Theory and Applications of Bio-inspired Algorithms

a dissertation presented

by

Markus Wagner

to

The School of Computer Science

in fulfilment of the requirements

for the degree of

Doctor of Philosophy

in the subject of

Computer Science

The University of Adelaide

Adelaide, South Australia

July 2013

Thesis advisors:
Associate Prof. Frank Neumann
Prof. Zbigniew Michalewicz

Markus Wagner

Theory and Applications of Bio-inspired Algorithms

Abstract

Evolutionary algorithms, which form a sub-class of bio-inspired algorithms, mimic some fundamental aspects of the neo-Darwinian evolutionary process. They simultaneously search with a *population* of candidate solutions and associate an objective score as a fitness value for each one. The algorithms then select among the population to favour those solutions that are more fit. The next generation (i.e. a new population) consists of replicates of the fitter solutions that have been *genetically mutated and crossed over* in a biological metaphor: the decision variables were perturbed such that they inherit characters of their *parents*, as well as change in random ways.

For the past decades, the algorithms' success has led to strongly practical-oriented interests. Although the theory of them is far behind the knowledge gained from experiments, there are theoretical investigations about some of their properties. This thesis spans theoretical investigations, theory-motivated algorithm engineering, and also the real-world application of evolutionary algorithms.

First, we analyse different algorithms that work with solutions of variable length. We show theoretically and experimentally that certain design choices can have drastic impacts on the ability of an algorithm to find optimal solutions.

Second, motivated by recent theoretical investigations, we design a framework for solving problems with conflicting objectives. We demonstrate that it can efficiently handle problems with many such objectives, which most existing algorithms have difficulties dealing with.

Finally, we consider the problem of maximising the energy yield of wind farms. Our problem-specific algorithm achieves higher quality results than existing approaches, and it allows for an optimisation within minutes or hours instead of days or weeks.

Contents

Listing of Figures

List of Tables

List of Algorithms

Statement of Originality

To my loving parents, and my beloved family.

Acknowledgments

It was a great honour to work in such lively, inspiring, and outstanding places such as the Algorithms and Complexity Group at the Max Planck Institute for Informatics, and the School of Computer Science at the University of Adelaide.

First of all, I would like to thank Frank Neumann. He was the one to draw my attention to the Max Planck Institute for Informatics. I am most grateful to him for supporting my research on evolutionary algorithms within his research groups, and for inviting me to join him in South Australia. I also would like to thank Benjamin Doerr and Zbyszek Michalewicz for their support of my studies and for asking many thought-provoking questions. Furthermore, I am in debt to the heads of both institutions Kurt Mehlhorn and David Suter for allowing me to be part of their teams.

The best experiences during my research arouse from collaborations with others. Therefore, I am grateful to my co-authors Thomas Ackling, Bernhard Beckert, Bernd Bischl, Cody Boisclair, Thorsten Bormer, Jakob Bossek, Karl Bringmann, Gerd Beuster, Jareth Day, Christopher Denison, Carola Doerr, Tobias Friedrich, Sophia Gao, Niklas Henrich, Daniel Johannsen, Diora Jordan, Timo Kötzing, Per Kristian Lehre, Trent Kroeger, Olaf Mersmann, Samadhi Nallaperuma, Anh Quang Nguyen, Una-May O'Reilly, Tomasz Oliwa, Claudia Schon, Dirk Sudholt, Raymond Tran, Heike Trautmann, Tommaso Urli, Kalyan Veeramachaneni, Katya Vladislavleva, Junhua Wu, and Joseph Yuen. There are many more who I would like to thank, and amongst them are my fellow students in the labs, the professional staff at the institutions, and many conference attendees. Many invaluable friendships were formed.

My work was supported by several scholarships and grants, which I gratefully acknowledge. The Max Planck Research School Postgraduate Scholarship and the School of Computer Science Postgraduate Scholarship allowed me to focus on my studies by covering my living expenses and the numerous conference travels. Further generous travel support includes a Google PhD Top-Up Grant, a Google PhD Travel Prize, a Bupa Postgraduate Travel Grant, and the direct support from the conference organisers of the International Joint Conference on Artificial Intelligence (IJCAI) 2011 and of the Genetic and Evolutionary Computation Conference (GECCO) 2013.

I would also like to thank the University of Adelaide for selecting me as their representative at the China Nine / Group of Eight HDR Forum on “Clean Energy and Sustainable Future 2011” in Beijing, China (related to Part III). The attendance of that event has significantly broadened my personal and professional horizons.

Lastly, I would also like to express my gratitude to several anonymous reviewers and the members of my examination committee. Their comments significantly helped to improve the presentation of the results presented here.

Love is like a friendship caught on fire. In the beginning a flame, very pretty, often hot and fierce, but still only light and flickering. As love grows older, our hearts mature and our love becomes as coals, deep-burning and unquenchable.

Bruce Lee

1 Introduction

In general, the research presented in this thesis is on the so-called *evolutionary algorithms*. These form a sub-class of bio-inspired "problem solving recipes", which mimic some fundamental aspects of the neo-Darwinian evolutionary process. There are several types of bio-inspired algorithms, such as evolutionary algorithms [31], particle swarm optimisation [49], and ant colony optimisation [26], with each type being an individual algorithm paradigm, and having many different concrete instances. Since the 1960's, these recipes are used in computer science to solve complex problems, e.g., route planning, roster scheduling, and design optimisation.

Typically, the algorithms simultaneously search with a *population* of candidate solutions and associate an objective score as a fitness value for each one. They then select among the population to favour those solutions that are more fit. The next generation (i.e. a new population) consists of replicates of the fitter solutions that have been *genetically mutated and crossed over* in a biological metaphor: the decision variables were perturbed such that they inherit some characteristics of their *parents* as well as change in random ways.

For the past decades, the algorithms' success has led to strongly practical-oriented interests [4, 58]. Although the theory of them is far behind the knowledge gained from experiments, there are theoretical investigations about some of their properties.

This thesis consists of the following three parts, which we will outline in more detail in the remainder of this chapter:

- Part I Computational Complexity of Variable-Length Algorithms: we theoretically analyse a typical problem and the impacts that different mechanisms have to solve this problem.
- Part II Design of Evolutionary Multi-Objective Algorithms: motivated by recent theoretical investigations on the approximation indicator, we design a framework for which we theoretically and experimentally show its efficiency.
- Part III Applications to Wind Farm Optimisations: we design efficient solvers for a computationally costly and constrained real-world problem.

Electronic preprints of the papers underlying this thesis are available on the internet at the following URL: `http://www.acrocon.com/~wagner/publications.html`

1.1 Part I: Computational Complexity of Variable-Length Algorithms

One of the challenges of computer science is to get a computer to do what needs to be done, without telling it how this can be accomplished. Genetic Programming (GP) [50], which belongs to the paradigm of evolutionary algorithms, addresses this challenge. It provides a scheme for automatically building computer programs from a high-level statement of the problem. This is achieved by 'breeding' a population of computer programs using the above-mentioned principles of Darwinian natural selection and biologically inspired operations. We refer the interested reader to Poli et al. [73] for a detailed presentation of GP.

