trapped in suboptimal solutions. Various techniques have been proposed to overcome this shortage.

SA is based on a strategy analogous to the physical process of annealing of

solids. In this process, a heated metal is cooled down. A drastic cooling would lead to the formation of random, high energy states of the atoms. A slow cooling, however, allows the atoms to settle in more stable patterns, which results in a low-energy crystalline state.

[Kirkpatrick et al., 1983] were the first to propose SA as a general optimisation method. During the search, a state s' from the neighbourhood of the current solution s is randomly selected. If s' is a superior solution, $f(s') \leq f(s)$, the search moves from state s to s'. In case of an inferior solution s', $f(s') > f(s)$, this move is only accepted with probability $e^{\frac{-\Delta f}{T}}$. The parameter T is referred to as temperature. For higher temperatures, this scheme accepts many non-improving steps, while for lower temperatures the majority of such moves are rejected. During the search process, an initially high temperature is gradually lowered. As a result, the search moves towards better solutions but avoids getting trapped in local minima. Only in the final stage the search settles in a good solution. Central to the performance of SA is the way the temperature is lowered. Many different annealing schedules have been proposed. The interested reader may refer to [Osman, 1995] for a classification.

SA has been applied very successfully to a wide range of optimisation problems. Examples include Very Large Scale Integration layout problems [Chandy et al., 1997] and Vehicle Routing [Osman, 1993].

Simulated Annealing for the TSP

Our implementation of a Simulated Annealing method for the TSP is loosely based on the basic technique outlined in [Moldover and Coddington, 1994]. This algorithm has five parameters which are discussed below.

Parameter $\mathcal{P}_1$ - number of restarts: The complete search process involves the selection of a total of $2,000,000$ random moves. This move number can be evenly divided between 1, 2 or 5 restarts of the SA algorithm.

Parameter $\mathcal{P}_2$ - neighbourhood operators: A solution for the TSP is represented as a permutation of $(1, \ldots, N)$. A neighbouring solution can be reached by applying one of three possible operators: swap, reinsert or reverse. As an example, if we are given the solution $(1, 2, 3, 4, 5, 6)$, then a swap of the cities 2 and 5 results in the route $(1, 5, 3, 4, 2, 6)$, a reinsertion of 2 after 5 leads to $(1, 3, 4, 5, 2, 6)$, and a reversal of the cities between 2 and 5 yields the path $(1, 2, 4, 3, 5, 6)$. The second parameter decides whether one, two or all three of these operators are applicable. The possible values 1 to 7 correspond to 1=swap, 2=reinsert, 3=reinsert+swap, 4=reverse, 5=reverse+swap, 6=reverse+reinsert and 7=reverse+reinsert+swap.

The parameters $\mathcal{P}_3$, $\mathcal{P}_4$ and $\mathcal{P}_5$ define a stepwise temperature reduction annealing schedule. At the start of the SA process, the initial and the final temperatures have to be determined. To do so, the search accepts initially all random moves for a while and observes the average (Δ_{avg}) and the minimal (Δ_{min}) objective difference of all non-improving moves. Given parameter $\mathcal{P}_4$,

the **initial acceptance probability**, the initial temperature is set to a value which accepts a non-improving move with the average objective difference of Δ_{avg} with probability $\mathcal{P}_4$[6]. Analogically, based on the parameter $\mathcal{P}_5$, the **final acceptance probability**, the final temperature is calculated by accepting non-improving moves with objective differences of Δ_{min} with probability $\mathcal{P}_5$. During the annealing process, the temperature remains constant for a number of steps before it is reduced. This number is controlled by parameter $\mathcal{P}_3$, the **thermalisation**. At each reduction step, the temperature is reduced according to the geometric cooling rule $T = T \cdot \alpha$ with $\alpha < 1$. Knowing the initial and final temperatures, the total number of neighbourhood moves and the thermalisation, α can be easily calculated.

Table 6.2 summarises the five parameters and their domains as used in our experiments. The initial parameter choices are highlighted.

Parameter	Description	Domain
$\mathcal{P}_1$	number of restarts	$\{\mathbf{1}, 2, 5\}$
$\mathcal{P}_2$	neighbourhood operators	$\{\mathbf{1}, 2, 3, 4, 5, 6, 7\}$
$\mathcal{P}_3$	thermalisation	$\{1, \mathbf{10}, 100, 1000, 10000, 100000\}$
$\mathcal{P}_4$	initial acceptance prob.	$[0.01, 0.99]$ $(\mathbf{0.5})$
$\mathcal{P}_5$	final acceptance prob.	$[0.01, 0.99]$ $(\mathbf{0.5})$

Table 6.2: Parameters of a Simulated Annealing algorithm for the TSP

[6]In order to determine the initial temperature $T_{initial}$, the equation $e^{\frac{-\Delta_{avg}}{T_{initial}}} = \mathcal{P}_4$ has to be solved.

Experimental MAPAA Results

Both SPO and the MAPAA are applied to the aforementioned Simulated Annealing algorithm. The SPO process starts with the initial setting $(1, 1, 10, 0.5, 0.5)$ and optimises the parameters in the order given in table 6.2. In each step, 100 TSP instances are solved by the heuristic search to establish the best parameter choices. The result produced is the vector $(1, 6, 10, 0.5, 0.99)$, i.e. the configuration with 1 restart, neighbourhood 6, a thermalisation of 10, and initial and final acceptance probabilities of 0.5 and 0.99, respectively. A full account of the numerical results is presented in appendix D.1.

Furthermore, 20 MAPAA runs are performed. For each MAPAA instance, a different vector of 200 sample TSP problems is used. Table 6.3 lists the parameter choices found in each of these 20 runs alongside the initial parameter setting and the setting yielded by SPO. The last column of the table shows the average objective achieved by the SA algorithm with the respective settings for 100 out-of-sample[7] TSP instances.

- For 13 out of 20 runs, the MAPAA produces a solution that suggests 1 restart. In the remaining 7 cases, 2 restarts are favoured.
- In all 20 runs, the neighbourhood 6, i.e the use of the reverse and reinsert neighbourhood operators, is found to be the best choice.

[7]The TSP instances used during the MAPAA runs are different from these test problems. Therefore, the former are called sample problems while the latter are referred to as out-of-sample problems.

Parameter Setting	$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_5$	f_{avg}
Initial	1	1	10	0.50	0.50	11.99
SPO	1	6	10	0.50	0.99	9.38
MAPAA #1	2	6	10000	0.24	0.97	9.38
MAPAA #2	2	6	1	0.11	0.74	9.40
MAPAA #3	1	6	10	0.41	0.94	9.39
MAPAA #4	1	6	100	0.04	0.88	9.38
MAPAA #5	1	6	10000	0.13	0.97	9.38
MAPAA #6	1	6	10000	0.43	0.62	9.41
MAPAA #7	1	6	10	0.15	0.76	9.40
MAPAA #8	1	6	100	0.41	0.83	9.40
MAPAA #9	1	6	10000	0.40	0.86	9.40
MAPAA #10	1	6	10	0.47	0.97	9.40
MAPAA #11	1	6	100	0.03	0.90	9.39
MAPAA #12	1	6	10000	0.32	0.76	9.42
MAPAA #13	1	6	100	0.04	0.91	9.39
MAPAA #14	2	6	100	0.02	0.94	9.37
MAPAA #15	2	6	1	0.52	0.86	9.41
MAPAA #16	2	6	10000	0.12	0.62	9.41
MAPAA #17	1	6	10000	0.24	0.59	9.42
MAPAA #18	2	6	1000	0.15	0.95	9.37
MAPAA #19	2	6	1000	0.25	0.95	9.39
MAPAA #20	1	6	100	0.18	0.95	9.40
f_{avg}: $mean = 9.396$, $\sigma = 0.014$						

Table 6.3: Experimental results for a Simulated Annealing algorithm for the TSP

- No clear statement can be made about the thermalisation parameter. Except for the largest value of 100000, all possible choices can be found in solutions produced by the MAPAA.

- The results achieved for the initial and the final acceptance probabilities are consistent. While smaller values from $[0.02, 0.52]$ are chosen for the former parameter, the latter is always selected from $[0.59, 0.97]$.

The parameter settings produced by the MAPAA are comparable to the solution produced by the SPO approach. The only exception is the restart parameter. SPO strongly favours the value 1, but the MAPAA often finds 2 to be efficient as well.

The similarity of the parameter settings produced by SPO and MAPAA is mirrored by their performance. For 100 VRP instances, the MAPAA solutions and the SPO result produce very similar average objective values. In few cases, the MAPAA solution is slightly but not significantly better, but the majority of the results are equivalent. Only the parameter setting found by MAPAA run #17 performs worse than the single optimisation setting at a 1% significance level. Furthermore, all produced parameter settings dramatically improve the initial solution that is used as a starting point for both SPO and MAPAA runs.

The experimental results suggest that the right choice for the neighbourhood parameter is central for the SA performance. In all 20 MAPAA runs, the choices 6 or 7 are proven to be better than the initial choice of 1 after only 10 generational cycles. After 60 MAPAA generations at most, this choice is narrowed down to the best value of 6. That means that the MAPAA is able to very rapidly find high performance parameter settings and prove their superiority.

Conclusion

For the Simulated Annealing mechanism discussed, SPO and the MAPAA yield equivalent results. Because of the simple parameter structure of the heuristic search, both methods have no difficulties in finding good parameter settings. Our population-based adaptation approach concentrates its effort very quickly on the relevant areas of the parameter space, and therefore produces solutions with a proven high performance after few generational cycles.

6.3 Applications for the Vehicle Routing Problem

Three applications of the MAPAA are presented in this section. After applying our adaptation approach individually to a Tabu Search method and a Genetic Algorithm for Vehicle Routing, we test its efficiency when applied to both aforementioned heuristic search techniques simultaneously.

6.3.1 The Vehicle Routing Problem

The Vehicle Routing Problem (VRP), in its most basic form and as it is used here, is the task of serving a number of customers through a fleet of vehicles. These vehicles are based at one single depot and must be routed in order to

fulfill demands for certain amounts of goods by the customers. The routing is constrained by the amount of goods one vehicle can carry, and the maximal length of a lorry route. The most common objective is to serve all customers, i.e. to find a set of routes for the fleet that visit all customers, with minimal vehicle travelling distance while taking account of both the capacity and the distance constraints for the vehicles.

A candidate solution for the VRP can be represented as the assignment of routes, i.e. sequences of customers to visit, to all vehicles. These routes must cover every customer exactly once. If all routes fulfill both capacity and distance constraints, we speak of a legal solution, otherwise it is called illegal.

Example 6.1. An example VRP instance is presented in figure 6.1. A fleet of three vehicles $V1$, $V2$ and $V3$ are based at the depot D. The task is to serve 10 customers $C1$ to $C10$, the demand of which is exactly 1 in this example. The figure also shows one possible set of routes for the vehicles. The first vehicle $V1$ serves the customer $C4$, $C2$ and $C5$ in that order, while $V2$ serves $C10$, $C9$, $C1$ and $C3$, and $V3$ travels to $C7$, $C6$ and $C8$. As the number beside an arrow corresponds to the traveling distance, vehicle $V1$ would travel a distance of 10 and would need a capacity of 3, vehicle $V2$ would drive a distance of 12 and would need a loading capacity of 4, and vehicle $V3$'s traveled distance would be 11 with a required capacity of 3.

[Beasley, 1990] provides several sets of VRP instances, among others 14

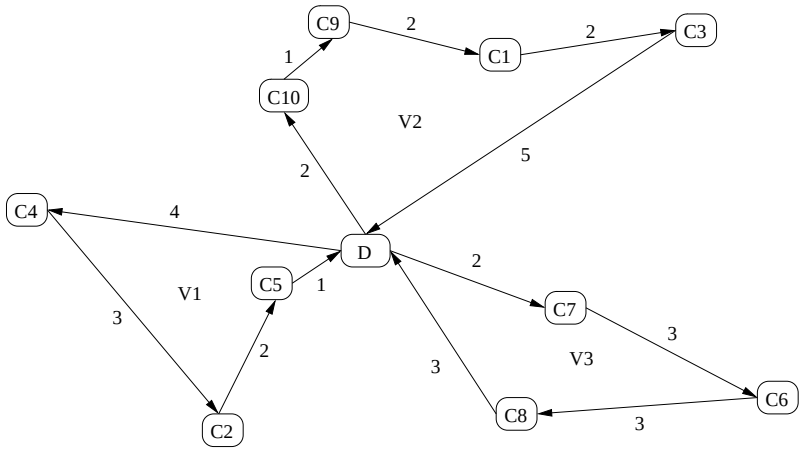

Figure 6.1: A Vehicle Routing Problem

VRPNC problems. In order to test the MAPAA efficiently, a larger number of problem scenarios is required. Therefore, further instances similar to these VRPNC problems are randomly created. They all show the following characteristics:

- The single depot is located at position $(40, 40)$.
- The 50 to 75 customers have a demand between 1 and 40 each, and are located inside the square $[0, 80] \times [0, 80]$.
- The distance between any two locations is the Euclidean distance.
- Vehicles can travel a maximum distance of 200, and can carry no more than 150 units.
- For each served customer, an additional drop time equivalent to 10 distance units is considered when calculating the fleet's overall travelling distance.

Various extensions of the basic VRP have been extensively studied, among others VRPs with time constraints (time windows) for customer servicing [Badeau et al., 1997, Kilby et al., 1999], VRPs which include pickup and delivery tasks [Gendreau et al., 1998, Li and Lim, 2001], multiple depot problems [Irnich, 2000], and dynamic VRP scenarios [Psaraftis, 1995, Kilby et al., 1998, Bianchi, 2000]. For further information refer to recent reviews such as [Toth and Vigo, 2001].

6.3.2 Application II: A Tabu Search for the VRP

General Concept

Tabu Search methods are a class of algorithms that were pioneered by [Glover, 1989]. This class is a further exponent of neighbourhood search techniques. The basic strategy, or heuristic, employed by all Tabu Search implementations while manoeuvring through the search space is to guide the search away from local optima once they have been visited. In order to do so, Tabu Search maintains a data structure called tabu list. All neighbourhood moves made by the search are recorded there for a certain time. In each search cycle, the oldest move from the list gets replaced by the most recently used. The length of the tabu list thus determines how many cycles a move is avoided. A move found on this list and its reverse move are said to be tabu. That means that this move and its reverse are temporarily not available, and the search cannot apply them as long as they are present in

the tabu list. Hence the search cannot reverse its most recent modification steps, and, consequently, is forced to explore other points of the search space.

During the search process, the algorithm stores the best solution encountered so far. If a move would improve this overall best solution but is considered tabu, then a mechanism known as aspiration overrides the move's tabu status so that the new best solution can be reached.

Tabu Search for the VRP

After the initial remarks about Tabu Search, we now introduce a specific implementation for VRPs as previously discussed. This algorithm is a modified and extended version of the Tabu Search outlined in [Duncan, 1995]. It is controlled by seven parameters which are listed and described below.

Parameter $\mathcal{P}_1$ - number of restarts: Each search instance can use a fixed number of cycles. These cycles can be assigned to a single Tabu Search run, or the search can be restarted several times from different initial solutions using less cycles. Therefore, the number of restarts is the first parameter. In our implementation, we limit the choices to $\mathbb{D}_1 = \{1, 2, 5\}$. Allowing search restarts is an extension of the original work by [Duncan, 1995].

Parameter $\mathcal{P}_2$ - neighbourhood operators: The search allows three different kind of neighbourhood moves: swap, re-insert and re-link moves.

- A swap move for two customers leads to the swapping of their positions.

This move can be applied to customers both on the same route of one vehicle, or to customers on routes of two different vehicles.

- A re-insert move results in the removal of a customer from its current position and the re-insertion at a different position. The re-insertion can take place in the route of any vehicle.

- A re-link move allows the exchange of sub-routes between two different vehicles. On each route, a customer is selected, and the partial routes consisting of all customers coming after these two selected ones are swapped.

Example 6.2. To illustrate these three neighbourhood move operators, we give an example for each when applied to the situation presented earlier in this chapter. In figure 6.2, the graph in the upper left-hand corner shows the initial state. On its top-right, the situation after swapping the customers $C1$ and $C7$ is given. In the lower left-hand graph, the routes after re-inserting customer $C1$ between $C7$ and $C6$ are presented, while in the lower right-hand figure the result of re-linking the routes after customers $C1$ and $C7$ is displayed.

Whether the search uses one, two or all three of these neighbourhood operators is controlled by the second parameter. The values 1 to 7 for these parameters describe the following choices: 1 = re-insert, 2 = re-link, 3 = re-link + re-insert, 4 = swap, 5 = swap + re-insert, 6 = swap + re-link and 7 = swap + re-link + re-insert. Hence the second parameter domain is

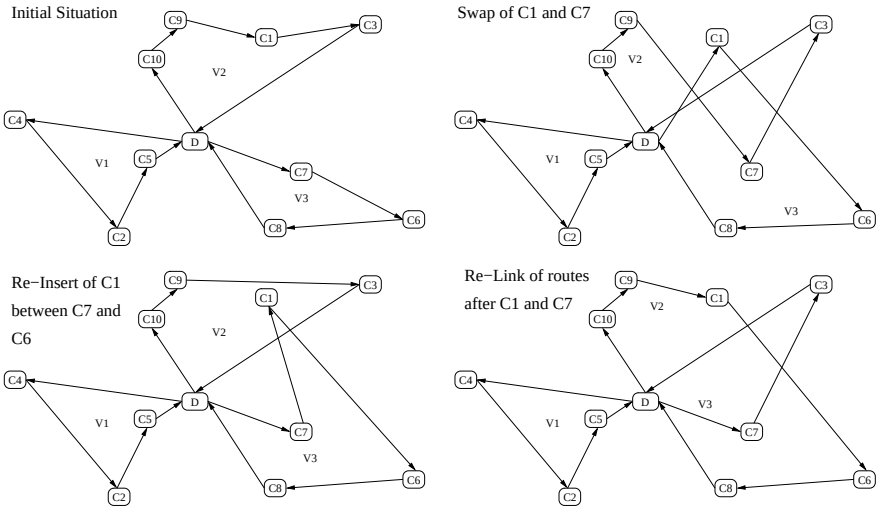

Figure 6.2: The VRP Neighbourhood Operators

$\mathbb{D}_2 = \{1, 2, 3, 4, 5, 6, 7\}$.

Parameter $\mathcal{P}_3$ - stepping factor: [Duncan, 1995] proposes a factor for stepping through the neighbourhood in order to speed the search up. A factor of 2 for instance would mean that just every second member of the neighbourhood set would be considered. This stepping factor is a further search parameter, and we consider values from $\mathbb{D}_3 = \{1, 2, 4, 8, 16\}$.

Parameter $\mathcal{P}_4$ - aspiration: A fourth parameter controls the use of aspiration. The value 1 means that the aspiration criterion is used, while 0 stands for no aspiration: $\mathbb{D}_4 = \{0, 1\}$.

Parameter $\mathcal{P}_5$ - random move probability: During the search process, the best non-tabu or aspired move is selected, unless a purely random move is chosen. The probability for such a random move is controlled by the fifth search parameter. We allow this likelihood to be between 0% and 50%:

$\mathbb{D}_5 = [0.0, 0.5]$. The possibility of random moves is an addition to the original method.

Parameters $\mathcal{P}_6$ and $\mathcal{P}_7$- tabu list length and its variation: The tabu list length $\mathcal{P}_6$ is the sixth search parameter. To avoid unwanted cycling effects, a random variation of the tabu list length $\mathcal{P}_7$ is introduced, which constitutes the final search parameter. More specifically, a move that gets added to the tabu list becomes tabu for $\mathcal{P}_6 + random(\mathcal{P}_7)$ cycles. We select these values from the domains $\mathbb{D}_6 = \{5, 10, 15, 25, 35, 50, 75\}$ and $\mathbb{D}_7 = \{0, 1, 2, 5, 10, 20, 50\}$.

Table 6.4 lists all seven search parameters along with their description and their domains as used in this chapter's experiments. The choice of parameter values is kept as broad as possible in order to show that the MAPAA can learn to distinguish between promising and less promising values. The initial choices as used by the adaptation schemes are highlighted.

Parameter	Description	Domain
$\mathcal{P}_1$	number of restarts	$\{\mathbf{1}, 2, 5\}$
$\mathcal{P}_2$	neighbourhood operators	$\{1, 2, 3, 4, 5, 6, \mathbf{7}\}$
$\mathcal{P}_3$	stepping factor	$\{\mathbf{1}, 2, 4, 8, 16\}$
$\mathcal{P}_4$	aspiration	$\{0, \mathbf{1}\}$
$\mathcal{P}_5$	random move probability	$[0.0, 0.5](\mathbf{0.0})$
$\mathcal{P}_6$	tabu list length	$\{5, \mathbf{10}, 15, 25, 35, 50, 75\}$
$\mathcal{P}_7$	tabu list length variation	$\{\mathbf{0}, 1, 2, 5, 10, 20, 50\}$

Table 6.4: Parameters of the Tabu Search for the VRP

The Tabu Search, used with all three neighbourhood operators and a stepping factor of 1, is run for $1,000$ cycles. With fewer operators or a higher stepping

factor, this cycle number is increased to guarantee comparable computation times of the search runs. Similarly, increasing the number of restarts results in less cycles per run.

Experimental MAPAA Results

The SPO for tuning the aforementioned Tabu Search method starts with the initial setting $(1, 7, 1, 1, 0.0, 10, 0)$. The order in which the parameters of this heuristic search are optimised is the same as in table 6.4. At each step of the optimisation process, the objective is averaged over the same set of 100 randomly created VRP instances. Appendix D.2 gives the results of the complete SPO run. It yields the parameter setting $(5, 7, 1, 0, 0.15, 35, 0)$. In other words, this approach favours a Tabu Search with 5 restarts, the richest neighbourhood of 7, a stepping factor of 1, no use of the aspiration criterion, random moves with a probability of 0.15, and a tabu list of a fixed length of 35 elements.

In 20 independent runs based on different sample VRP instances, the MA-PAA is applied to the Tabu Search method as well. Table 6.5 lists the returned findings together with the initial parameter setting and the result of SPO. For all settings, the objective result of the corresponding heuristic search, averaged over 200 out-of-sample VRP instances, is given.

In contrast to Application I, where SPO and the MAPAA produced very similar settings, table 6.5 highlights some significant differences in the results

Parameter Setting	$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_5$	$\mathcal{P}_6$	$\mathcal{P}_7$	f_{avg}
Initial	1	7	1	1	0.00	10	0	1402.6
SPO	5	7	1	0	0.15	35	0	1376.5
MAPAA #1	2	7	4	1	0.02	75	50	1370.6
MAPAA #2	2	7	2	1	0.05	50	20	1372.0
MAPAA #3	2	7	2	1	0.07	75	1	1370.7
MAPAA #4	1	6	2	1	0.08	75	20	1371.8
MAPAA #5	1	7	2	1	0.03	75	10	1373.4
MAPAA #6	5	7	2	1	0.09	75	2	1371.2
MAPAA #7	5	7	4	1	0.02	50	50	1371.5
MAPAA #8	5	7	2	1	0.05	50	20	1370.7
MAPAA #9	1	7	2	0	0.10	35	2	1372.1
MAPAA #10	2	7	2	1	0.20	10	2	1374.6
MAPAA #11	5	7	2	1	0.06	75	50	1372.1
MAPAA #12	1	7	2	0	0.13	15	10	1373.3
MAPAA #13	2	7	4	1	0.05	50	5	1371.6
MAPAA #14	5	7	2	1	0.12	25	0	1372.6
MAPAA #15	2	7	2	0	0.10	35	1	1372.6
MAPAA #16	1	7	2	1	0.16	35	2	1371.8
MAPAA #17	1	7	2	1	0.07	50	1	1372.2
MAPAA #18	5	7	4	1	0.03	75	0	1371.1
MAPAA #19	5	7	2	1	0.02	35	10	1371.7
MAPAA #20	2	7	2	1	0.11	35	50	1372.5
f_{avg}: $mean = 1372.0$, $\sigma = 0.97$								

Table 6.5: Experimental results for a Tabu Search for the VRP

between these two approaches to the current scenario.

- While SPO strongly favours 5 restarts, our adaptation approach produces solutions based on 1, 2 or 5 restarts with comparable frequency.
- Both methods agree on neighbourhood 7, the neighbourhood that includes all three available operators, as being the best option. Only the fourth MAPAA run differs slightly by proposing neighbourhood 6.

- The first real significant difference in the results is the choice of the stepping factor. In 16 out of the 20 MAPAA runs, a factor of 2 is found to be the most effective one, the remaining 4 runs suggest a factor of 4. This is in strong contrast to the SPO result which sees 1 as superior.

- 85% of the MAPAA settings include aspiration and thus disagree with the SPO result. The surprising absence of the usage of this criterion by SPO can be explained as follows. During the optimisation process, the decision about its application is made using a very short tabu list of length 10. With only few moves being tabu, aspiration is much less beneficial and, by chance, is rejected. These findings are supported by MAPAA results. Only in runs 9, 12 and 15 is no aspiration used, and these cases also feature tabu lists of moderate length.

- The majority of settings produced by our adaptation approach recommend small random move probabilities between 0.0 and 0.1. Only in five cases, namely runs 10, 12, 14, 16 and 20, is a higher value from $[0.1, 0.2]$ proposed. Very interesting to note is that these higher probabilities coincide with short or moderate tabu list lengths. Further experiments support this observation: Tabu Search is sensitive to the chosen random move probability. It works very well with either a large tabu list combined with few random moves, or a shorter tabu list with a higher chance for random moves.

- In general, the MAPAA results recommend a tabu list of at least length 35. Only in 15% of the runs was a shorter length favoured. In these

cases, higher random move probabilities are applied as described above. The tabu list length variation does not seem to have a decisive influence.

Both SPO and the MAPAA clearly improve on the initial setting, which mainly constitutes the parameter choices tested in [Duncan, 1995]. However, the considerably different parameter settings produced by the two approaches lead to performance differences. Specifically, all parameter settings yielded by our novel adaptation method are significantly better than the SPO solution with a significance level of 1%.

Conclusion

The experiments provide strong evidence for the superiority of the MAPAA over SPO when it comes to finding good parameter settings for the discussed Tabu Search when solving VRPs. The collected data demonstrate that the MAPAA is better able to detect and consider interdependencies of parameters, e.g. between aspiration, random move probability and tabu list length in this scenario. Consequently, the MAPAA consistently and significantly outperforms the more basic SPO approach.

6.3.3 Application III: A Genetic Algorithm for the VRP

Genetic Algorithm for the VRP

In addition to the Tabu Search method discussed in section 6.3.2, [Duncan, 1995] proposed a Genetic Algorithm (GA) for the Vehicle Routing Problem . His method differs from the standard GA approach that was discussed in section 3.1.2. Firstly, the solution representation is not based on a bit string but rather uses a model for the vehicle tours. Because of this difference, the classical crossover and mutation operators are not applicable. Therefore, he secondly introduces a new set of operators to produce offspring. These mechanisms are the swap, re-insert and re-link moves that we outlined in section 6.3.2 and illustrated in example 6.2. These operators modify a single parent and produce a single offspring. Consequently, the discussed GA does not include a crossover scheme but rather applies three specialised mutation techniques. Our implementation of this heuristic search algorithm has seven parameters, these are specified below.

Parameter $\mathcal{P}_1$ - number of restarts: The complete search process can make use of a specified number of generational cycles. These generations can be evenly divided in order to facilitate a restart mechanism. In our experiments, we allow 1, 2 or 5 restarts of the GA. This parameter was not considered in the original work by [Duncan, 1995].

Parameter $\mathcal{P}_2$ - neighbourhood operators: The GA can apply one, two or all three of the explained mutation operators. The possible values 1 to 7 correspond to 1 = re-insert, 2 = re-link, 3 = re-link + re-insert, 4 = swap, 5 = swap + re-insert, 6 = swap + re-link and 7 = swap + re-link + re-insert.

Parameter $\mathcal{P}_3$ - population size: The third parameter controls the size of the population. We consider here the choices 50, 100, 150, 200 and 250.

Parameter $\mathcal{P}_4$ - elitism: The boolean elitism flag establishes whether the best found candidate solution in the current population is passed on to the next generation, $\mathcal{P}_4 = 1$, or whether it is replaced, $\mathcal{P}_4 = 0$.

Parameter $\mathcal{P}_5$ - better initial tours: [Duncan, 1995] recommends to seed the GA with good initial solutions. The fifth parameter defines whether a simple savings algorithm is used for the determination of the initial population, $\mathcal{P}_5 = 1$, or whether this population is made up of purely random tours, $\mathcal{P}_5 = 0$.

Parameter $\mathcal{P}_6$ - parenthood proportion: To model the principle of natural selection, only candidate solutions with good evaluations are allowed to participate in the reproduction process. In this GA, only the best performing population members get this chance. The sixth parameter determines which proportion of the population will be marked for parenthood. Possible choices are 5, 10, 15, 20, 30, 40 and 50 percent.

Parameter $\mathcal{P}_7$ - replacement scheme: Two different replacement schemes are possible. In the generational scheme, $\mathcal{P}_7 = 1$, the population is replaced

entirely by the offspring population. In the second option $\mathcal{P}_7 = 0$, the incremental scheme, offspring are produced one at a time and replace the worst member of the population.

Table 6.6 provides an overview of the seven aforementioned GA parameters. The initial parameter choices, that resemble the values used by [Duncan, 1995], are highlighted in bold.