In many scenarios the size of a solution is not specified in advance and solutions of larger size may have a larger benefit, especially in the case of classification problems. The flexibility of evolutionary algorithms that work with a variable-length representation often comes at the cost of the so-called bloat problem: individuals grow without providing additional benefit to the quality of solutions [52]. In addition to this growth of the individuals, the additional elements can block the optimisation process so that problems that are relatively easy to optimise can not be handled by variable-length evolutionary algorithms. For these reasons, several methods have been proposed in the past to deal with the bloat problem. Typically, the effect of bloating is investigated by means of empirical evaluation.

In computational complexity analysis, one considers simplified versions of such algorithms and analyses them rigorously on certain classes of problems by treating them as classical randomised algorithms [60]. Taking this point of view, it allows one to use a sophisticated pool of techniques and to treat the algorithms in a strict mathematical sense. In the past, computational complexity analysis has significantly increased the theoretical understanding of evolutionary algorithms for discrete search spaces. Initial results on the computational complexity of evolutionary algorithms have been obtained for artificial pseudo-Boolean functions [27, 75]. These results constitute the foundations for later results on classical combinatorial optimisation, among them some of the most prominent problems in computer science such as minimum spanning trees, shortest paths, and maximum matchings (see [64] for an overview).

Our Contribution

Poli et al. [73] stated in 2008 "we expect to see computational complexity techniques being used to model simpler GP systems, perhaps GP systems based on mutation and stochastic hill-climbing." However, not many studies on such variable-length systems exist to date. With Part I of this thesis, we make significant contributions to their understanding. We start from recent well-founded theoretical results on the behaviour of variable-length evolutionary algorithms [29, 62] and we add to these.

1. In Chapter 3, we theoretically analyse the effect of different bloat-control strategies on the time needed to solve one of the most basic problems in computer science. We prove that certain design choices can have drastic impacts on the ability of an algorithm to find optimal solutions.

2. In Chapter 4, we conduct rigorous experiments as some features of the optimisation process are difficult to analyse theoretically. We study the effects of different initialisation and mutation strategies. Furthermore, our studies allow us to conjecture average-case runtimes.

In our investigations on the bloat problem, we analyse single-objective approaches, as well as multi-objective ones. For several scenarios, we prove that the multi-objective approaches are the more successful ones. However, as we show in the second part of this thesis, not all multi-objective approaches are alike.

1.2 Part II: Design of Evolutionary Multi-Objective Algorithms

Most real-world optimisation problems are characterised by multiple objectives, and these objectives are often in conflict with each other. For example, very cheap cars should also be very safe, and durable aircraft components should also be light. The general goal of solving a multi-objective optimisation (MOO) problem is to find a set of compromise solutions. This set of solutions can then be inspected, and a decision maker can choose one or potentially several solutions to a problem.

Due to the hardness of almost all interesting multi-objective problems, different heuristic approaches have been used to tackle them. Among these methods evolutionary algorithms are frequently used as they work at each time step with a set of solutions called population. The population of an evolutionary algorithm for a MOO problem is used to store desired trade-off solutions for the given problem.

The Pareto front of a MOO problem consists of the function values representing the different trade-offs with respect to the given objective functions. In practice, it is impossible to compute the whole Pareto front, and MOO problems can often only be solved approximately by heuristic approaches. As the size of the Pareto front is often very large, evolutionary algorithms and all other algorithms for MOO have to restrict themselves to a smaller set of solutions. Eventually, this set of solutions should be a good approximation of the Pareto front. The main question is now how to define approximation. The literature (see, e. g., [19]) on evolutionary multi-objective optimisation just states that the set of compromise solutions

- should be close to the true Pareto front,
- should cover the complete Pareto front, and
- should be uniformly distributed.

However, this notion of approximation is not a formal definition. Having no formal definition of approximation makes it hard to evaluate and compare algorithms for MOO problems. Therefore, we think that it is necessary to use a formal definition of approximation in this context and to evaluate algorithms with respect to this definition.

Our Contribution

The algorithms in the literature use different measures to ensure diversity in the objective space, but they are not guided by a formal notion of approximation. There are a few exceptions that we will point out later, but they are only of theoretical interest.

This research is motivated by many theoretical investigations on the beneficial use of formal definitions of approximation in the context of solving MOO problems. In Part II of this thesis, we demonstrate a framework for approximation-guided evolutionary algorithms. It is for the efficient optimisation of problems with many objectives.

1. In Chapter 6, we design a framework that uses a formal notion of approximation for solving problems with conflicting objectives. We theoretically analyse its computational complexity and we experimentally show its advantages over several existing MOO approaches.

2. In Chapter 7, we present two major improvements that (1) drastically increase the framework's solution quality, and (2) decrease its computational costs at the same time. We demonstrate that the framework can efficiently handle problems

with up to twenty conflicting objectives, which most existing algorithms have difficulties dealing with.

Our framework enables practitioners now to add objectives with only minor consequences, and to explore problems for even higher dimensions. While many real-world problems have multiple conflicting objectives, some have just a single objective, but the solutions need to satisfy certain constraints. Then, if the evaluation is computationally costly, even a single-objective optimisation problem, such as the one presented in the Part III of this thesis, can be difficult to optimise within reasonable time.

1.3 Part III: Applications to Wind Farm Optimisations

Renewable energy is energy that comes from natural resources, such as sunlight, wind, tides, and geothermal heat. With renewable energy being a 'hot topic' right now and the renewable energy market rapidly expanding worldwide, the rapid growth of the renewable energy industry has led to cost reduction challenges.

The *wind farm layout problem* entails the process of planning the placement of turbines on a potential wind farm site, and the layout design of a wind farm is an important component of ensuring the profitability of a wind farm project [95]. There, an inadequate design would lead to lower than expected wind power capture, increased maintenance costs, etc. The creation of a farm layout involves the invocation of a software optimisation module, which attempts to efficiently place the turbines while adhering to the constraints and optimising the stated objectives. Often, this module is embedded within a specialised tool provided by wind power consultants such as Garrad Hassan or AWS TruePower, who offer a product such as the publicly available OpenWind [3]. One of the problems that such tools have to deal with, apart from the actual optimisation, is the scaling cost of computing wake effects when estimating energy capture for increasing numbers of turbines. To estimate the energy capture of a layout the optimisation module models the free stream wind flowing through the site in and out of the turbines. Some degree of non-linear wind turbulence occurs at the outflow of a turbine and affects the inflow to turbines close enough behind it. Modelling this effect is necessary because wake has a great effect on the actual energy output of a wind farm. However, computing the wake effects with respect to a given wake model such as the Park wake model [66] is computationally expensive. This computational effort is significant if one is applying iterative search algorithms such as local search, simulated annealing or evolutionary algorithms for the optimisation of the placement. Such methods need to evaluate each farm layout with respect to the wake model under consideration and would therefore require costly simulations for each solution that is considered during the optimisation process.