Parameter	Description	Domain
$\mathcal{P}_1$	number of restarts	$\{\mathbf{1}, 2, 5\}$
$\mathcal{P}_2$	neighbourhood operators	$\{1, 2, 3, 4, 5, 6, \mathbf{7}\}$
$\mathcal{P}_3$	population size	$\{50, \mathbf{100}, 150, 200, 250\}$
$\mathcal{P}_4$	elitism	$\{0, \mathbf{1}\}$
$\mathcal{P}_5$	better initial tours	$\{0, \mathbf{1}\}$
$\mathcal{P}_6$	parenthood proportion	$\{5, \mathbf{10}, 15, 20, 30, 40, 50\}$
$\mathcal{P}_7$	replacement scheme	$\{0, \mathbf{1}\}$

Table 6.6: Parameters of a Genetic Algorithm for the VRP

To guarantee a fair comparison of all possible GA configurations, the number of generational cycles is chosen depending on the number of restarts and the used population size. More specifically, the product of the number of generations, restarts and population size is always kept constant. A GA with one restart and a population of 100 candidate solutions runs for 1000 generations.

Experimental MAPAA Results

Starting with the parameter choices $(1, 7, 100, 1, 1, 10, 1)$, the SPO run, a detailed account of which can be found in appendix D.3, returns the setting $(1, 7, 50, 1, 1, 5, 1)$ as its result. All configurations are tested for 100 VRP instances. The final parameter setting encodes a GA that is started once, applies all three possible mutation operators, uses a population of size 50, uses elitism and a savings algorithm for the initial solutions, selects the top 10% of a population as potential parents and uses a generational replacement scheme. Only two improvements are found over the initial choices which are the values analysed by [Duncan, 1995].

As in the previous two applications, the MAPAA is applied in 20 independent runs to optimise the parameters of the discussed GA, starting with the same initial parameter setting as SPO. The results of these experiments are presented in table 6.7. This table also shows the initial parameter setting and the solution produced by SPO. To compare the quality of the configurations found, the corresponding GA instances are applied to 200 VRPs. The last column of the table lists the average objective function values achieved for these out-of-sample problems.

- In 13 out of 20 runs, 2 restarts are favoured. In the remaining 7 cases, exactly one run of the GA is recommended.
- In 90% of the settings produced by the MAPAA, the usage of all three available mutation operators is suggested. Only in runs #4 and #16 is

Parameter Setting	$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_5$	$\mathcal{P}_6$	$\mathcal{P}_7$	f_{avg}
Initial	1	7	100	1	1	10	1	1429.3
SPO	1	7	50	1	1	5	1	1404.3
MAPAA #1	1	7	150	1	1	5	0	1400.7
MAPAA #2	2	7	50	1	1	5	0	1398.4
MAPAA #3	2	7	150	1	1	5	0	1399.0
MAPAA #4	1	6	200	1	1	5	0	1401.0
MAPAA #5	2	7	100	1	1	5	0	1397.5
MAPAA #6	1	7	150	0	1	5	0	1400.7
MAPAA #7	2	7	50	0	1	5	0	1398.4
MAPAA #8	1	7	50	1	1	20	0	1399.5
MAPAA #9	1	7	150	0	1	10	0	1399.4
MAPAA #10	2	7	100	0	1	5	0	1397.5
MAPAA #11	2	7	50	0	1	15	0	1397.6
MAPAA #12	2	7	100	0	1	5	0	1397.5
MAPAA #13	2	7	50	0	1	15	0	1397.6
MAPAA #14	2	7	50	1	1	10	0	1396.9
MAPAA #15	2	7	200	1	1	5	0	1400.7
MAPAA #16	1	6	150	1	1	10	0	1401.5
MAPAA #17	2	7	50	0	1	10	0	1396.9
MAPAA #18	2	7	50	1	1	10	0	1396.9
MAPAA #19	2	7	100	1	1	5	0	1397.5
MAPAA #20	1	7	50	1	1	15	0	1397.7
f_{avg}: $mean = 1398.6$, $\sigma = 1.51$								

Table 6.7: Experimental results for a Genetic Algorithm for the VRP

this choice limited to two operators. However, these two settings show the worst performance amongst all MAPAA results.

- In contrast to SPO, our adaptation method strongly recommends the usage of an incremental replacement scheme instead of a generational technique. As the incremental replacement scheme automatically ensures the survival of the fittest parent, the results about the elitism parameter are not significant here.

- All 20 runs yield GA configurations that include the usage of a savings approach to create better initial tours.
- Although the vast majority of MAPAA results recommend small population sizes of 50 or 100, some solutions perform well with populations of 150 or 200 individuals.
- Regarding the proportion of the population that should be used for reproduction, smaller values of 5% and 10% dominate. Percentages larger than 20% are dismissed.

The decisive difference between the parameter settings yielded by SPO and the MAPAA is that the former suggests the usage of a generational replacement scheme while the latter strongly hints that replacing population members incrementally is superior. It seems that the specific combination of one start, a population of just 50 individuals and the usage of the top 5% for reproduction is unfavourable for the incremental method. Therefore, SPO concludes that the generational approach is superior. The MAPAA, on the other hand, is capable of detecting more favourable settings for the incremental scheme, and produces solutions that do not contain the specific aforementioned parameter combination.

Both the SPO and the MAPAA significantly improve the parameter choices studied and suggested by [Duncan, 1995]. Furthermore, the superiority of the results achieved by our adaptation approach over the SPO results can be statistically substantiated. Under a strict significance level of 1%, a significantly better performance is achieved in 15 out of 20 cases. Relaxing

the confidence level to 10% shows that all 20 runs yield superior parameter settings.

Conclusion

The experiments with the MAPAA demonstrate that the performance of the GA proposed by [Duncan, 1995] can be significantly improved by using better parameter choices. In particular, the statement that "results are inconclusive" regarding the usage of an incremental replacement scheme could not be confirmed. On the contrary, our study suggests that such an incremental approach can be very beneficial. Our adaptation method is very robust in producing configurations that outperform both the original settings as well as the parameter values yielded by SPO.

6.3.4 Application IV: A Tabu Search and a Genetic Algorithm for the VRP

In the fourth application presented in this chapter, the MAPAA is applied simultaneously to the two heuristic search methods introduced in the previous sections 6.3.2 and 6.3.3. We show that the adaptation method is capable of detecting the algorithm with superior performance and can produce good parameter settings accordingly.

Experimental MAPAA Results

For both the Tabu Search and the Genetic Algorithm for the VRP, the SPO is exactly the same as in the scenarios where these methods are studied alone. As we have to conduct the SPO process for two algorithms, the required computation effort is doubled.

The MAPAA run, on the other hand, does not require additional resources in case of multiple heuristic search methods. The outcome of 20 experimental runs is summarised in table 6.8. In addition, the table provides average objective results achieved for a set of 200 VRP instances.

The results are clear and without ambiguity: the Tabu Search is the superior method in our tests. The MAPAA finds it to be the best performing algorithm in every single run.

Statements about the parameter settings produced by the adaptation scheme for the Tabu Search are comparable to those made in section 6.3.2. We abstain from repeating these observations and rather concentrate on two differences. Firstly, the MAPAA finds against using the richest neighbourhhod as its choice in 3 out of 20 cases, compared to just one miss in the single algorithm scenario. Secondly, higher mutation rates, above 0.10, are more frequently proposed in the multiple algorithm case. These differences indicate that the MAPAA comes, on average, to slightly different results if it has to distribute its computation resources between two heuristic search methods.

Parameter Setting	Alg.	$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_5$	$\mathcal{P}_6$	$\mathcal{P}_7$	f_{avg}
SPO	GA	1	7	50	1	1	5	1	1404.3
SPO	TS	5	7	1	0	0.15	35	0	1376.5
MAPAA #1	TS	5	7	2	1	0.07	25	1	1372.7
MAPAA #2	TS	2	6	2	1	0.13	50	5	1372.7
MAPAA #3	TS	1	7	2	1	0.15	50	5	1372.2
MAPAA #4	TS	2	7	2	1	0.09	50	20	1371.3
MAPAA #5	TS	2	7	2	1	0.11	75	20	1371.2
MAPAA #6	TS	1	7	2	1	0.14	75	50	1373.0
MAPAA #7	TS	5	7	2	1	0.07	75	50	1371.3
MAPAA #8	TS	1	7	2	0	0.15	35	50	1373.7
MAPAA #9	TS	1	7	2	1	0.11	75	10	1371.6
MAPAA #10	TS	5	7	4	1	0.05	75	5	1371.4
MAPAA #11	TS	2	6	2	1	0.08	50	20	1372.8
MAPAA #12	TS	1	7	2	1	0.10	25	0	1372.5
MAPAA #13	TS	2	7	2	1	0.10	50	10	1371.8
MAPAA #14	TS	2	7	2	0	0.10	75	10	1372.5
MAPAA #15	TS	2	7	2	1	0.07	35	0	1372.1
MAPAA #16	TS	2	7	4	1	0.03	35	1	1371.2
MAPAA #17	TS	1	7	2	1	0.17	75	2	1373.0
MAPAA #18	TS	1	6	2	1	0.13	35	20	1373.0
MAPAA #19	TS	2	7	4	1	0.04	50	10	1371.9
MAPAA #20	TS	2	7	2	1	0.12	50	50	1371.8
									f_{avg}: $mean = 1372.2$, $\sigma = 0.72$

Table 6.8: Experimental results for a Tabu Search/Genetic Algorithm for the VRP

The parameter settings recommended by MAPAA for the Tabu Search again clearly outperform the configurations yielded by SPO for the Tabu Search and the GA. Their performance is on average worse by 0.2 than the performance of the settings produced in the single algorithm scenario in section 6.3.2. However, a sample size of only 20 results is not sufficient to prove a significance of this difference.

Conclusion

The MAPAA demonstrates in this experiment that it can, when it is applied to two heuristic search algorithms at once, detect the better performing search and can consistently produce high performance parameter settings for it. Although the adaptation process has to divide its resources between the Tabu Search and the GA, it still clearly outperforms the SPO approach. The results indicate that such a sharing of resources, i.e. a sharing of available computation time, leads to a slight degradation of the solution quality in comparison to the scenario where the focus is only on one heuristic search.

6.4 Applications for Job Shop Scheduling

The final three applications of the MAPAA presented in this chapter look at heuristic search methods for the Job Shop Scheduling Problem (JSSP). The parameter adaptation processes are studied individually for a GA and a Tabu Search as well as for both methods together.

6.4.1 The Job Shop Scheduling Problem

The Job Shop Scheduling Problem is a further example of the class of NP-hard optimisation problems. It contains a set of concurrent jobs. Each job consists of a set of operations that must be scheduled according to a given

process plan. This order differs from job to job. As each operation requires a resource, called a machine, for a certain time span to be completed, operations are in competition for available resources. The task is to schedule all operations while taking account of the process orders and the constraint that a machine can process only one operation at any given time. A widely used criterion to assess the performance of a schedule is the makespan which is the duration between the starting time of the first operation and the completion time of the last operation. In the experiments reported in section 6.4.2, the objective is to minimise the makespan. The following example discusses a small scheduling task.

Example 6.3. We consider a 3×2 Job Shop Scheduling Problem, i.e. a scenario with 3 jobs and 2 machines. The problem data

```
job 1:  1 (2)  2 (5)
job 2:  2 (3)  1 (7)
job 3:  1 (4)  2 (4)
```

describe 3 jobs, each consisting of 2 operations. The first operation of job 1 requires machine 1 for 2 time units. After its completion, the second operation of job 1, which requires machine 2 for 5 time units, can be processed. Analogically, job 2 needs first the second resource for 3 time units, and thereafter machine 1 for a time span of 7. The third job has to be processed first on machine 1 and then on machine 2, each time for 4 units of time.

The first operations of both job 1 and 3 must be processed on machine 1. As a machine can handle only one operation at a time, one of these jobs must delay its start. As a consequence of such order decisions, different schedules can be generated. One possible plan for the completion of the example jobs is:

```
machine 1: job 1 (time 0-2), job 3 (time 2-6), job 2 (time 6-13)
machine 2: job 2 (time 0-3), job 1 (time 3-8), job 3 (time 8-12)
```

As the last operation is finished at time point 13 under this schedule, the makespan is 13. □

In order to have sufficient problem instances for the MAPAA process, we generate random JSSPs that closely resemble the famous $FT10 \times 10$ problem [Fisher and Thompson, 1963]:

- Each problem consists of 10 jobs and 10 machines.
- The order of machines a job has to be processed on is a random permutation of $(1, \ldots, 10)$.
- The processing time of each operation is randomly chosen from $[2, 99]$.

Among the heuristic search methods that have been successfully applied to the JSSP are Genetic Algorithms [Lin et al., 1997], Simulated Annealing [van Laarhoven et al., 1992] and Tabu Search [Taillard, 1994].

6.4.2 Application V: A Genetic Algorithm for the JSSP

A Genetic Algorithm for the JSSP

[Lin et al., 1997] describe different GA approaches for Job Shop Scheduling. To counter the problem of premature convergence in the classical GA, they propose and analyse fine-grained, coarse-grained and hybrid models.

We study here the coarse-grained algorithm, often referred to as an island-parallel GA. Instead of a single population, a number of subpopulations is maintained. These subpopulations evolve independently. The fundamental idea is that the separation leads to diverse island populations and thus to a higher diversity overall. The islands are arranged in a ring topology. At certain times individuals are allowed to migrate to other, neighbouring subpopulations to facilitate the exchange of information.

A direct representation approach for modelling candidate solutions, i.e. schedules for a JSSP, is used: a solution encodes the operation starting times. The genetic operators applied by [Lin et al., 1997] to these schedules are time horizon exchange crossover and mutation. These operators, which are based on the G&T algorithm [Giffler and Thompson, 1960], are highly specialised and incorporate very problem specific knowledge. For more details, the interested reader is referred to the original work.

Our GA implementation has the following eight parameters:

Parameter $\mathcal{P}_1$ - number of restarts: The available computation resources for one search instance can be divided between 1, 2 or 5 restarts of the GA. This constitutes an extension of the original method by [Lin et al., 1997].

Parameter $\mathcal{P}_2$ - number of subpopulations: As described above, the GA maintains a number of subpopulations. We consider the values 1, 2, 5 or 10 in our experiments.

Parameter $\mathcal{P}_3$ - size of subpopulations: The third parameter defines the number of individuals in each of the subpopulations. Although [Lin et al., 1997] allow for very large populations with up to 2000 individuals, we limit the choices to $\mathbb{D}_3 = \{10, 20, 50, 100, 200\}$.

Parameter $\mathcal{P}_4$ - migration: This control parameter describes the time interval in which a subpopulation's best individual is allowed to migrate to another island.

Parameter $\mathcal{P}_5$ - crossover rate: After the selection of two parents from the current population, two offspring are produced through time horizon exchange crossover with probability $\mathcal{P}_5$. With probability $1 - \mathcal{P}_5$, these parents are passed on directly to the next generation.

Parameter $\mathcal{P}_6$ - mutation rate: With a certain percentage $\mathcal{P}_6$, individuals are mutated when being passed on to the new population.

Parameter $\mathcal{P}_7$ - elitism: This flag controls the application of the elitism mechanism. If $\mathcal{P}_7 = 1$, the best individual is guaranteed survival.

Parameter $\mathcal{P}_8$ - selection mechanism: The reproduction process of the GA involves the selection of parents from the current population. We test two possible schemes. Fitness selection ($\mathcal{P}_8 = 0$) chooses parents from the subpopulations with probabilities proportional to their fitness[8]. Tournament selection ($\mathcal{P}_8 = 1$), on the other hand, selects a parent by picking the fittest of three randomly selected individuals. This choice between different selection mechanisms is an addition to the original method.

Table 6.9 provides a summary of all eight parameters and their respective domains. The values of the initial parameter setting, that describe a classical GA with just one population of 100 individuals, are highlighted.

Parameter	Description	Domain
$\mathcal{P}_1$	number of restarts	$\{\mathbf{1}, 2, 5\}$
$\mathcal{P}_2$	number of subpopulations	$\{\mathbf{1}, 2, 5, 10\}$
$\mathcal{P}_3$	size of subpopulations	$\{10, 20, 50, \mathbf{100}, 200\}$
$\mathcal{P}_4$	migration	$\{0, 10, 20, \mathbf{50}, 100\}$
$\mathcal{P}_5$	crossover rate	$[0, 1]$ $(\mathbf{0.6})$
$\mathcal{P}_6$	mutation rate	$[0, 1]$ $(\mathbf{0.1})$
$\mathcal{P}_7$	elitism	$\{0, \mathbf{1}\}$
$\mathcal{P}_8$	selection mechanism	$\{\mathbf{0}, 1\}$

Table 6.9: Parameters of a Genetic Algorithm for the JSSP

Experimental MAPAA Results

SPO modifies the initial parameter setting $(1, 1, 100, 50, 0.6, 0.1, 1, 0)$ step by step and returns $(2, 2, 50, 10, 0.6, 0.7, 1, 1)$ as its result. In other words, this

[8] As the objective is to minimise the makespan of schedules, a lower objective leads to a higher probability.

basic parameter optimisation approach recommends a GA configuration with 2 restarts, 2 subpopulations of 50 individuals, migration every 10 generations, crossover and mutation rates of 0.6 and 0.7, respectively, the use of elitism and a tournament selection mechanism. Appendix D.4 contains the full numerical results.

Table 6.10 confronts the parameter settings found by the MAPAA in 16 independent runs with the initial parameter values and the SPO choices. To allow for a performance comparison, all GA configurations are tested for 100 JSSP instances, the average results of which are listed in the last column.

Parameter Setting	$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_5$	$\mathcal{P}_6$	$\mathcal{P}_7$	$\mathcal{P}_8$	f_{avg}
Initial	1	1	100	50	0.6	0.1	1	0	882.6
SPO	2	2	50	10	0.6	0.7	1	1	858.0
MAPAA #1	2	2	200	50	0.97	0.81	1	1	854.6
MAPAA #2	2	5	10	50	0.70	0.94	0	1	852.3
MAPAA #3	5	1	200	100	0.74	0.85	1	1	854.0
MAPAA #4	2	5	100	10	0.56	0.94	1	1	850.9
MAPAA #5	5	2	50	50	0.20	0.96	1	1	851.5
MAPAA #6	2	5	100	50	0.76	0.88	1	1	851.5
MAPAA #7	5	5	10	20	0.48	0.88	0	1	853.4
MAPAA #8	5	5	20	10	0.21	0.99	0	1	853.4
MAPAA #9	2	5	50	10	0.75	0.95	1	1	852.9
MAPAA #10	5	5	10	20	0.19	0.86	0	1	852.6
MAPAA #11	5	2	50	50	0.83	0.80	0	1	856.5
MAPAA #12	5	2	200	100	0.53	0.96	1	1	853.3
MAPAA #13	2	5	100	10	0.81	0.97	1	1	851.6
MAPAA #14	5	1	200	50	0.35	0.96	1	1	851.6
MAPAA #15	2	5	10	100	0.60	0.97	0	1	852.4
MAPAA #16	2	5	100	10	0.56	0.66	1	1	854.0
									f_{avg}: $mean = 852.9$, $\sigma = 1.39$

Table 6.10: Experimental results for a Genetic Algorithm for the JSSP

The following observations can be made for the MAPAA results:

- A search process with 2 or 5 restarts of the GA is clearly favoured as these two values appear in all runs.
- In 14 out of 16 runs, the MAPAA suggests to use 2 or 5 subpopulations. In the 2 remaining cases, a single population of maximal size is found to be effective. Configurations with 10 subpopulations never appear approved.
- All possible subpopulation sizes feature in parameter settings produced by the MAPAA. As a general trend, smaller population sizes occur in combination with a higher number of subpopulations while larger population sizes are mainly paired with fewer subpopulations.
- Migration is found to be beneficial as its absence, $\mathcal{P}_4 = 0$, is clearly rejected. In the vast majority of cases, migration cycles between 10 and 50 are chosen.
- The rate of crossover does not seem to influence the GA performance very much as a wide spectrum of values is found to work well. Only small values lead to a significantly reduced performance. Therefore, the MAPAA never recommends crossover rates below 0.19 in the experiments.
- The analysis of the established mutation rates is very surprising. 15 out of 16 runs yield a mutation rate above 0.8. Even in the one remaining case, a high value of 0.66 is returned. Further experiments, among

others also in the SPO run given in appendix D.4, show that a high time horizon exchange mutation rate between 0.7 and 1.0 is essential for a good GA performance. Smaller values lead to a clear performance loss.

- The MAPAA usually produces solutions that include the elitism criterion. Only in cases where small population sizes are used is this mechanism not supported.
- In all MAPAA runs, the tournament selection mechanism is found to be the superior scheme.

Instead of finding a specific area of the parameter space to be superior, our novel adaptation approach proves that a wide variety of different GA configurations do perform very well. All 16 runs significantly improve the initially applied parameter setting, and also have a better performance than the SPO result. For all runs but MAPAA #11, the t-test shows this better performance to be significant at a significance level of 1%.

This experiment illustrates a further facet of the MAPAA. As the user can define very broad and general parameter domains, the danger of missing good parameter values due to inappropriate assumptions is reduced. As an example, GAs are usually recommended to be run with small to moderate mutation rates. Here, however, this recommendation is wrong. The specific nature of the time horizon exchange mutation operator makes it a central element in the process of improving candidate solutions, and a high frequency for its application is suggested.

Conclusion

In the experiments conducted, the MAPAA is very robust in improving the initial parameter setting and outperforms the SPO approach. It consistently finds dramatic performance enhancing upgrades for a single population GA with rates of 0.6 and 0.1 for crossover and mutation, respectively. Moreover, the tests with JSSP instances that resemble the FT10 $\times$ 10 problem suggest that a mutation rate of 0.1, as recommended in [Lin et al., 1997], is too small for the specific time horizon exchange mutation operator. Higher mutation rates yield clearly better results.

6.4.3 Application VI: A Tabu Search for the JSSP

A Tabu Search for the JSSP

[Taillard, 1994] proposes a Tabu Search, the general concept of which is introduced in section 6.3.2, for the JSSP. His parallel technique uses a graph representation for the schedules, and it employs a neighbourhood based on the permutation of specific, so called "critical" operations within such graphs. A detailed discussion of these two concepts is beyond the scope of this thesis, and the interested reader is therefore referred to the original work.

Our Tabu Search implementation for the JSSP, which is based on the aforementioned method, can be controlled by seven parameters:

Parameter $\mathcal{P}_1$ - number of restarts: As an extension to the original algorithm, we allow the restarting of the search from a different, randomly created initial solution. Possible values for $\mathcal{P}_1$ are 1, 2 and 5.

Parameter $\mathcal{P}_2$ - aspiration: The binary second parameter controls the usage of the aspiration criterion, with $\mathcal{P}_2 = 1$ representing its application.

Parameter $\mathcal{P}_3$ - random move probability: In contrast to the original Tabu Search by [Taillard, 1994], we consider the possibility of random moves during the search process. Such moves are performed with probability $\mathcal{P}_3 \in [0.0, 0.5]$.

Parameters $\mathcal{P}_4$, $\mathcal{P}_5$ and $\mathcal{P}_6$ - tabu list parameters: The fourth parameter $\mathcal{P}_4$ determines the basic time span a move, once chosen, becomes tabu. To counter unwanted cycling effects, the actual tabu period is randomly chosen from the interval $[\mathcal{P}_4 \times \mathcal{P}_5, \mathcal{P}_4 \times \mathcal{P}_6]$.

Parameter $\mathcal{P}_7$ - penalty factor: In order to discourage repetition, the Tabu Search records its moves and considers additional costs for moves found in its memory. The amount of these penalties is controlled by the seventh parameter.

Table 6.11 lists all parameters with their domains and default values.

Parameter	Description	Domain
$\mathcal{P}_1$	number of restarts	$\{\mathbf{1}, 2, 5\}$
$\mathcal{P}_2$	aspiration	$\{0, \mathbf{1}\}$
$\mathcal{P}_3$	random move probability	$[0.0, 0.5]$ $(\mathbf{0.0})$
$\mathcal{P}_4$	tabu list length	$\{4, 6, 8, 10, \mathbf{13}, 15, 20, 25, 35, 50\}$
$\mathcal{P}_5$	min. tabu list length factor	$[0.0, 1.0]$ $(\mathbf{0.8})$
$\mathcal{P}_6$	max. tabu list length factor	$[1.0, 5.0]$ $(\mathbf{1.2})$
$\mathcal{P}_7$	penalty factor	$[0.0, 2.0]$ $(\mathbf{0.5})$

Table 6.11: Parameters of a Tabu Search for the JSSP

Experimental MAPAA Results

If the Tabu Search method under consideration is subjected to the SPO process, then the initial values are altered only for two of the seven parameters. Instead of the default tabu list length of 13, 8 is suggested as a good choice, and 0.0 is favoured over the initial 0.8 for parameter $\mathcal{P}_5$. The full account of the SPO results can be found in appendix D.5.

If, on the other hand, the MAPAA is applied to improve the Tabu Search, then the recommended adaptations are more versatile, as table 6.12 confirms:

- The configuration with exactly 1 restart is found to be best in the majority of cases. However, 6 out of 16 MAPAA runs yield settings with 2 restarts.
- The application of the aspiration mechanism is clearly beneficial. In 13 out of the 16 experiments, this criterion is approved.
- Although random move probabilities of up to 0.5 are considered, the pa-

rameter adaptation process finds only small values to be useful. Probabilities between 0.0 and 0.1 are advised in 75% of the cases, while values between 0.11 and 0.17 are obtained in the remaining 25%.

- The analysis of the optimal tabu list length reveals different recommendations between the initial parameter setting, the result of SPO and the findings of the MAPAA experiments. While [Taillard, 1994] suggests a base length of 13 and varies it between $0.8 \cdot 13 \approx 10$ and $1.2 \cdot 13 \approx 16$, the process of optimising each parameter independently favours a length of 8 and its variation between $0.0 \cdot 8 = 0$ and $1.2 \cdot 8 \approx 10$. The MAPAA runs, however, yield a tabu list length of 4 as the best choice in the overwhelming majority of cases. Only in scenario #12, the final setting features the parameter value 6. With regard to the tabu list length variation parameters, the results are manifold, but values smaller than 0.2 for $\mathcal{P}_5$ and values greater than 2.83 for $\mathcal{P}_6$ do not occur.

- Small penalty factors between 0.04 and 0.38 occur in 15 out of 16 experimental runs, though values from the domain $[0.0, 2.0]$ are considered by the MAPAA. The one remaining run #12 yields 0.57.

The initial parameter setting achieves an average objective value of 857.8 for 100 test JSSP instances. SPO improves significantly on this by reducing the measure to 853.6. The MAPAA produces even better findings by averaging 853.1 in the 16 presented runs. Although these results hint at a superiority of the MAPAA, the available data are not sufficient to prove a statistical sig-

Parameter Setting	$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_5$	$\mathcal{P}_6$	$\mathcal{P}_7$	f_{avg}
Initial	1	1	0.0	13	0.8	1.2	0.5	857.8
SPO	1	1	0.0	8	0.0	1.2	0.5	853.6
MAPAA #1	2	1	0.06	4	0.55	1.77	0.13	853.9
MAPAA #2	1	1	0.07	4	0.87	2.48	0.12	853.1
MAPAA #3	1	1	0.10	4	0.25	2.34	0.27	852.5
MAPAA #4	1	1	0.09	4	0.20	2.54	0.06	852.7
MAPAA #5	2	1	0.17	4	0.20	1.64	0.38	853.8
MAPAA #6	1	0	0.10	4	0.73	1.23	0.26	854.0
MAPAA #7	2	1	0.11	4	0.60	2.09	0.19	853.3
MAPAA #8	1	1	0.10	4	0.78	2.44	0.29	853.6
MAPAA #9	1	1	0.06	4	0.93	1.54	0.20	853.3
MAPAA #10	2	0	0.02	4	0.92	2.04	0.29	852.6
MAPAA #11	2	1	0.13	4	0.81	1.21	0.36	852.2
MAPAA #12	1	1	0.03	6	0.58	1.52	0.57	852.7
MAPAA #13	2	1	0.05	4	0.42	2.83	0.29	854.1
MAPAA #14	1	0	0.15	4	0.24	1.75	0.16	852.5
MAPAA #15	1	1	0.05	4	0.43	2.60	0.06	853.2
MAPAA #16	1	0	0.01	4	0.70	2.23	0.04	851.9
								f_{avg}: $mean = 853.1$, $\sigma = 0.65$

Table 6.12: Experimental results for a Tabu Search for the JSSP

nificance. It has to be acknowledged furthermore that 25% of the adaptation experiments yield results slightly inferior to the SPO output.

The experimental data do however provide evidence for the robustness of our novel adaptation process. The achieved f_{avg} test measures all lie between 851.9 and 854.1. The standard deviation is just 0.65 around the mean 853.1.

Conclusion

A Tabu Search method for the JSSP was subjected to our parameter adaptation process. The MAPAA consistently proposes settings which strongly improved on the original configuration recommended by [Taillard, 1994]. It also outperforms, on average, the SPO method, but the difference can not be proven to be of statistical significance with the limited available data. Overall, the MAPAA once again proves its strength in terms of solution quality and robustness.

6.4.4 Application VII: A Genetic Algorithm and a Tabu Search for the JSSP

In the seventh and final application, the MAPAA is applied simultaneously to the two heuristic search methods previously introduced for the JSSP. Full details of these GA and Tabu Search techniques can be found in sections 6.4.2 and 6.4.3.