With increasing frequency, wind farms are getting larger. For example, the Horse Hollow Wind Energy Center in Texas, USA operates with 735.5 megawatt (MW) capacity and consists of more than 300 turbines spread over nearly 47,000 acres (190 km^2). The layout of turbines in such large wind farms is challenged by large numbers of turbines, large farm areas, constraints on feasible sitings and expensive wake models because the number of siting combinations of turbines on a large area is huge, constraints must be respected and the cost of calculating wake loss scales non-linearly with each additional turbine.

Our Contribution

In order to deal with these complex challenges bio-inspired algorithms have been applied several times. However, the results achieved so far are not very satisfying. For example, most of the underlying models make inaccurate assumptions in order to make the optimisation process computationally feasible, and there is often room left for improvements of the results as well.

In Part III of this thesis, we demonstrate accurate and efficient optimisation algorithms for placing hundreds of turbines. Such an algorithm can be incorporated as an software optimisation module in the above-mentioned tool OpenWind.

1. In Chapter 9, we present our first approach to optimise the placements of up to one thousand turbines. We analyse the benefits of increasing the number of wind turbines, as well as the benefits of increasing the available area in order to reduce the disadvantageous wake effects.

2. In Chapter 10, we present a fast approach for evaluating new layouts. In addition, we demonstrate a self-adaptive variation operator that makes effective use of the problem characteristics to produce new layouts. We also show that our approach significantly outperforms the industry tool OpenWind.

The single-objective yield optimisation of wind turbine placements on a given area of land is a challenging optimisation problem. However, with our results in mind, we can consider this problem solved.

Part I

Computational Complexity of Variable-Length Algorithms

The further a mathematical theory is developed, the more harmoniously and uniformly does its construction proceed, and unsuspected relations are disclosed between hitherto separated branches of the science.

David Hilbert

2 Algorithms and Methods for Their Analysis

Evolutionary algorithms have found many applications in different domains, such as combinatorial optimisation, multi-objective optimisation, finance and renewable energy (see, e.g., [16, 17, 73]). In many scenarios the size of a solution is not specified in advance and solutions of larger size may have a larger benefit especially in the case of classification problems.

The flexibility of evolutionary algorithms that work with a variable-length representation often comes at the cost of the so-called bloat problem: individuals grow without providing additional benefit to the quality of solutions [52]. In addition to this growth of the individuals, the additional elements can block the optimisation process so that problems that are relatively easy to optimise can not be handled by variable-length evolutionary algorithms. For these reasons, several methods have been proposed in the past to deal with the bloat problem. In this part of the thesis, we study the behaviour of variable-length evolutionary algorithms and the influence of different bloat-control mechanisms.

2.1 Motivation

Genetic programming (GP) [50] is the most prominent example of a variable-length evolutionary algorithm, as it often evolves tree-like solutions for a given problem. Just recently, the first computational complexity results on this type of algorithm have been obtained, following the line of successful research on evolutionary algorithms with fixed-length representation (see the books by Auger and Doerr [2], Neumann and Witt [64] for an overview). In general, variable-length representations increase the search space significantly, and it is desirable to better understand the behaviour of algorithms using such representations from a theoretical point of view. For example, Cathabard et al. [14] investigated non-uniform mutation rates for problems with unknown solution lengths. They used a simple evolutionary algorithm to find a bitstring with an unknown number of leading ones and although the bitstring had some predetermined maximum length, only an unknown number of initial bits was used by the fitness function. Durrett et al. [29] investigated worst-case and average-case runtimes of a simple tree-based genetic programming algorithm. The tackled problems were separable, with independent and additive fitness structures.

Many evolutionary algorithms that work with a variable-length representation do not work (in their most basic variant) with a form of bloat-control. One prominent way of dealing with the bloat problem is the parsimony approach: in the case that two solutions are of equal quality the solution of lower complexity is preferred. Another way of coping with the bloat problem, which is sometimes even implemented in industrially used GP packages such as Datamodeller [33], is to use a multi-objective approach that uses a population representing the different trade-offs according to the original goal function and the complexity of a solution. The solutions that represent the trade-offs are called Pareto optimal. Both approaches of coping with the bloat problem have recently been examined for the problems ORDER and MAJORITY in the context of genetic programming [62, 68, 84].

Neumann [62] shows that both approaches help to solve the problems ORDER and MAJORITY, but the differences between these two approaches are not examined. In the next chapters we point out that switching from the parsimony approach to the multi-objective one can significantly reduce the runtime. In particular, we will present examples that prove to be local optima for the parsimony approach, whereas the multi-objective approach is able to compute the optimal solution provably within a polynomial number of steps.

This first part of the thesis is organised as follows. In this particular chapter, we first introduce different problems in Section 2.2. Afterwards, we present the variable-length algorithms that we will analyse in Section 2.3. Lastly, in Section 2.4 we show some methods that are frequently used in the computational complexity analysis of evolutionary algorithms. Then, in Chapter 3 we investigate the ability of different variable-length algorithms to solve one of the fundamental problems in computer science, namely the problem of sorting elements. In Chapter 4 we complement recent theoretical investigations by rigorous experiments, in order to analyse important features of the optimisation process.

2.2 Problems

Our goal is to study the differences between bloat-control mechanisms for variable-length evolutionary algorithms. In our theoretical and experimental investigations, we will treat the algorithms and problems analysed in [29, 62, 67]. We consider tree-based genetic programming, where a possible solution is represented by a syntax tree. The inner nodes of such a tree are labelled by function symbols from a set F and the leaves of the tree are labelled by terminals from a set T.

Even though many GP algorithms allow complex functions for the inner nodes, we restrict the set of functions to the single binary function “join” J. Effectively, we use J's to achieve variable-length lists by concatenating leaf nodes.

2.2.1 Weighted ORDER and Weighted MAJORITY

In Chapter 4, we examine the problems Weighted ORDER (WORDER) and Weighted MAJORITY (WMAJORITY). The tackled problems are to some extent easy, as they are separable, with independent and additive fitness structures. Furthermore, they all have multiple solutions, which can be regarded as a key property of a real GP problem.

In these problems, the only function symbol is the above-mentioned binary “join”. The terminal set consists of $2n$ variables, where $\bar{x}_i$ is considered the complement of x_i. Hence, $F := \{J\}$, and $L := \{x_1, \bar{x}_1, x_2, \bar{x}_2, ..., x_n, \bar{x}_n\}$.

The ORDER problem represents problems, where the set $F \cup L$ includes conditional functions. For example, numerical comparison functions (e.g, $\leq, <, >$) are often used in classification problems. Such functions have two arguments (two subtrees) and only one of the subtrees will be executed depending on the outcome of the conditional function. Thus, a conditional function results in a conditional execution path. The algorithm's task is now to correctly position the functions to achieve the correct conditional execution behaviour for all test data. Note that ORDER is an abstraction from this, as trees are inspected instead of executed. Then, if a terminal is encountered before its complement in an inorder parse, then the conditional execution path is considered to be correct. Solutions to ORDER reflect a property that is commonly found in GP solutions, where conditional functions are used: multiple equivalent solutions exist, with different conditional execution paths.