Experimental MAPAA Results

This experiment is similar to application IV in that the parameters of two search algorithms are optimised together. The main difference, however, is that here the two techniques are very evenly matched.

In the single algorithm scenario, the GA achieves an average test objective of 852.9 while the Tabu Search is just marginally worse with 853.1. The results of application VII, summarised in table 6.13, reflect this closeness. A GA configuration is proposed at the end of 6 of 16 MAPAA runs, and a Tabu Search parameter setting is the best found solution in the remaining 10 cases. As both methods are comparable in their performance, both are almost equally likely to feature as the MAPAA output.

Param. Sett.	Alg.	$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_5$	$\mathcal{P}_6$	$\mathcal{P}_7$	$\mathcal{P}_8$	f_{avg}
SPO	GA	2	2	50	10	0.6	0.7	1	1	858.0
SPO	TS	1	1	0.0	8	0.0	1.2	0.5		853.6
MAPAA #1	GA	1	5	200	20	0.48	0.98	0	1	852.7
MAPAA #2	GA	2	10	50	100	0.72	0.94	1	1	851.8
MAPAA #3	GA	2	5	200	100	0.49	0.84	1	1	854.5
MAPAA #4	TS	1	1	0.12	4	0.27	2.29	0.02		852.6
MAPAA #5	TS	2	1	0.01	4	0.49	2.82	0.49		854.2
MAPAA #6	GA	5	2	200	10	0.60	0.85	1	1	852.9
MAPAA #7	TS	1	0	0.09	4	0.47	1.82	0.14		853.1
MAPAA #8	TS	1	0	0.07	4	0.63	1.73	0.43		853.3
MAPAA #9	GA	5	2	100	100	0.63	0.89	1	1	852.1
MAPAA #10	TS	1	1	0.16	4	0.43	1.57	0.06		854.1
MAPAA #11	GA	2	5	100	10	0.27	0.83	1	1	853.7
MAPAA #12	TS	1	0	0.00	8	0.30	1.39	0.44		853.7
MAPAA #13	TS	1	0	0.01	6	0.23	1.98	0.33		853.9
MAPAA #14	TS	1	1	0.04	4	0.79	2.13	0.07		853.4
MAPAA #15	TS	1	0	0.07	4	0.65	1.24	0.13		851.6
MAPAA #16	TS	5	0	0.00	4	0.23	2.39	0.27		854.5
										f_{avg}: $mean = 853.3$, $\sigma = 0.89$

Table 6.13: Experimental results for a Tabu Search/Genetic Algorithm for the JSSP

With regard to the actual parameter value choices, the observations are similar to the findings in the respective single algorithm studies. The interested

reader should refer to the corresponding sections for a thorough discussion.

The results of the 16 MAPAA runs are again tested for the same 100 instances as in the earlier JSSP scenarios. The achieved average of $f_{avg} = 853.3$ is slightly worse than the single algorithm adaptation results. This outcome is expected as the MAPAA's computation resources have to be split between the two heuristic search techniques. As both methods are very similar in terms of performance, our adaptation technique optimises both at the same time: the best parameter settings for the second placed, non-winning algorithms in the 16 experimental runs average 853.5 for the 100 JSSP test instances.

With a small standard deviation of just 0.89, the MAPAA is once more very robust in producing high quality solutions. All 16 runs significantly improve the results achieved with the initial parameter settings for both the GA and the Tabu Search. Equally, they all clearly outperform the SPO result for the GA. Only the result of the SPO process for the Tabu Search comes close with $f_{avg} = 853.6$ so that a significant difference cannot be proven.

Conclusion

The MAPAA shows in this final application that it is very robust and efficient when confronted with two local search methods with similar performance. By chance, either of the two algorithms can be part of the final recommendation. Though slightly inferior to the results achieved when adapting just a single method, the produced solutions are of high quality.

6.5 Parameter Adaptation and iOpt

[Voudouris and Dorne, 2002] describe iOpt as "a large software system comprising several libraries and frameworks dedicated to the development of combinatorial optimisation applications based on Heuristic Search." This toolkit is the result of an ongoing research project at the BT Laboratories.

A central component of iOpt is an algorithm modeling framework named Heuristic Search Framework (HSF) [Dorne and Voudouris, 2004]. "The main idea in HSF is to break down ... heuristic algorithms into a plurality of constituent parts." These parts facilitate the quick and efficient development of existing heuristic search methods for a wide variety of applications. Novel recombinations of such modules even allow the building of entirely new heuristic algorithms. Due to the flexibility of the aforementioned framework, components can be easily extended to accommodate new concepts.

The iOpt package provides a tool for visually building heuristic search algorithms. This tool represents such an algorithm as a tree of modules from the Heuristic Search Framework. The user can graphically modify this tree by adding, replacing or removing components. Knowledge about internal details of the modules is not required.

Such modules possess parameters that determine their exact operation. As an example, the component modeling a tabu list has a parameter that defines the length of this data structure. Although these parameters are set to default values, these choices are not always good. Components are highly reusable,

and different applications might require different parameter settings.

The desire to free the user from having to tune parameters by hand has motivated the realisation of the Multiple Algorithm's Parameter Adaptation Algorithm within iOpt. A program has been developed that supports the optimisation of parameter values. With this tool, the user can choose one or more heuristic search algorithms. These methods are graphically displayed, and all parameters of their modules are listed. By simply marking them, parameters are selected for the adaptation process. After defining domains for the selected parameters and providing files with sample problem instances for the heuristic search methods, the tool employs the MAPAA and automatically finds good parameter settings. A number of graphical monitors are provided to facilitate the observation of the adaptation process. After the completion of the parameter adaptation phase, the heuristic search methods and the established parameter choices can be saved for later reuse.

Our algorithm optimisation tool contributes to the user-friendliness of iOpt. Combined with the tool for visually building algorithms, it allows a fast and easy development of efficient heuristic search methods for new scenarios. Planned improvements for the future include a better cooperation of and interaction between these two tools, and the realisation of default domains for parameters of iOpt's search modules in order to prevent the need for users to specify such domains before the parameter adaptation processes.

6.6 Summary

This chapter focused on the application of the Multiple Algorithms' Parameter Adaptation Algorithm (MAPAA) to heuristic search methods. We applied our method to find good parameter settings for a Simulated Annealing algorithm for Travelling Salesman Problems, for a Tabu Search method and a Genetic Algorithm for Vehicle Routing, and for a Genetic Algorithm and a Tabu Search for Job Shop Scheduling. Furthermore, the techniques for Vehicle Routing and Job Shop Scheduling were combined in additional scenarios.

The experimental results strongly suggest that the MAPAA consistently detects very good parameter settings. Our mechanism never performed worse than the more basic Single Parameter Optimisation approach, and outperformed it in the majority of the presented case studies. We also showed that the MAPAA is very robust in choosing the best among two available heuristic search methods or, if both algorithms are very similar in their performance, optimises both at the same time and then chooses one of the methods.

To summarise, the novel approach proposed is an efficient, mainly autonomously working technique for tuning parameters of heuristic search methods. It has been successfully integrated into the iOpt package, BT's toolkit for the development of heuristic search applications.

Chapter 7

Summary

This thesis introduced and analysed the Multiple Algorithms' Parameter Adaptation Algorithm, an approach for tuning parameters of heuristic search methods. The final chapter is a summary of this study. It looks at the work presented, lists the main contributions, discusses limitations and takes an outlook on further research.

7.1 Summary of the Presented Work

In the first chapter, we described the general optimisation problem and presented heuristic search as a class of algorithms to tackle it. Motivated by the fact that the effectiveness of these methods often depends on good choices for their parameters, we defined the problem of parameter adaptation. In its

most general formulation as the Multiple Algorithms' Parameter Adaptation Problem (MAPAP), the task is to find the best algorithm among a number of heuristic search methods together with the best parameter choices. The introductory section furthermore listed characteristics which make this problem particularly difficult to solve.

Chapter 2 saw a review of existing approaches for the tuning of parameters in heuristic search methods. We briefly introduced two general techniques, alternating variable search and the experimental design approach, which could be applied to this problem scenario, and then gave an overview of parameter adaptation methods for Genetic Algorithms as exponents of the class of heuristic search algorithms. The chapter concluded with a discussion of the No Free Lunch Theorem in the context of parameter adaptation.

The aim of the third chapter was to discuss Population-Based Incremental Learning (PBIL), an approach that successfully combines the concepts of Competitive Learning and Genetic Algorithms. PBIL is a generational technique that works with a population of candidate solutions. Its main innovation is that it maintains a probability distribution over a binary solution space. In each cycle, new candidate solutions are sampled based on this distribution and entirely replace the old population. The best population member with regard to a fitness function is then used to update the probabilities. Through this learning process, the probability distribution represents high quality solutions over time.

Based on a generalised formulation of the PBIL technique, the Multiple Al-

gorithms' Parameter Adaptation Algorithm (MAPAA) was introduced in chapter four. This method, using only small populations and relatively few generational cycles, can optimise heuristic search parameters that have either finite or interval domains. When applied to several heuristic searches simultaneously, it concentrates its resources on the better performing algorithms. The central element of the MAPAA is the maintenance of joint probability distributions of the considered parameter spaces. Through sampling of and both positive and negative learning from small sets of candidate parameter configurations, the probability distributions are constantly updated. This mechanism allows the search to focus on promising areas of the search space.

Chapter five saw a rigorous examination of the main techniques employed in the MAPAA. We analysed how the positive learning schemes help to intensify the search around promising solutions and how negative feedback works as a diversification mechanism. Furthermore, the examination of mutation showed how moderate amounts can be successfully incorporated into the learning strategy. These experiments led to the determination of a successful MAPAA setup.

In chapter six, seven applications of the MAPAA were studied. The considered heuristic search methods included Simulated Annealing (SA) for the Travelling Salesman Problem (TSP), and Tabu Search (TS) and Genetic Algorithms (GA) for Vehicle Routing (VRP) and Job Shop Scheduling Problems (JSSP). While these five cases involved the optimisation of a single heuristic search, two further studies looked at improving tow of the search

techniques together in multiple algorithms scenarios. The MAPAA never performed worse than a more basic benchmark approach, and often outperformed it.

Table 7.1 summarises all experiments with and applications of the MAPAA covered in this thesis.

7.2 Contributions

The main contribution of this work is the introduction of an incremental learning mechanism for tackling the problem of parameter adaptation in heuristic search. The MAPAA constitutes both a significant extension and a specialisation of the standard PBIL technique. It is novel and significant because[1]:

- **Versatile search spaces:** The MAPAA extends the scope of the PBIL method to cope with more complex search spaces:
  - **Multiple values:** The basic learning rule for binary variables has been extended to deal with finite variables with more than two possible values (experiments I, II and V).
  - **Interval domains:** For variables with an interval domain, a completely new learning scheme based on Self-Organising Maps has been introduced (experiments III, IV and VI).

[1] An overview of all experiments and applications referred to is given in table 7.1.

- **Noisy data:** PBIL has been extended so that it can manage noisy problem data. Our approach incorporates a mechanism for transfering good candidate solutions to new generations so that a memory of their performance characteristics can be compiled and used to reduce the influence of noise.

- **Small population sizes:** One major element of the motivation for the MAPAA were the high computational costs of heuristic search runs, that are therefore limited in number. Consequently, the MAPAA uses much smaller population sizes and fewer generational cycles than PBIL, and its learning mechanism utilises information from all population members as either positive or negative feedback.

- **Effectiveness:** The MAPAA consistently shows good performance in both parameter tuning (applications I to VII) and algorithm selection (applications IV and VII) when applied to tackle a MAPAP.

7.3 Limitations

Although the MAPAA performs well when applied in its domain, it has limitations, some of which are addressed below:

- In our experiments, the MAPAA was only compared to the basic Single Parameter Optimisation (SPO) approach. Performance comparisons with other, more advanced techniques, such as experimental design

based methods, are currently not available.

- Although the MAPAA performs well in all test cases presented, the performance gains over the basic SPO procedure are sometimes only marginal. The high computational requirements of the MAPAA can often only be justified in scenarios where even small gains are of central importance. In cases where such small gains can be neglected but computational costs play a more prominent role, other parameter adaptation approaches might be better suited.

- The joint probability distribution maintained by the MAPAA is unconditional, i.e. distributions of different parameters are treated independently from each other. This is, of course, a simplifying assumption. The MAPAA in its current formulation is not capable of directly identifying interdependencies between parameters. Whether the MAPAA is able to find high performance parameter settings in test scenarios with strong parameter interdependencies remains unclear for the moment.

- The MAPAA in its current formulation does not try to approximate the landscape of the parameter space with regard to the objective function. Such information could, however, be very useful in guiding the search towards better regions of the parameter space.

- Each heuristic search method is treated as a black box. In its current formulation, our adaptation approach cannot make use of additional knowledge of the search algorithm.

- The MAPAA was specifically designed and studied for the case of pa-

rameter adaptation in heuristic search. Although many of the novelties within this algorithm have the potential to be successfully applied to a multitude of further optimisation scenarios, statements about their performance can currently only be substantiated for the original task.

7.4 Future Research

A thorough and detailed study of the MAPAA has been presented in the thesis. But this challenging research area offers many more questions of scientific interest. The following outlook summarises a selection of possible future research directions.

- A crucial direction for future research is a more detailed comparison of the MAPAA with alternative parameter adaptation approaches. Instead of comparing it only with the basic SPO method, further experiments should be conducted to compare the MAPAA with alternating variable search and experimental design techniques.

- [Shapiro, 2002, Shapiro, 2003] and [Yang and Yao, 2003] propose interesting extensions and improvements of the standard PBIL algorithm. Future research could aim to incorporate these modifications into the MAPAA. It would be interesting to see whether our adaptation approach can benefit from concepts like detailed balance or dual populations.

- A simplification and thus limitation of the MAPAA is that probabilities for different parameters are implemented as being independent. How could one extend the technique to handle dependencies between parameters directly? This is an area of high research activity, examples of which include [Pelikan and Mühlenbein, 1999]. Can existing techniques be adapted so that they successfully work with the limited information available in the MAPAA?

  As the MAPAA relies on small populations and relatively few generational cycles, the amount of available information about inter-parameter relations is extremely limited. Further research into this direction should therefore initially look at how existing approaches can cope with reduced amounts of information and, if not, how they could be successfully adapted.

- The high cost in computation time when using heuristic search methods resulted in specific MAPAA configurations with small populations, few generations and relatively high learning rates. If these resource restrictions became less important, how much better would the MAPAA perform given the chance to employ larger populations, more generational cycles and smaller, better suited rates of learning?

  With the increasing availability of powerful parallel computer technology, the experiments described in this thesis could be repeated with larger populations of sizes of 100 or above and for 500 or more generations. We expect that more moderate rates of learning would be beneficial in such scenarios and that MAPAA results could be further

improved both in efficiency and robustness.