The MAJORITY problem reflects a generally desired property of solutions: they should have only correct functionality, and no incorrect functionality. Again, MAJORITY is an abstraction. A correct solution needs to have at least as many occurrences of a terminal than of its corresponding complement, and it must furthermore contain all (non-complement) terminals at least once.

In the generalisations WORDER and WMAJORITY, each variable x_i is assigned a weight $w_i \in \mathbb{R}$, $1 \leq i \leq n$ so that the variables can differ in their contributions to the

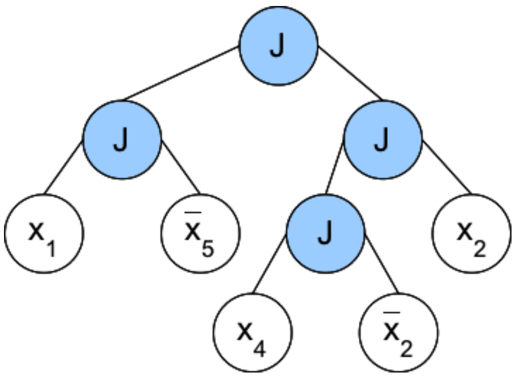


Figure 2.1: Example for evaluations according to WORDER and WMAJORITY. Let $n = 5$ and $w_1 = 15$, $w_2 = 14$, $w_3 = 12$, $w_4 = 7$, and $w_5 = 2$. For the shown tree X, we get (after inorder parsing) $l = (x_1, \bar{x_5}, x_4, \bar{x_2}, x_2)$. For WORDER, we get $S = (x_1, x_4)$ and WORDER$(X) = w_1 + w_4 = 22$. For WMAJORITY, we get $S = (x_1, x_4, x_2)$ and WMAJORITY$(X) = w_1 + w_4 + w_2 = 36$.

fitness of a tree. Without loss of generality, we assume that $w_1 \geq w_2 \geq w_3 \geq \ldots \geq w_n > 0$. We get the ORDER and MAJORITY as specific cases of WORDER and WMAJORITY where $w_i = 1$, $1 \leq i \leq n$.

For a given solution X, the fitness value is computed by parsing the represented tree inorder. For WORDER, the weight w_i of a variable x_i contributes to the fitness of X iff x_i is visited in the inorder parse before all the $\bar{x}_i$ in the tree. For WMAJORITY, the weight of x_i contributes to the fitness of X iff the number of occurrences of x_i in the tree is at least one and not less than the number of occurrence of $\bar{x}_i$ (see Algorithms 2.1 and 2.2). We call a variable redundant if it occurs multiple times in the tree; in this case the variable contributes only once to the fitness value. The goal of WORDER and WMAJORITY problems is to maximise their function values. We illustrate both problems by an example (see Figure 2.1).

Algorithm 2.1: WORDER(X)

```
  input: a syntax tree $X$
  init  : an empty leaf list $l$, an empty statement list $S$
1 Parse $X$ in-order and insert each leaf the rear of $l$ as it is visited;
2 Generate $S$ by parsing $l$ front to rear and adding a leaf to $S$ only if its
  complement is not yet in $S$;
3 WORDER $(X) = \sum_{x_i \in S} w_i$;
```

MO-WORDER and MO-MAJORITY are variants of the above-described problems, which take the complexity C of a syntax tree (computed by the number of leaves of the tree) as the second objective:

- MO-WORDER (X) = (WORDER (X), C(X))

Algorithm 2.2: WMAJORITY(X)

```
input: a syntax tree $X$
init  : an empty leaf list $l$, an empty statement list $S$
1 Parse $X$ in-order and insert each leaf the the rear of $l$ as is is visited;
2 For $1 \leq i \leq n$: if count$(x_i \in l) \geq$ count$(\overline{x}_i \in l)$ and count$(x_i \in l) \geq 1$, then add $x_i$
  to $S$;
3 WMAJORITY $(X) = \sum_{x_i \in S} w_i$;
```

- MO-WMAJORITY (X) = (WMAJORITY (X), C(X))

Optimisation algorithms can then use this to cope with the bloat problem: if two solutions have the same fitness value, then the solution of lower complexity can be preferred. In the special case, where $w_i = 1$ holds for all $1 \leq i \leq n$, we have the problems:

- MO-ORDER (X) = (ORDER (X), C(X))
- MO-MAJORITY (X) = (MAJORITY (X), C(X))

2.2.2 Sortedness Measures

The problem that we use as the basis for our investigations in Chapter 3 is a classical problem from the computational complexity analysis of evolutionary algorithms with fixed-length representations, namely the sorting problem (SORTING). Scharnow et al. [76] considered SORTING as an optimisation problem, where different fitness functions measure the sortedness of a permutation of elements. It was discovered that different fitness functions lead to problems of different difficulties.

It is important to note that, in contrast to WORDER and WMAJORITY, the SORTING problem cannot be split into subproblems that can be solved independently. As we shall see, these dependencies have a significant impact on the time needed to solve the problem.

We will analyse our algorithms on different measures of sortedness. The problem SORTING can be stated as follows. Given a totally ordered set (of terminals) $T = \{1, \ldots, n\}$ of n elements, the task is to find a permutation π_{opt} of the elements of T such that

$$\pi_{opt}(1) < \pi_{opt}(2) < \ldots < \pi_{opt}(n)$$

holds, where $<$ is the order on T. Without loss of generality, we assume $\pi_{opt} = id$, i. e. $\pi_{opt}(i) = i$ for all i, throughout our analyses.

The set of all permutations π forms a search space that has already been investigated by Scharnow et al. [76] for the analysis of permutation-based evolutionary algorithms.

The authors of that article investigate SORTING as an optimisation problem where the goal is to maximise the sortedness of a given permutation. We will consider the following fitness functions measuring the sortedness of a given permutation introduced in [76]:

- INV(π), measuring the number of pairs in correct order (larger values are better),
- HAM(π), measuring the number of elements at correct position, which is the number of indices i such that $\pi(i) = i$ (larger values are better),
- RUN(π), measuring the number of maximal sorted blocks, which is the number of indices i such that $\pi(i+1) < \pi(i)$ plus one (smaller values are better),
- LAS(π), measuring the length of the longest ascending subsequence (larger values are better),
- EXC(π), measuring the minimal number of pairwise exchanges in π, in order to sort the sequence (smaller values are better).

Given a tree X, we determine the permutation π that it represents according to Algorithm 2.3. Once we have seen an element during an inorder parse, we skip its duplicates. This is necessary, as the resulting sequence of elements for which we determine its sortedness should contain each element at most once.

Algorithm 2.3: Derivation of $F(X)$ for SORTING

```
1 Generate $\pi$ by parsing $X$ front to rear and adding an element to $\pi$ only if it is not
  yet in $\pi$;
2 Return $F(\pi)$;
```

Note that $EXC(\pi)$ can be computed in linear time, based on the cycle structure of permutations. If the sequence is sorted, it has n cycles. Otherwise, it is always possible to increase the number of cycles by exchanging an element that is not sitting at its correct position with the element that is currently sitting there. For any given permutation π consisting of $n-k$ cycles, $EXC(\pi) = k$.

We will investigate the different measures for variable-length evolutionary algorithms. Consequently, we might have to deal with incomplete permutations as not all elements have to be contained in a given individual. Most measures can also be used for incomplete permutation, but we have to make sure that complete permutations always obtain a better fitness than incomplete ones, so that the sortedness measure guides the algorithm from incomplete permutations to complete ones.

We will use the sortedness measures as above and use the following special fitness assignments that enforce these properties:

- INV(π) is the number of pairs in order, except INV(π) $= 0$ if $|\pi| = 0$, and INV(π) $= 0.5$ if $|\pi| = 1$,
- RUN(π) $= n+1$ if $|\pi| = 0$, otherwise RUN(π) $= b+m$ is the sum of the number of maximal sorted blocks b, and the number of elements missing $m = n - |\pi|$,
- If $|\pi| \leq n$ then EXC(π) $= e + m + 1$, otherwise EXC(π) $= e$, where e is the number of necessary exchanges, and $m = n - |\pi|$ the number of missing elements.

Note that e can be computed for incomplete permutations as well, as only the order $<$ on the expressed variables has to be respected. This means that the permutations $\pi_1 = (1, 4)$ and $\pi_2 = (1, 2, 3, 4)$ require no changes, but EXC(π_1) $\neq$ EXC(π_2), as the number of missing elements differs.

For example, for a tree X with $\pi = (2, 3, 4, 5, 1, 6)$ and $n = 7$, the sortedness results are HAM(X) $= 1$, RUN(X) $= 1 + 1 = 2$, and EXC(X) $= 4 + 1 + 1 = 6$.

Lastly, we can again create multi-objective variants of the above-described problems, which take the complexity C of a syntax tree (again computed by the number of leaves of the tree) as the second objective:

- MO-INV (X) = (INV (X), C(X))
- MO-HAM (X) = (HAM (X), C(X))
- MO-EXC (X) = (EXC (X), C(X))
- MO-RUN (X) = (RUN (X), C(X))
- MO-LAS (X) = (LAS (X), C(X))

2.3 Algorithms

All GP algorithms that are analysed in this thesis only use the mutation operator HVL-Prime to generate offspring. HVL-Prime is an update of O'Reilly's HVL mutation operator [70, 71]. It is motivated by minimality, rather than by problem-specific operations. HVL-Prime produces a new tree by making changes to the original tree via three basic operators: insertion, deletion and substitution (see Algorithm 2.4). In each step of the algorithms, k mutations are applied to the selected solution. For the single-operation variants of the algorithms, $k = 1$ holds. For the multi-operation variants, the number of operations performed is drawn each time from the distribution $k = 1 + Pois(1)$, where $Pois(1)$ is the Poisson distribution with parameter1.

The algorithm (1+1)-GP* that we investigate first has no explicit mechanism to control bloat whatsoever. The only feature that can potentially prevent the solution's

Algorithm 2.4: HVL-Prime mutation operator

```
1 Mutate $Y$ by applying HVL-Prime $k$ times: each time randomly choose either
  insert, substitute or delete.
2 if Insert then
3     Choose a variable $u \in L$ uniformly at random and select a node $v \in Y$
      uniformly at random. Replace $v$ by a join node whose children are $u$ and $v$, in
      which their orders are chosen randomly.
4 if Substitute then
5     Replace a randomly chosen leaf $v \in Y$ by a randomly chosen leaf $u \in L$.
6 if Delete then
7     Choose a leaf node $v \in Y$ randomly with parent $p$ and sibling $u$. Replace $p$ by
      $u$ and delete $p$ and $u$.
```

Algorithm 2.5: (1+1)-GP*-single for maximisation

```
1 Choose an initial solution $X$;
2 repeat
3     Set $Y := X$;
4     Apply the mutation operator (given in Figure 2.4) with $k = 1$ to Y;
5     if $f(Y) > f(X)$ then set $X := Y$;
```

size to become too large is that only strict fitness improvements are accepted. Thus, the maximum solution size is limited based on the size of the initial solution and by the number of possible fitness improvements that can be performed.[1]

The single-objective variant called (1+1)-GP*-single (see Algorithm 2.5) starts with an initial solution X, and produces in each iteration a single offspring Y by applying the mutation operator HVL-Prime given in Algorithm 2.4 with $k = 1$. This means that it is a stochastic hill-climber that explores its local neighbourhood. In the case of maximisation, Y replaces X if $f(Y) > f(X)$ holds. Minimisation problems are tackled in the analogous way.

The single-objective variant called (1+1)-GP (see Algorithm 2.6 for the single-mutation variant) is identical to the just described (1+1)-GP*, with the exception that, in the case of maximisation, Y replaces X if $f(Y) \geq f(X)$ holds. Again, minimisation problems are tackled in the analogous way. As a consequence of the relaxed acceptance condition, the complexity of the solution can increase as long as the fit-

[1]Note that the naming of our GP variants follows the conventions often used in the computational complexity analysis of evolutionary algorithms: an asterisk indicates that a strict fitness improvement over the old solution is required in order for the new solution to replace the current solution.

Algorithm 2.6: (1+1)-GP-single for maximisation

```
1 Choose an initial solution $X$;
2 repeat
3     Set $Y := X$;
4     Apply the mutation operator (given in Figure 2.4) with $k = 1$ to Y;
5     if $f(Y) \geq f(X)$ then set $X := Y$;
```

ness does not decrease. Thus, (1+1)-GP truly has no mechanism to prevent bloat whatsoever.

In order to introduce the parsimony pressure to (1+1)-GP, where in case of identical fitnesses the solution of lower complexity is preferred, we employ the multi-objective variants of the presented sortedness measures, i.e. MO-WMAJORITY, MO-EXC, etc. Without loss of generality, we assume that C is to be minimised and all fitness functions F, except RUN and EXC, are maximised.[2] In the parsimony approach, we optimise the above-defined multi-criteria fitness functions MO-F$(X) = (F(X), C(X))$ with respect to the lexicographic order, that is, MO-F$(X) \geq$ MO-F(Y) holds iff

$$F(X) > F(Y) \vee (F(X) = F(Y) \wedge C(X) \leq C(Y)) \,. \tag{2.1}$$

As the last algorithm, we consider the Simple Evolutionary Multi-Objective Genetic Programming (SMO-GP) algorithm introduced by Neumann [62] and motivated by the SEMO algorithm for fixed length representations by Laumanns et al. [55]. Variants of SEMO have been frequently used in the runtime analysis of evolutionary multi-objective optimisation for fixed length representations [see 34–36, 63, 64].