- As noted in the previous section on limitations, our adaptation technique sees each heuristic search as a black box. Is it possible to make the optimisation process more informed, i.e. include problem specific knowledge about the impact and relationship of parameters? How could that be achieved?

  A starting point could be the examination of one particular class of heuristic search methods, e. g. of GAs. Through analysing correlations between their parameters, one could try to find and formulate patterns. This would allow the development of more informed and thus improved learning mechanisms which take account of the aforementioned parameter dependencies.

Name	Section	Subject
Exp I	5.1.1	Study of how positive learning over finite domains leads to the estimation of target probability distributions
Exp II	5.1.1	Study of how negative learning over finite domains works as a diversification mechanism
Exp III	5.1.2	Study of how positive learning over interval domains leads to the estimation of target probability distributions
Exp IV	5.1.2	Study of how negative learning over interval domains works as a diversification mechanism
Exp V	5.2.3	Determination of effective learning rates for finite domains
Exp VI	5.2.4	Determination of effective learning rates for interval domains
Exp VII	5.2.5	Determination of effective mutation rates
Exp VIII	5.2.6	Study of how resources are allocated efficiently and intelligently in scenarios with multiple heuristic searches
Exp IX	5.2.7	Justification of the applicability of the statistical t-test
Appl I	6.2.2	Study of a basic parameter adaptation scenario by applying the MAPAA to a SA method for the TSP
Appl II	6.3.2	Study of a more complex parameter adaptation scenario by applying the MAPAA to a TS for the VRP
Appl III	6.3.3	Study of a more complex parameter adaptation scenario by applying the MAPAA to a GA for the VRP
Appl IV	6.3.4	Study of a multiple algorithms scenario in which one heuristic search is clearly superior by applying the MAPAA simultaneously to a TS and a GA for the VRP
Appl V	6.4.2	Further study of a more complex parameter adaptation scenario by applying the MAPAA to a GA for the JSSP
Appl VI	6.4.3	Further study of a more complex parameter adaptation scenario by applying the MAPAA to a TS for the JSSP
Appl VII	6.4.4	Study of a multiple algorithms scenario in which all heuristic searches are comparable in their performance by applying the MAPAA simultaneously to a GA and a TS for the JSSP

Table 7.1: Summary of experiments

Appendix A

Notation

A.1 Abbreviations

CL	Competitive Learning
DOE	Design of Experiments
EDA	Estimation of Distribution Algorithm
GA	Genetic Algorithm
GPBIL	Generalised Population-Based Incremental Learning
JSSP	Job Shop Scheduling problem
MAPAP	Multiple Algorithms' Parameter Adaptation Problem
NP	nondeterministic polynomial
PAP	Parameter Adaptation Problem
PBIL	Population-Based Incremental Learning

PDF	probability distribution function
PSP	Parameter Setting Problem
SA	Simulated Annealing
SOM	Self-Organising Map
SPO	Single Parameter Optimisation
TS	Tabu Search
TSP	Travelling Salesman Problem
VRP	Vehicle Routing Problem

A.2 Mathematical Symbols

Spaces are always written in $\mathbb{BLACKBOARD} - \mathbb{BOLD}$ letters:

$\mathbb{N}$	natural numbers
$\mathbb{R}$	real numbers
$\mathbb{S}$	solution space
$\tilde{\mathbb{S}}$	space of all possible solution spaces
$\mathbb{D}$	domain
$\mathbb{P}$	parameter space
$\mathbb{E}$	evaluation space
$\mathbb{I}$	individual space
$\mathbb{F}$	space of all probability distribution functions
$\mathbb{H}$	history space
Φ	optimisation problem

f	objective function
$s, \vec{s}$	solution, solution vector
A	heuristic search algorithm
$\mathcal{P}$	parameter of an algorithm
π	parameter setting
$I = \langle \vec{s}, \vec{e} \rangle$	individual, i.e. a population member in GPBIL, pair of a solution $\vec{s} \in \mathbb{S}$ and an evaluation history $\vec{e} \in \mathbb{H}$

By adding a superscript ^{best} we indicate the best found item, e.g. I^{best} represents the best encountered individual in the GPBIL algorithm.

A.3 Population-Based Incremental Learning

PBIL Algorithm

$\vec{s}^{best} = \emptyset$
FOR $l = 1, \ldots, L$ DO $P_l = 0.5$
FOR $g = 1, \ldots, G$ DO
    FOR $i = 1, \ldots, M$ DO
        $\vec{s}_i = sampling(\vec{P})$
        $e_i = f(\vec{s}_i)$
    $min = argmin_{i=1}^{M} e_i$
    FOR $l = 1, \ldots, L$ DO
        $P_l = P_l \cdot (1 - \epsilon_1) + \vec{s}_{min,l} \cdot \epsilon_1$
        IF $random((0, 1]) < \mu$ THEN
            $P_l = P_l \cdot (1 - \epsilon_2) + random(\{0, 1\}) \cdot \epsilon_2$
    $\vec{s}^{best} = minimum(\vec{s}^{best}, \vec{s}_{min})$
RETURN $\vec{s}^{best}$

Figure A.1: The Population-Based Incremental Learning algorithm

Variables:

$\vec{P} = (P_1, \ldots, P_L)$ - probability vector

$\vec{s}_1, \ldots, \vec{s}_M$ - candidate solutions in the population

$e_1, \ldots, e_M$ - evaluations of the solution vectors

min - index of the solution with the minimal evaluation

$\vec{s}^{best}$ - best solution found so far in all generations

l, g, i - loop variables

Constants:

L - bit length of the encoded solutions

G - number of generational cycles

M - population size

ϵ_1 - learning rate

μ - mutation probability (for each bit position)

ϵ_2 - amount of mutation

Functions:

$sampling(\vec{P})$ - generate a sample solution according to $\vec{P}$

$f(\vec{s}_i)$ - objective evaluation of solution $\vec{s}_i$

A.4 Generalised Population-Based Incremental Learning

Notation:

I - each individual is a tuple $\langle \vec{s}, \vec{e} \rangle$ with $\vec{s} \in \mathbb{S}$ and $\vec{e} \in \mathbb{H}$

$\biguplus$ - adds an element to the evaluation history of an individual

GPBIL Algorithm

1 $I^{best} = \langle \vec{s}_{initial}, \emptyset \rangle$
2 $\mathcal{F}_{\mathbb{S}} = initialisation(\mathbb{S})$
3 $N = M$
4 FOR $g = 1, \ldots, G$ DO
5 FOR $i = M - N + 1, \ldots, M$ DO $I_i = \langle sampling(\mathcal{F}_{\mathbb{S}}), \emptyset \rangle$
6 FOR $i = 1, \ldots, M$ DO $I_i = I_i \biguplus_g evaluation(I_i, g)$
7 $I^{best} = I^{best} \biguplus_g evaluation(I^{best}, g)$
8 $\mathcal{F}_{\mathbb{S}} = learning(\mathcal{F}_{\mathbb{S}}, I^{best}, \vec{I}, g)$
9 $I^{best} = optimum(I^{best}, \vec{I})$
10 $\vec{I} = selection(I^{best}, \vec{I})$
11 $N = M - size(\vec{I})$
12 RETURN I^{best}

Figure A.2: The Generalised Population-Based Incremental Learning algorithm

Variables:

$\mathcal{F}_{\mathbb{S}}$- joint probability distribution over solution space

$\vec{I} = (I_1, \ldots, I_M)$ - population

I^{best} - best individual found so far in all generations

N - number of individuals to be sampled

g, i - loop variables

Constants:

$\mathbb{S}$ - solution space

$\mathbb{D}_i$ - domain for the i-th position in the encoded solution

G - number of generational cycles

M - population size

$\vec{s}_{initial}$ - initial solution

Functions:

$initialisation(\mathbb{S})$ - initialise the joint probability distribution over $\mathbb{S}$

$sampling(\mathcal{F}_{\mathbb{S}})$ - generate a sample solution vector according to $\mathcal{F}_{\mathbb{S}}$

$evaluation(I, g)$ - evaluate the individual I in generation g

$learning(\mathcal{F}_{\mathbb{S}}, I^{best}, \vec{I}, g)$ - update the joint probability distribution $\mathcal{F}_{\mathbb{S}}$

$optimum(I^{best}, \vec{I})$ - select the optimal solution

$selection(I^{best}, \vec{I})$ - select individuals for next generation

Appendix B

Mathematical Concepts

B.1 Random Variables and Probability Distribution Functions

B.1.1 Random Variables

In general probability theory [Bronstein and Semendjajew, 1991, Tucker, 1962], a random variable is usually defined as a real variable that can take different values depending on the random outcome of an experiment. Typical examples for such variables are the random results of rolling a dice, or measuring a car's speed. The set of all values such a variable can take is called its range or domain. In the above examples, the ranges are $\{1, 2, 3, 4, 5, 6\}$, and, for instance, $[0, 100]$ (mph).

A probability distribution function (PDF) $\mathcal{F}$ is associated with every random variable X. This function characterises a random variable completely. For each $x \in \mathbb{R}$, it yields the probability of the random variable taking a value less than x:

$$\mathcal{F}(x) = P(X < x).$$

With the term probability distribution over a set S we refer to the PDF of a random variable with the range S.

B.1.2 Discrete Random Variables

A random variable X and its PDF $\mathcal{F}$ are called discrete if the range of X is finite or denumerable. Such a random variable is characterised by the values $x_1, x_2, \ldots$ it can take, and the corresponding probabilities $p_i = P(X = x_i)$. Of course, $\sum_i p_i = 1$ has to be satisfied. The PDF is defined as

$$\mathcal{F}(x) = \sum_{x_i < x} p_i.$$

Example B.1. In the case of rolling a dice, we have the six possible outcomes 1, 2, 3, 4, 5 and 6. All results are equally likely, and thus $p_1 = p_2 = \ldots =$

$p_6 = \frac{1}{6}$ holds. The PDF is consequently given as

$$\mathcal{F}(x) = \begin{cases} 0 & \text{if } x \leq 1 \\ \frac{1}{6} & \text{if } 1 < x \leq 2 \\ \dots & \\ \frac{5}{6} & \text{if } 5 < x \leq 6 \\ 1 & \text{if } 6 < x \end{cases}.$$

B.1.3 Continuous Random Variables

A random variable X is called continuous, if its PDF $\mathcal{F}$ can be written as

$$\mathcal{F}(x) = \int_{-\infty}^{x} d(t)dt$$

where d is continuous. In this case, the function d is called the density function of X, and it has to satisfy

$$\int_{-\infty}^{\infty} d(t)dt = 1.$$

One can easily calculate the probability that the random variable falls into a given interval $[a, b)$:

$$P(a \leq X < b) = \int_{a}^{b} d(t)dt.$$

This formula can be graphically interpreted. The probability that X falls into $[a, b)$ equals the area under the density function over this interval.

Example B.2. We assume that a random variable X can take all values from the interval $[0, 100]$, e.g. by measuring a car's speed. If all results are "equally likely", then the density function is given as

$$d(x) = \begin{cases} \frac{1}{100} & \text{if } x \in [0, 100] \\ 0 & \text{if } x \notin [0, 100] \end{cases},$$

i.e. it is constant over the considered interval and zero elsewhere. The resulting PDF can hence be written as

$$\mathcal{F}(x) = \int_{-\infty}^{x} d(t)dt = \begin{cases} 0 & \text{if } x < 0 \\ \frac{1}{100}x & \text{if } 0 \leq x \leq 100 \\ 1 & \text{if } 100 < x \end{cases} \cdot$$

B.1.4 Joint Probability Distributions

A number of random variables $X_1, \ldots, X_n$ are said to form an n-dimensional random vector $(X_1, \ldots, X_n)$. This random vector can be characterised by an n-dimensional joint probability distribution function

$$\mathcal{F}(x_1, \ldots, x_n) = P(X_1 < x_1, \ldots, X_n < x_n).$$

$X_1, \ldots, X_n$ are called independent if

$$\mathcal{F}(x_1, \ldots, x_n) = \mathcal{F}_1(X_1 < x_1) \cdot \ldots \cdot \mathcal{F}_n(X_n < x_n)$$

where $\mathcal{F}_i$ is the PDF of X_i. That means that the joint probability distribution function $\mathcal{F}$ is fully described by the distribution functions $\mathcal{F}_1$ to $\mathcal{F}_n$.

B.2 One-Sample and One-Tailed t-Test

The t-tests are a family of statistical tests that are used to analyse sample data and to draw conclusions about the sample means. The one-sample and one-tailed t-test is applied in the following situation:

Problem: A single data sample $x_1, \ldots, x_n$ of n real numbers, which is uniformly distributed[1], is given. The assumption that this sample is not drawn from a population with a known and given mean m, but rather has a smaller mean μ has to be probed statistically.

The t-test for this case consists of four major steps:

1. **Hypotheses:** The first step of the t-test is to set up the null and the alternative hypotheses. The null hypothesis is that the population mean μ takes the value m: $H_0 : \mu = m$, while the alternative hypothesis

[1] [Bronstein and Semendjajew, 1991] points out that the t-test can be applied in practise as long as the frequency distribution of the data shows a single peak and is not too skewed.

states that the the population mean is smaller than m: $H_1 : \mu < m$.

2. **Establish critical t-value:** The critical t-value $t_{critical}$ can be taken from tables, e.g. as given in [Bronstein and Semendjajew, 1991, Ehrenberg, 1986]. It depends firstly on the kind of t-test being used, here one-tailed, secondly on the chosen significance level α (which is a measure for the degree of reliability of the t-test), and thirdly on the degree of freedom DF of the sample data which is given as the sample size minus one: $DF = n - 1$.

3. **Calculate t-statistics:** Central to the t-test is the calculation of the t-statistics $t_{calculated} = \frac{\bar{x}-m}{s/\sqrt{n}}$. In this formula, $\bar{x}$ is the sample mean and is given by $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$, and s is the estimated standard deviation of the sample which can be established as $s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$.

4. **Conclusion:** By comparing the t-statistics with the critical t-value, we can accept or reject the set up hypotheses. If the absolute value of the calculated t-value is smaller than or equal to the critical t-value, $|t_{calculated}| \leq t_{critical}$, then we accept the null hypothesis. But if the absolute value of the calculated t-value is larger than the critical t-value and the calculated t-value itself is negative, $|t_{calculated}| > t_{critical}$ and $t_{calculated} < 0$, then we reject the null hypothesis and accept the alternative. In the latter case, we say that the population mean μ is significantly smaller than the hypothesized mean m with a significance level of α.

Example B.3. The following example illustrates this procedure:

Problem: We consider the sample 6, 11, 10, 8, 8, 9, 7, 12, 8 and 9 and want to decide whether the corresponding population mean is significantly smaller than $m = 10$ with a significance level of $\alpha = 0.05$.

Step 1: Null hypothesis $H_0 : \mu = 10$, alternative hypothesis $H_1 : \mu < 10$.