In this multi-objective variable-length algorithm, we treat the two criteria F and C as equally important. In order to compare two solutions, we consider the classical Pareto dominance relations:

1. A solution X *weakly dominates* a solution Y (denoted by $X \succeq Y$) iff $(F(X) \geq F(Y) \wedge C(X) \leq C(Y))$.
2. A solution X *dominates* a solution Y (denoted by $X \succ Y$) iff $((X \succeq Y) \wedge (F(X) > F(Y) \vee C(X) < C(Y))$.
3. Two solution X and Y are called *incomparable* iff neither $X \succeq Y$ nor $Y \succeq X$ holds.

A *Pareto optimal solution* is a solution that is not dominated by any other solution in the search space. All Pareto optimal solutions together form the Pareto optimal set, and the set of corresponding objective vectors forms the Pareto front. The classical

[2]The notions can be easily adjusted to other minimisation/maximisation problems.

Algorithm 2.7: SMO-GP

```
1 Choose an initial solution $X$;
2 Set $P := \{X\}$;
3 repeat
4     Choose $X \in P$ uniformly at random;
5     Set $Y := X$;
6     Apply mutation to Y;
7     if $\{Z \in P \mid Z \succeq Y\} = \emptyset$ then set $P := (P \setminus \{Z \in P \mid Z \succ Y\}) \cup \{Y\}$;
```

goal in multi-objective optimisation is to compute for each objective vector of the Pareto front a Pareto optimal solution. Alternatively, if the Pareto front is too large, the goal then is to find a representative subset of the front, where the definition of 'representative' depends on the choice of the conductor.

SMO-GP (see Algorithm 2.7) is a population-based approach that starts with a single solution and it maintains a set of non-dominated solutions obtained during the optimisation run. This set of solutions constantly approximates the true Pareto front, i.e. the set of optimal trade-offs between fitness and complexity. In each iteration, it picks one solution uniformly at random and produces one offspring Y by mutation. Y is introduced into the population iff it is not weakly dominated by any other solution in P. If Y is added to the population all individuals that are dominated by Y are discarded.

Similar to the previously introduced algorithms, SMO-GP-single uses the mutation operator HVL-Prime with k=1. We also consider SMO-GP-multi that differs from SMO-GP-single by choosing k according to $1 + Pois(1)$.

2.4 Methods for the Analysis

In the following, we will present several methods that are used frequently in the computational complexity analysis of evolutionary algorithms. The computational complexity analysis of genetic programming analyses the expected number of fitness evaluations until an algorithm has produced an optimal solution for the first time. This is called the *expected optimisation time.* In the case of multi-objective optimisation the number of fitness evaluations until the whole Pareto front has been computed is analysed and referred to as the expected optimisation time.

Note that we do not present proofs of these methods here, as these are considered standard tools. For an in-depth introduction with many examples and proofs, we refer the interested reader to the books by Auger and Doerr [2] and Neumann and Witt [64].

2.4.1 Fitness-Level Method

The first method is the so-called *fitness-level method*, to estimate the expected optimisation time. We will use this method on many occasions in Chapter 3. It has originally been introduced for the analysis of elitist evolutionary algorithms (see, e. g., [94]), where the fitness of the current search point can never decrease. The idea is to partition the search space into levels $A_1, \ldots, A_m$ that are ordered with respect to fitness values. Formally, we require that for all $1 \leq i \leq m-1$ all search points in A_i have a strictly lower fitness than all search points in A_{i+1}. In addition, A_m must contain all global optima. Now, if s_i is (a lower bound on) the probability of discovering a new search point in $A_{i+1} \cup \cdots \cup A_m$, given that the current best solution is in A_i, the expected optimisation time is upper bounded by $\sum_{i=1}^{m-1} 1/s_i$, as $1/s_i$ is (an upper bound on) the expected time until fitness level i is left and each fitness-level has to be left at most once.

2.4.2 Drift Analysis

Drift analysis was introduced to the theory of evolutionary algorithms by He and Yao [45]. Since then, it has become one of the strongest (albeit deep mathematical) tools for proving run-time guarantees for many evolutionary algorithm (e.g., see [37, 44, 69]). In drift analysis, one uses an auxiliary potential function and tracks its behaviour. This is in contrast to the fitness-level method, where one tracks how the objective function value improves. It is required that the progress depends on the current potential value, and for a number of problems such potential functions are a natural choice (e.g., minimum spanning tree problem, single-source shortest path problem).

In the proof of Theorem 7, we will use drift analysis with tail bounds [23, 24]. When a feasible drift-function according to Definition 1 exists, then this allows for an elegant computational complexity analysis via Theorem 1. In the following, we list the original definition and theorem for the reader's convenience.

Let $n \in \mathbb{N}$ be the problem size. Let Ω_n be a search space and $f_n : \Omega_n \to \mathbb{R}$ be an objective function defined on Ω_n. Furthermore, let $\Omega_{opt} \subseteq \Omega_n$ be the set of optimal search points.

Definition 1 (Definition 1 in [23])**.** *Let $\nu : \mathbb{N} \to \mathbb{R}$ be monotonically increasing. We call $\Phi : \Omega_n \to \mathbb{R}$ a feasible ν-drift function for f_n and a given (1+1) evolutionary algorithm, if the following conditions are satisfied.*

1. $\Phi(x) = 0$ *for all* $x \in \Omega_{opt}$,

2. $\Phi(x) \geq 1$ *for all* $x \in \Omega_n \backslash \Omega_{opt}$,

3. *there is a constant $\delta > 0$ (independent of n) such that for all $x \in \Omega_n \backslash \Omega_{opt}$*

$$E(\Phi(x_{new})) \leq \left(1 - \frac{\delta}{\nu(n)}\right) \Phi(x)$$

where as above we denote by x_{new} the solution resulting from executing a single iteration (consisting of mutation and selection) with initial solution x.

Theorem 1 (Theorem 1 in [23])**.** *Let $\Phi : \Omega_n \to \mathbb{R}$. Denote by $\Phi_{max} := max\{\Phi(x) \,|\, x \in \Omega_n\}$ the maximum value of Φ. If Φ is a feasible ν-drift function (with implicit constant δ) for f_n and a given (1+1)-EA, then the expected optimisation time is at most*

$$\frac{\nu(n)}{\delta}(1 + \ln \Phi_{max}).$$

Also, for any $c > 0$ (possibly depending on n), we have that the optimisation time exceeds $\frac{\nu(n)}{\delta}(\ln \Phi_{max} + c \ln n)$ with probability at most n^{-c}.

2.4.3 Chernoff Inequalities

In probability theory, these bounds give exponentially decreasing bounds on tail distributions of independent random variables. This is in contrast to Markov's inequality of Chebyshev's inequality, which both yield only power-law bounds on tail distributions.

In the proof of Theorem 7, we will apply Chernoff's inequalities. There exist many versions and extensions of Chernoff's inequalities, but we will restrict ourselves to the following.