Step 2: The critical t-value for a one-tailed t-test with a significance level of 0.05 and a degree of freedom of $DF = 10 - 1 = 9$ is $t_{critical} = 1.83$.

Step 3: We calculate the mean of the sample data as $\bar{x} = 8.8$, and compute an estimated standard deviation of the sample of $s = 1.81$. These results yield a t-statistics of $t_{calculated} = -2.09$.

Step 4: As we find $|t_{calculated}| = 2.09 > 1.83 = t_{critical}$, we reject the null hypothesis and, as $t_{calculated} < 0$, accept the alternate hypothesis instead. The population mean is therefore significantly smaller than 10, and the difference is not due to chance at a 0.05 level of significance.

Appendix C

Self-Organising Maps

Self-Organising Maps (SOMs) are neural networks with one layer of active neurons. The basic idea is to train the neuron weight vectors through unsupervised learning such that they cover the space of the input vectors effectively. The neurons are arranged in a lattice that leads to neighbourhood relationships between neurons. Here, we consider only the case of a one-dimensional chain of neurons. For more detailed information about the general case refer to the original work [Kohonen, 1995].

n neurons, each associated with a weight vector $\vec{W}_i$, are arranged in a chain. Neuron 1 is the first, neuron n the last member of the chain. During the learning phase, k input vectors $\vec{X}_1, \ldots, \vec{X}_k$ are successively presented to the network. If the input vector $\vec{X}_i$ is presented, the weight vectors of the neurons are compared to this input using a metric (e.g. the Euclidean distance). The

neuron, c, that is most similar to the input wins [Zell, 1994]:

$$c = argmin_{j=1}^{n}(||\vec{X}_i - \vec{W}_j||).$$

The weight of this winning neuron is now modified so that it better matches the input. Neurons neighbouring the winner are also adapted through this process although to a less degree. This can be summarised in the following formula, where t stands for t-th training step:

$$\vec{W}_j(t+1) = \vec{W}_j(t) + \eta(t)h_{cj}(t)(\vec{X}_t - \vec{W}_j(t)).$$

$\eta(t)$ is a time dependent learning rate which usually decreases over time from 1 to 0. $h_{cj}(t)$ is the neighbourhood kernel which determines to what extent weights get adapted depending on the distance to the winning neuron. Due to the chain structure of the neurons this function can be written as:

$$h_{cj}(t) = h(|c-j|, t).$$

A typical example is the cylinder neighbourhood kernel:

$$h(|c-j|,t) = h_\delta(|c-j|) = h_{cylinder}(|c-j|,\delta) = \begin{cases} 1 & \text{if } |c-j| \leq \delta \\ 0 & \text{else} \end{cases}$$

Appendix D

Experimental Results for Single Parameter Optimisation

At each stage of the single parameter optimisation process, the best parameter value is highlighted in bold.

D.1 Application I

Results are averaged over 100 TSP instances.

Initial parameter setting:

Parameter	$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_5$
Value	1	1	10	0.5	0.5

Number of restarts:

$\mathcal{P}_1$	**1**	2	5
f_{avg}	**11.99**	12.23	12.81

Neighbourhood operators:

$\mathcal{P}_2$	1	2	3	4	5	**6**	7
f_{avg}	11.99	10.04	10.24	9.48	9.55	**9.40**	9.45

Thermalisation:

$\mathcal{P}_3$	1	**10**	100	1000	10000	100000
f_{avg}	9.43	**9.40**	9.42	9.42	9.43	9.42

Initial acceptance probability:

$\mathcal{P}_4$	0.01	0.1	0.2	0.3	0.4	**0.5**	0.6	0.7	0.8	0.9	0.99
f_{avg}	9.40	9.40	9.41	9.42	9.42	**9.40**	9.42	9.43	9.42	9.43	9.46

Final acceptance probability:

$\mathcal{P}_5$	0.01	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	**0.99**
f_{avg}	9.43	9.44	9.43	9.46	9.41	9.40	9.43	9.42	9.39	9.43	**9.38**

Final parameter setting:

Parameter	$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_5$
Value	1	6	10	0.5	0.99

D.2 Application II

Results are averaged over 100 VRP instances.

Initial parameter setting:

Parameter	$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_5$	$\mathcal{P}_6$	$\mathcal{P}_7$
Value	1	7	1	1	0.0	10	0

Number of restarts:

$\mathcal{P}_1$	1	2	**5**
f_{avg}	1431.3	1416.8	**1408.8**

Neighbourhood operators:

$\mathcal{P}_2$	1	2	3	4	5	6	**7**
f_{avg}	1860.3	1476.4	1410.7	1448.4	1430.5	1412.9	**1408.8**

Stepping factor:

$\mathcal{P}_3$	**1**	2	4	8	16
f_{avg}	**1408.8**	1410.5	1414.7	1421.4	1431.5

Aspiration:

$\mathcal{P}_4$	**0**	1
f_{avg}	**1408.7**	1408.8

Random move probability:

$\mathcal{P}_5$	0.00	0.05	0.10	**0.15**	0.20	0.30	0.50
f_{avg}	1408.7	1404.6	1405.2	**1403.9**	1404.3	1412.9	1591.7

Tabu list length:

$\mathcal{P}_6$	5	10	15	25	**35**	50	75
f_{avg}	1404.7	1403.9	1403.2	1403.4	**1402.3**	1403.1	1402.7

Tabu list length variation:

$\mathcal{P}_7$	**0**	1	2	5	10	20	50
f_{avg}	**1402.3**	1402.7	1402.4	1403.0	1402.9	1403.3	1402.8

Final parameter setting:

Parameter	$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_5$	$\mathcal{P}_6$	$\mathcal{P}_7$
Value	5	7	1	0	0.15	35	0

D.3 Application III

Results are averaged over 100 VRP instances.

Initial parameter setting:

Parameter	$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_5$	$\mathcal{P}_6$	$\mathcal{P}_7$
Value	1	7	100	1	1	10	1

Number of restarts:

$\mathcal{P}_1$	**1**	2	5
f_{avg}	**1456.7**	1473.1	1506.8

Neighbourhood operators:

$\mathcal{P}_2$	1	2	3	4	5	6	**7**
f_{avg}	1517.7	1545.1	1467.5	1465.1	1458.9	1459.1	**1456.7**

Population size:

$\mathcal{P}_3$	**50**	100	150	200	250
f_{avg}	**1442.0**	1456.7	1473.1	1484.3	1490.9

Elitism:

$\mathcal{P}_4$	0	**1**
f_{avg}	1701.2	**1442.0**

Better initial tours:

$\mathcal{P}_5$	0	**1**
f_{avg}	1449.3	**1442.0**

Parenthood proportion:

$\mathcal{P}_6$	**5**	10	15	20	30	40	50
f_{avg}	**1428.9**	1442.0	1447.3	1462.8	1478.0	1500.0	1518.1

Replacement scheme:

$\mathcal{P}_7$	0	**1**
f_{avg}	1435.8	**1428.9**

Final parameter setting:

Parameter	$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_5$	$\mathcal{P}_6$	$\mathcal{P}_7$
Value	1	7	50	1	1	5	1

D.4 Application V

Results are averaged over 50 JSSP instances.

Initial parameter setting:

Parameter	$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_5$	$\mathcal{P}_6$	$\mathcal{P}_7$	$\mathcal{P}_8$
Value	1	1	100	50	0.6	0.1	1	0

Number of restarts:

$\mathcal{P}_1$	1	**2**	5
f_{avg}	879.8	**879.7**	880.8

Number of subpopulations:

$\mathcal{P}_2$	1	**2**	5	10
f_{avg}	879.7	**879.1**	888.9	897.1

Size of subpopulations:

$\mathcal{P}_3$	10	20	**50**	100	200
f_{avg}	879.2	875.9	**872.1**	879.1	887.6

Migration:

$\mathcal{P}_4$	0	**10**	20	50	100
f_{avg}	873.9	**870.2**	872.1	872.1	873.6

Crossover rate:

$\mathcal{P}_5$	0.0	0.1	0.2	0.3	0.4	0.5
f_{avg}	885.4	874.9	872.5	873.3	873.3	872.2
$\mathcal{P}_5$	**0.6**	0.7	0.8	0.9	1.0	
f_{avg}	**870.2**	873.6	871.8	872.6	875.2	

Mutation rate:

$\mathcal{P}_6$	0.0	0.1	0.2	0.3	0.4	0.5
f_{avg}	898.2	870.2	865.3	863.7	865.7	863.7
$\mathcal{P}_6$	0.6	**0.7**	0.8	0.9	1.0	
f_{avg}	864.6	**857.1**	861.9	859.3	861.6	

Elitism:

$\mathcal{P}_7$	0	**1**
f_{avg}	889.1	**857.1**

Selection mechanism:

$\mathcal{P}_8$	0	**1**
f_{avg}	857.1	**852.6**

Final parameter setting:

Parameter	$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_5$	$\mathcal{P}_6$	$\mathcal{P}_7$	$\mathcal{P}_8$
Value	2	2	50	10	0.6	0.7	1	1

D.5 Application VI

Results are averaged over 50 JSSP instances.

Initial parameter setting:

Parameter	$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_5$	$\mathcal{P}_6$	$\mathcal{P}_7$
Value	1	1	0.0	13	0.8	1.2	0.5

Number of restarts:

$\mathcal{P}_1$	**1**	2	5
f_{avg}	**847.6**	848.3	849.2

Aspiration:

$\mathcal{P}_2$	0	**1**
f_{avg}	852.4	**847.6**

Random move probability:

$\mathcal{P}_3$	**0.00**	0.05	0.1	0.15	0.20	0.25
f_{avg}	**847.6**	850.8	853.0	855.1	857.7	857.4
$\mathcal{P}_3$	0.30	0.35	0.4	0.45	0.50	
f_{avg}	859.2	862.4	862.0	866.4	869.4	

Tabu list length:

$\mathcal{P}_4$	4	6	**8**	10	13
f_{avg}	847.7	844.2	**844.0**	844.5	847.6
$\mathcal{P}_4$	15	20	25	35	50
f_{avg}	850.9	858.4	868.3	874.1	887.1

Minimal tabu list length factor:

$\mathcal{P}_5$	**0.0**	0.1	0.2	0.3	0.4	0.5
f_{avg}	**842.6**	844.2	843.1	843.1	845.2	844.2
$\mathcal{P}_5$	0.6	0.7	0.8	0.9	1.0	
f_{avg}	843.8	844.0	844.0	846.8	845.8	

Maximal tabu list length factor:

$\mathcal{P}_6$	1.0	**1.2**	1.4	1.8	2.2	2.6
f_{avg}	848.2	**842.6**	845.2	846.6	845.9	847.7
$\mathcal{P}_6$	3.0	3.4	3.8	4.2	4.6	5.0
f_{avg}	849.1	848.1	852.4	853.5	856.9	855.2

Penalty factor:

$\mathcal{P}_7$	0.0	0.2	0.4	**0.5**	0.6	0.8
f_{avg}	845.8	844.5	845.1	**842.6**	845.1	845.2
$\mathcal{P}_7$	1.0	1.2	1.4	1.6	1.6	2.0
f_{avg}	846.6	845.6	850.8	846.3	846.7	848.6

Final parameter setting:

Parameter	$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_5$	$\mathcal{P}_6$	$\mathcal{P}_7$
Value	1	1	0.0	8	0.0	1.2	0.5

Bibliography

[Badeau et al., 1997] Badeau, P., Gendreau, M., Guertin, F., Potvin, J.-Y., and Taillard, E. D. (1997). A Parallel Tabu Search Heuristic for the Vehicle Routing Problem with Time Windows. *Transportation Research*, 5:109–122.

[Baeck, 1993] Baeck, T. (1993). Optimal Mutation Rates in Genetic Search. In Forrest, S., editor, *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 2–8, San Mateo, CA, USA. Morgan Kaufmann.

[Bagley, 1967] Bagley, J. D. (1967). *The Behavior of Adaptive Systems which employ Genetic and Correlation Algorithms.* PhD thesis, University of Michigan.

[Balas and Toth, 1985] Balas, E. and Toth, P. (1985). Branch and Bound Methods. In Lawler, E., Lenstra, J., Kan, A. R., and Shmoys, D., editors, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, page 361, Wiley, New York.

[Baluja, 1994] Baluja, S. (1994). Population-Based Incremental Learning: A Method for Integrating Genetic Search based Function Optimization and Competitive Learning. Technical Report CMU-CS-94-163, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

[Beasley, 1990] Beasley, J. E. (1990). OR-Library: Distributing Test Problems by Electronic Mail. *Journal of the Operational Research Society*, 41(11):1069–1072. (See http://mscmga.ms.ic.ac.uk/info.html).

[Bianchi, 2000] Bianchi, L. (2000). Notes on dynamic vehicle routing - the state of the art -. Technical Report IDSIA-05-01, IDSIA, Switzerland.

[Boettcher and Percus, 1998] Boettcher, S. and Percus, A. (1998). Nature's Way of Optimizing.

[Branke et al., 2000] Branke, J., Kaussler, T., Schmidt, C., and Schmeck, H. (2000). A Multi-Population Approach to Dynamic Optimization Problems. In *Proceedings of the 4th International Conference on Adaptive Computing in Design and Manufacturing*.

[Bronstein and Semendjajew, 1991] Bronstein, I. N. and Semendjajew, K. A. (1991). *Taschenbuch der Mathematik*. B. G. Teubner Verlagsgesellschaft and Verlag Nauka, Stuttgart, Leipzig, Moskau, twenty-fifth edition.

[Cavicchio, 1970] Cavicchio, D. J. (1970). *Adaptive Search Using Simulated Evolution*. PhD thesis, University of Michigan, Ann Arbor, MI.

[Chandy et al., 1997] Chandy, J., Kim, S., Ramkumar, B., Parkes, S., and Banerjee, P. (1997). An Evaluation of Parallel Simulated Annealing Strategies with Application to Standard Cell Placement. *IEEE Transactions on Computer-Aided Design*, pages 100–114.

[Cox and Reid, 2000] Cox, D. R. and Reid, N. (2000). *The Theory of the Design of Experiments*. Chapman & Hall/CRC, Boca Raton, London, New York, Washington D. C.

[da Graca Lobo, 2000] da Graca Lobo, F. M. P. (2000). *The Parameterless Genetic Algorithm: Rational and Automated Parameter Selection for Simplified Genetic Algorithm Operation*. PhD thesis, Lisbon University.

[Davidon, 1991] Davidon, W. C. (1991). Variable Metric Method for Minimization. *SIAM Journal Optimization*, 1(1):1–17. With a belated preface for ANL 5990.

[Davis, 1989] Davis, L. (1989). Adapting Operator Probabilities in Genetic Algorithms. In *Proceedings of the third international conference on Genetic algorithms*, pages 61–69. Morgan Kaufmann Publishers Inc.

[De Jong, 1975] De Jong, K. (1975). *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan.

[Dorne and Voudouris, 2004] Dorne, R. and Voudouris, C. (2004). HSF: The iOpt's framework to easily design metaheuristic methods. In Resende, M. G. C. and de Sousa, J. P., editors, *Metaheuristics: Computer Decision-Making*, chapter 11. Kluwer Academic Publishers.

[Duncan, 1995] Duncan, T. (1995). Experiments in the use of Neighbourhood Search techniques for Vehicle Routing. Presented at BCS Expert Systems 95.

[Ehrenberg, 1986] Ehrenberg, A. S. C. (reprinted with corrections 1986). *A Primer in Data Reduction*. John Wiley & Sons.

[Eiben et al., 1999] Eiben, A. E., Hinterding, R., and Michalewicz, Z. (1999). Parameter Control in Evolutionary Algorithms. *IEEE Trans. on Evolutionary Computation*, 3(2):124–141.

[Fisher and Thompson, 1963] Fisher, H. and Thompson, G. L. (1963). Probabilistic learning combinations of local job-shop scheduling rules. In Muth, J. F. and Thompson, G. L., editors, *Industrial Scheduling*, pages 225–251. Prentice Hall, Englewood Cliffs, New Jersey.

[Gendreau et al., 1998] Gendreau, M., Guertin, F., Potvin, J., and Sguin, R. (1998). Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries. Technical Report CRT-98-10, Centre de recherche sur les transports, Universit de Montral.

[Giffler and Thompson, 1960] Giffler, J. and Thompson, G. L. (1960). Algorithms for Solving Production Scheduling Problems. *Operations Research*, 8:487–503.

[Glover, 1989] Glover, F. (1989). Tabu Search, Part 1. *ORSA Journal on Computing*, 1(3):190–206.

[Goldberg, 1989] Goldberg, D. E. (1989). Optimal Initial Population Size for Binary-Coded Genetic Algorithms. Technical report, University of Alabama.

[Goldberg et al., 1992] Goldberg, D. E., Deb, K., and Thierens, D. (1992). Towards a Better Understanding of Mixing in Genetic Algorithms. Technical Report IlliGAL Report No 92009, University of Illinois.

[Gosling, 2003] Gosling, T. (2003). The Simple Supply Chain Model and Evolutionary Computation. In *Proceedings of the Congress on Evolutionary Computation, Canberra, Australia*, pages 2322–2329.

[Grefenstette, 1986] Grefenstette, J. (1986). Optimization of Control Parameters for Genetic Algorithms. *IEEE Trans. Syst. Man Cybern.*, 16(1):122–128.

[Harik et al., 1999] Harik, G., Cantu-Paz, E., Goldberg, D. E., and Miller, B. L. (1999). The Gambler's Ruin Problem, Genetic Algorithms, and the Sizing of Populations. *Evolutionary Computation*, 7:231–255.

[Ho and Pepyne, 2002] Ho, Y. C. and Pepyne, D. L. (2002). *Journal of Optimization Theory and Applications*, 115:549.

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

[Ingber, 1989] Ingber, L. (1989). Very fast simulated re-annealing. *Mathematical Computer Modelling*, 12(8):967–973.

[Irnich, 2000] Irnich, S. (2000). A Multi-Depot Pickup and Delivery Problem with a Single Hub and Heterogeneous Vehicles. *European Journal of Operational Research*, 122(2):310–328.

[Johnson and McGeoch, 1997] Johnson, D. and McGeoch, L. (1997). The Traveling Salesman Problem: A Case Study in Local Optimization. In Aarts, E. H. L. and Lenstra, J., editors, *Local Search in Optimisation*, pages 215–310. John Wiley and Sons, New York.

[Julstrom, 1995] Julstrom, B. A. (1995). What Have You Done for Me Lately? Adapting Operator Probabilities in a Steady-State Genetic Algorithm. In *ICGA*, pages 81–87.

[Kilby et al., 1998] Kilby, P., Prosser, P., and Shaw, P. (1998). Dynamic VRPs: a study of scenarios. Technical Report APES-06-1998, University of Strathclyde, UK.

[Kilby et al., 1999] Kilby, P., Prosser, P., and Shaw, P. (1999). Guided Local Search for the Vehicle Routing Problems With Time Windows. In S.Voss, Martello, S., Osman, I., and C.Roucairol, editors, *Meta-heuristics: Advances and Trends in Local Search for Optimization*, pages 473–486. Kluwer Academic Publishers, Boston.

[Kirkpatrick et al., 1983] Kirkpatrick, S., Gelatt Jr., C. D., and Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science*, 220(4598):671–680.

[Kohonen, 1995] Kohonen, T. (1995). *Self–Organizing Maps*. Springer Verlag, Berlin, Heidelberg.

[Korel, 1990] Korel, B. (1990). Automated software test data generation. *IEEE Transactions on Software Engineering*, 8(16):870–879.

[Li and Lim, 2001] Li, H. and Lim, A. (2001). A Metaheuristic for the Pickup and Delivery Problem with Time Windows. In *ICTAI*.

[Lin et al., 1997] Lin, S.-C., Goodman, E. D., and III, W. F. P. (1997). Investigating Parallel Genetic Algorithms on Job Shop Scheduling Problems. In *Proceedings of the sixth International Conference on Evolutionary Programming*.

[Machado et al., 2002] Machado, P., Tavares, J., Pereira, F. B., and Costa, E. (2002). Vehicle Routing Problem: Doing It The Evolutionary Way. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*.

[Mazumder and Rudnick, 1998] Mazumder, P. and Rudnick, E. (1998). *Genetic Algorithms for VLSI Design, Layout and Test Automation*. Prentice Hall Modern Semiconductor Design Series. Prentice Hall.

[Mercer and Sampson, 1978] Mercer, R. E. and Sampson, J. R. (1978). Adaptive Search Using a Reproductive Meta-Plan. *Kybernetes*, 7:215–228.

[Moldover and Coddington, 1994] Moldover, R. and Coddington, P. (1994). Improved Algorithms for Global Opimization.

[Montgomery, 2005] Montgomery, D. C. (2005). *Design and Analysis of Experiments*. John Wiley & Sons, Inc., New York, sixth edition.

[Muehlenbein, 1992] Muehlenbein, H. (1992). How genetic algorithms really work: Mutation and hill-climbing. *Parallel Problem Solving from Nature*, 2:15–26.

[Mühlenbein and Mahnig, 1999] Mühlenbein, H. and Mahnig, T. (1999). Convergence Theory and Applications of the Factorized Distribution Algorithm. *JCIT: Journal of Computing and Information Technology*, 7.

[Neal, 1993] Neal, R. M. (1993). Probabilistic inference using Markov chain Monte Carlo methods. Technical Report CRG-TR-93-1, Department of Computer Science, University of Toronto.

[Ono et al., 1996] Ono, I., Yamamura, M., and Kobayashi, S. (1996). A Genetic Algorithm for Job-shop Scheduling Problems Using Job-based Order Crossover. In *Proceedings of ICEC'96*, pages 547–552.

[Osman, 1993] Osman, I. H. (1993). Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, 41:421–451.

[Osman, 1995] Osman, I. H. (1995). An Introduction to Meta-Heuristics. In Lawrence, M. and Wilson, C., editors, *Operational Research Tutorial Papers*, pages 92–122, Birmingham. Operational Research Society Press.

[Pelikan et al., 1999] Pelikan, M., Goldberg, D. E., and Cantú-Paz, E. (1999). BOA: The Bayesian Optimization Algorithm. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., and Smith, R. E., editors, *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99*, volume I, pages 525–532, Orlando, FL. Morgan Kaufmann Publishers, San Fransisco, CA.

[Pelikan and Mühlenbein, 1998] Pelikan, M. and Mühlenbein, H. (1998). Marginal Distribution in Evolutionary Algorithms. In *Proceedings of the International Conference on Genetic Algorithms Mendel '98*, pages 90–95, Brno, Czech Republic.

[Pelikan and Mühlenbein, 1999] Pelikan, M. and Mühlenbein, H. (1999). The Bivariate Marginal Distribution Algorithm. In Roy, R., Furuhashi, T., and Chawdhry, P. K., editors, *Advances in Soft Computing - Engineering Design and Manufacturing*, pages 521–535, London. Springer-Verlag.

[Psaraftis, 1995] Psaraftis, H. N. (1995). Dynamic vehicle routing: Status and prospects. *Annals of Operations Research*, 61:143–164.

[Reinelt, 1995] Reinelt, G. (1995). TSPLIB. http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/index.html.

[Royston, 1992] Royston, P. (1992). Approximating the Shapiro-Wilk W test for Non-Normality. *Statistics and Computing*, 2:117–119.

[Russell and Norvig, 2003] Russell, S. and Norvig, P. (2003). *Artificial Intelligence A Modern Approach*. Prentice Hall Series in Artificial Intelligence. Prentice Hall, second edition.

[Sałustowicz and Schmidhuber, 1997] Sałustowicz, R. P. and Schmidhuber, J. (1997). Probabilistic Incremental Program Evolution. *Evolutionary Computation*, 5(2):123–141.

[Schaffer et al., 1989] Schaffer, J. D., Caruana, R., Eshelman, L. J., and Das, R. (1989). A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization. In *Proceedings of the*

3rd International Conference on Genetic Algorithms, pages 51–60, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Sebag and Ducoulombier, 1998] Sebag, M. and Ducoulombier, A. (1998). Extending Population-Based Incremental Learning to Continuous Spaces. In A. E. Eiben, T. Bitck, H.-P. S. and Schoenauer, M., editors, *Fifth International Conference on Parallel Problem Solving from Nature*, New York. Springer-Verlag.

[Servet et al., 1997] Servet, I., Trave-Massuyes, L., and Stern, D. (1997). Telephone network traffic overloading diagnosis and evolutionary computation techniques. In *Third European Conference on Artificial Evolution (AE'97)*, pages 137–144, Berlin. Springer Verlag.

[Shapiro, 2002] Shapiro, J. L. (2002). *The Sensitivity of PBIL to its Learning Rate, and How Detailed Balance Can Remove It.* Morgan Kaufmann.

[Shapiro, 2003] Shapiro, J. L. (2003). Scaling of Probability-Based Optimization Algorithms. *Advances in Neural Information Processing*, 15.

[Smith and Fogarty, 1996] Smith, J. and Fogarty, T. C. (1996). Self Adaptation of Mutation Rates in a Steady State Genetic Algorithm. In *International Conference on Evolutionary Computation*, pages 318–323.

[Smith and Smuda, 1995] Smith, R. E. and Smuda, E. (1995). Adaptively resizing populations: Algorithm, analysis, and first results. *Complex Systems*, 9:47–72.

[Southey and Karray, 1999] Southey, F. and Karray, F. (1999). Approaching Evolutionary Robotics Through Population-Based Incremental Learning. In *Proceedings of the 1999 International Conference on Systems, Man, and Cybernetics (SMC'99).*

[Stevens and D'Agostino, 1986] Stevens, M. and D'Agostino, R., editors (1986). *Goodness of Fit Techniques.* Marcel Dekker, New York.

[Sukthankar, 1997] Sukthankar, R. (1997). *Situation Awareness for Tactical Driving.* PhD thesis, Carnegie Mellon University, Pittsburgh.

[Taillard, 1994] Taillard, E. D. (1994). Parallel taboo search techniques for the job shop scheduling problem. *ORSA Journal on Computing 6*, pages 108–117.

[Tongchim and Chongstitvatana, 2002] Tongchim, S. and Chongstitvatana, P. (2002). Parallel genetic algorithm with parameter adaptation. *Information Processing Letters*, 82(1):47–54.

[Torczon, 1997] Torczon, V. (1997). On the Convergence of Pattern Search Algorithms. *SIAM Journal on Optimization*, 7(1):1–25.

[Toth and Vigo, 2001] Toth, P. and Vigo, D., editors (2001). *The Vehicle Routing Problem*. Siam Monographs on Discrete Mathematics and Applications. SIAM.

[Tucker, 1962] Tucker, H. G. (1962). *An Introduction to Probability and Mathematical Statistics*. Academic Press, New York, London.

[van Laarhoven et al., 1992] van Laarhoven, P. J. M., Aarts, E. H. L., and Lenstra, J. K. (1992). Job shop scheduling by simulated annealing. *Operations Research*, 40(1):113–125.

[Voudouris and Dorne, 2002] Voudouris, C. and Dorne, R. (2002). Integrating Heuristic search and One-Way Constraints in the iOpt Toolkit. In Voss, S. and Woodruff, D., editors, *Operations Research/Computer Science Interfaces (Optimization Software Class Libraries)*, volume 18, chapter 6, pages 177–192. Kluwer Academic Publishers.

[Voudouris and Tsang, 1999] Voudouris, C. and Tsang, E. P. K. (1999). Guided Local Search and its application to the Travelling Salesman Problem. *European Journal of Operational Research*, 113(2):469–499.

[Weinberg, 1970] Weinberg, R. (1970). *Computer Simulation of a Living Cell*. PhD thesis, University of Michigan.

[Whitley and Mathias, 1992] Whitley, D. and Mathias, K. (1992). Genetic Operators, the Fitness Landscape and the Traveling Salesman Problem.

In Manner, R. and Manderick, B., editors, *Parallel Problem Solving from Nature-PPSN 2*, pages 219–228. North Holland-Elsevier.

[Wolpert and Macready, 1995] Wolpert, D. H. and Macready, W. G. (1995). No Free Lunch Theorems for Search. Technical Report SFI-TR-95-02-010, Santa Fe Institute.

[Wolpert and Macready, 1997] Wolpert, D. H. and Macready, W. G. (1997). No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation*, 1:67–82.

[Wu and Hamada, 2000] Wu, C. F. J. and Hamada, M. (2000). *Experiments: Planning, Analysis, and Parameter Design Optimization*. John Wiley & Sons, Inc., New York.

[Yang and Yao, 2003] Yang, S. and Yao, X. (2003). Dual Population-Based Incremental Learning for Problem Optimization in Dynamic Environments. In et. al., M. G., editor, *Proceedings of the 7th Asia Pacific Symposium on Intelligent and Evolutionary Systems*, pages 49–56.

[Zachariasen and Dam, 1995] Zachariasen, M. and Dam, M. (1995). Tabu Search on the Geometric Traveling Salesman Problem. In Osman, I. H. and Kelly, J. P., editors, *Metaheuristics: theory and applications, Proceedings from Metaheuristics International Conference, Colorado*, pages 571–587.

[Zell, 1994] Zell, A. (1994). *Simulation Neuronaler Netze*. Addison–Wesley, Bonn, Paris, Reading Mass., first edition.

[Zhang and Mühlenbein, 2004] Zhang, Q. and Mühlenbein, H. (2004). On the Convergence of a Class of Estimation of Distribution Algorithms. *IEEE Transactions on Evolutionary Computation*, 8(2).