Definition 2. *Let random variables $X_1, \ldots, X_n$ be independent random variables taking on values 0 or 1. Further, assume that $P(X_i = 1) = p_i$. Then, if we let $X = \sum_{i=1}^{n} X_i$ and $E[X]$ be the expectation of X, then the following bounds hold:*

$$\begin{array}{ll} P(X > (1+\delta)E[X]) < \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{E[X]} & \delta > 0 \\ P(X \geq (1+\delta)E[X]) \leq e^{-E[X]\delta^2/3} & 0 < \delta \leq 1 \\ P(X \leq (1-\delta)E[X]) \leq e^{-E[X]\delta^2/2} & 0 < \delta \leq 1 \end{array}$$

If you want to know the taste of a pear, you must change the pear by eating it yourself. If you want to know the theory and methods of revolution, you must take part in revolution. All genuine knowledge originates in direct experience.

Mao Zedong

3 Theoretical Analysis of SORTING

Sorting is one of the most basic problems in computer science. It is also the first combinatorial optimisation problem for which computational complexity results have been obtained in the area of discrete evolutionary algorithms [25, 76]. In [76], sorting is treated as an optimisation problem where the task is to minimise the unsortness of a given permutation of the input elements. To measure unsortness, different fitness functions have been introduced in the past and studied with respect to the difficulty of being optimised by permutation-based evolutionary algorithms.

The problems ORDER and MAJORITY, which we are going to analyse in Chapter 4, are in a sense easy, as they have isolated problem semantics, and thus allow one to treat subproblems independently. The next step then is to consider problems that have dependent problem semantics, and we will do this in the following based on the SORTING problem. In particular, we are interested in the influence of different bloat-control techniques on the runtime of different algorithms. As we shall see, the choice of the bloat-control technique can significantly influence the time needed to solve the problem.

3.1 Standard Approach Without Bloat-Control

The algorithm (1+1)-GP* (see Algorithm 2.5) that we investigate first has no mechanism to control bloat whatsoever. The only feature that can potentially prevent the solution's size to become too large is that only strict fitness improvements are accepted.

Thus, the maximum solution size is limited based on the size of the initial solution and by the number of possible fitness improvements that can be performed.

Let us recall that the single-objective variant called (1+1)-GP*-single starts with an initial solution X, and produces in each iteration a single offspring Y by applying the mutation operator given in Figure 2.4 with $k = 1$. This means that it is a stochastic hill-climber that explores its local neighbourhood. In the case of maximisation, Y replaces X if $f(Y) > f(X)$ holds. Minimisation problems are tackled in the analogous way.

3.1.1 Upper Bound

In this section we analyse the performance of our (1+1)-GP* variants on one of the fitness functions introduced in Section 2.3.

We exploit a similarity between our variants and evolutionary algorithms to obtain an upper bound on the time needed to find an optimal solution. We use the method of *fitness-based partitions* (see Section 2.4). Although the used HVL-Prime operator is complex, we can obtain a lower bound on the probability of making an improvement considering fitness improvements that arise from the HVL-Prime sub-operations insertion and substitution. In combination with fitness levels defined individually for the used sortedness measures, this gives us the runtime bounds in this section.

We denote by T_{max} the maximal size of the tree during the run of the algorithm and show the following theorem.

Theorem 2. *The expected optimisation time is $O(n^3 T_{max})$ for the (1+1)-GP*-single and (1+1)-GP*-multi, using the sortedness measure INV.*

Proof. The proof is an application of the fitness-based partitions method. Based on the observation that $n \cdot (n-1)/2 + 1$ different fitness values are possible, we define the fitness levels $A_0, \ldots, A_{n\cdot(n-1)/2}$ with

$$A_i = \{\pi \,| INV(\pi) = i\}\,.$$

As there are at most $n \cdot (n-1)/2$ advancing steps between fitness levels to be made, the total expected runtime is upper bounded by the sum over all times needed to make such steps.

We bound the times by investigating the case when only a particular insertion of a specific leaf at its correct position achieves an increase of the fitness.[1] For this particular insertion, we consider the lexicographically smallest pair (i, j), $i < j$, that is currently incorrect: putting i directly before j makes this pair correct. We now

[1]For example, the tree with the sequence of leaves (when parsed inorder) $l = (n, n, 1, 2, \ldots, n-1)$ can only be improved (in a single HVL-Prime step) by inserting a leaf labelled 1 at the leftmost position.

have to show that this does not make any other pair which was previously correct, incorrect. Assume there is a pair (k,l), $k < l$ that was previously correct and has become incorrect due to the insertion of i. As only i is moved, $l = i$ has to hold, but we can show that this cannot be the case. k has to be smaller than j, otherwise the pair cannot become incorrect. Thus, $k < i < j$ has to hold because $k < l$ and $i < j$ and because of our assumption $l = i$. (k,j) was correct before the insertion, so it has to be lexicographically smaller than (i,j). Therefore k is before j in the list of expressed leaf nodes. As i is placed directly before j and therefore after k, (k,l) cannot become incorrect.

The probability for HVL-Prime to perform an insertion is $\frac{1}{3}$, and the probability for the insertion to insert the new leaf at the correct position of the introduced inner J-node is at least $\frac{1}{2}$. This, together with the probability of selecting the right element to add, which is bound by $\frac{1}{n}$, and the probability of adding it to the right position in the tree, which is bound by $\frac{1}{T_{max}}$, gives us a lower bound on the probability for doing such an improvement in (1+1)-GP*-single[2]

$$\frac{1}{3} \cdot \frac{1}{2} \cdot \frac{1}{n} \cdot \frac{1}{T_{max}} = \Omega\left(\frac{1}{nT_{max}}\right).$$

For the multi-operation variant, the probability for a single mutation operation occurring (including the mandatory one) is $\frac{1}{e}$, which is a constant. Thus we have an improvement with probability $\Omega\left(\frac{1}{nT_{max}}\right)$ in the multi-operation case as well. Therefore, the expected optimisation time for both algorithms is upper bounded by

$$\sum_{k=0}^{n\cdot(n-1)/2} O\left(nT_{max}\right) = O(n^3 T_{max}). \qquad \square$$

3.1.2 Local Optima

In the following, we examine our algorithms for the remaining measures of sortedness. We present several worst case examples for HAM, RUN, LAS, and EXC that demonstrate that (1+1)-GP*-single and (1+1)-GP*-multi can get stuck during the optimisation process. This shows that evolving a solution with our GP system is much harder than working with the permutation-based EA presented in [76], where only the sortedness measure RUN leads to an exponential optimisation time.

We study worst case solutions that are hard to improve by our algorithms. In the following, we write down such solutions by the order of the leaves in which they are visited by the inorder parse of the tree. We restrict ourselves to the case where we initialise with a tree of size linear in n and show that even this leads to difficulties for

[2]For example, for the new element to be inserted as the leftmost node of the tree, insertion has to be chosen, then the old leftmost node has to be chosen, and then the new node has to be placed as the left sibling of the old leftmost node, not as it's right sibling.

the mentioned sortedness measures. Note, that a linear size is necessary to represent a complete permutation of the given input elements.

For RUN and LAS, we investigate the initial solution I_{w1} defined as

$$I_{w1} = (\underbrace{n, n, \ldots, n}_{n+1 \text{ instances of n}}, 1, 2, 3, \ldots, n)$$

and show that it is hard for our algorithms to achieve an improvement.

Theorem 3. *Let I_{w1} be the initial solution. Using the sortedness measures RUN and LAS, the expected optimisation time of (1+1)-GP*-single and (1+1)-GP*-multi is infinite and $e^{\Omega(n)}$, respectively.*

Proof. We consider (1+1)-GP*-single first. It is clear that with a single HVL-Prime application, only one of the leftmost ns can be removed. For an improvement in the sortedness based on RUN or LAS, all leftmost $n + 1$ leaves have to be removed at once. Obviously, this cannot be done by the (1+1)-GP*-single, resulting in an infinite runtime.

(1+1)-GP*-multi can only improve the fitness by removing the leftmost $n+1$ leaves. Hence, in order to successfully improve the fitness, at least $n + 1$ sub-operations have to be performed, assuming that we, in each case, delete one of the leftmost ns. Because the number of sub-operations per mutation is distributed as $1 + Pois(1)$, the Poisson random variable has to take a value of at least n. This implies that the probability for such a step is $e^{-\Omega(n)}$ and the expected waiting time for such a step is therefore $e^{\Omega(n)}$, which completes the proof. □

Similarly, we consider the tree I_{w2} defined as

$$I_{w2} = (\underbrace{n, n, \ldots, n}_{n+1 \text{ instances of n}}, 2, 3, \ldots, n-1, 1, n)$$

and show that this is hard to improve the sortedness when using the measures HAM and EXC.

Theorem 4. *Let I_{w2} be the initial solution. Using the sortedness measures HAM and EXC, the expected optimisation time of (1+1)-GP*-single and (1+1)-GP*-multi is infinite and $e^{\Omega(n)}$, respectively.*

Proof. We use similar ideas as in the previous proof. Again, it is not possible for (1+1)-GP*-single to improve the fitness in a single step, as all $n + 1$ leftmost leaves have to be removed in order for the rightmost n to become expressed. Additionally, a leaf labelled 1 has to be inserted at the beginning, or alternatively, one of the $n + 1$ leaves labelled n has to be replaced by a 1. This results in a minimum number of $n+1$

F(X)	(1+1)-GP*	
	single	multi
INV	$O(n^3 T_{max})$	$O(n^3 T_{max})$
HAM	∞	$\Omega\left(\left(\frac{n}{e}\right)^n\right)$
EXC	∞	$\Omega\left(\left(\frac{n}{e}\right)^n\right)$
RUN	∞	$\Omega\left(\left(\frac{n}{e}\right)^n\right)$
LAS	∞	$\Omega\left(\left(\frac{n}{e}\right)^n\right)$

Table 3.1: SORTING: runtime results for (1+1)-GP* variants.

sub-operations that have to be performed by a single HVL-Prime application, leading to the lower bound of $e^{\Omega(n)}$ for (1+1)-GP*-multi. □

The upper and lower bound results for (1+1)-GP* on the problems defined in Section 2.2.2 are summarised in Table 3.1.

3.2 Local Optima and the Parsimony Approach

In this section, we consider simple variable-length evolutionary algorithms using the parsimony approach. The single-objective variant called (1+1)-GP (see Algorithm 2.6 for the single-mutation variant) is identical to the (1+1)-GP* from Section 3.1, with the exception that, in the case of maximisation, Y replaces X if $f(Y) \geq f(X)$ holds. Again, minimisation problems are tackled in the analogous way. Furthermore, in order to introduce the parsimony pressure, we employ the multi-objective variants of the presented sortedness measures, i.e. MO-EXC, MO-RUN, etc.

3.2.1 Upper Bounds

We start our analyses with the cases for which polynomial runtime bounds can be proven. The idea behind the proof of the expected polynomial optimisation time on MO-LAS is as follows. Given a tree T with its tree size of T_{init}, and its sortedness $LAS(T) = k < n$. For such a tree, we always have at least one of the following two ways to create a new tree that is accepted. First, we can improve the sortedness by extending the longest ascending sequence. Or second, we can reduce the size of the tree, if the tree has more than k leaves. If the latter is the case, we can trim the number of leaves down to k, thus eliminating blocking elements and duplicates, and then we can build up the sought permutation. Thus, we can now deal with trees such as I_{w1} from Section 3.1.2, which have previously been problematic.

Theorem 5. *The expected optimisation time of (1+1)-GP-single on MO-LAS is* $O\left(T_{init} + n^2 \log n\right)$.

Proof. For the analysis, we consider two phases. First, we show that we arrive at a tree with fitness k and k leaves after $O\left(T_{init} + n \log n\right)$ steps. Afterwards, we analyse the time needed to get from there to the optimal solution.

1. Phase. Initially, let $LAS(T) = k$ be the fitness of the current tree T with s leaves. Then, the distance to the desired tree size is $d = s - k$. As the probability for HVL-Prime to perform a deletion is $\frac{1}{3}$, the probability to reduce the size via a deletion in a single mutation step can be lower bounded by

$$\frac{1}{3} \cdot \frac{s-k}{s} = \frac{1}{3} \cdot \frac{d}{d+k} \geq \frac{1}{3} \cdot \frac{d}{d+n}.$$

Where the term $\frac{s-k}{s}$ comes from the fact that we need to select one of the redundant elements. Note that d cannot increase as for d to increase, k would have to decrease, which is impossible, as the primary objective is the maximisation of the LAS-value. Alternatively, d could increase if s increases. However, the tree size can only increase if the last accepted step increased the sortedness as well. In a single step, if s increases by 1, then k had to increase by 1 as well, which leaves the distance $s - k = d$ unchanged.

Now, with the fitness-based partitions method over the distance d, we can bound the expected runtime for this first phase:

$$\begin{aligned}
\sum_{d=1}^{T_{init}} 3\frac{d+n}{d} &= 3\sum_{d=1}^{n} \frac{d+n}{d} + 3\sum_{d=n+1}^{T_{init}} \frac{d+n}{d} \\
&\leq 3\sum_{d=1}^{n} \frac{d+n}{d} + 3\sum_{d=n+1}^{T_{init}} 2 \\
&= O\left(n \log n + T_{init}\right).
\end{aligned}$$

2. Phase. Next, we investigate the time needed in the second phase to arrive at the optimum. Therefore, we again apply the above-described fitness-based partitions method. We define the fitness levels $A_1, \ldots, A_n$ with $A_i = \{T \mid LAS(T) = i\}$. As there are at most $n-1$ advancing steps between fitness levels to be made, the total expected runtime is upper bounded by the sum over all expected times needed to make such steps.

After the initial trimming phase, we do not have any blockages that prevent elements from being expressed at their correct positions. Therefore, the existing longest ascending sequence can be extended by inserting *any* of the $n - k$ unblocked elements that are missing in the sequence into its correct position. The probability for a single of such an insertion to happen is at least

$$\frac{1}{3} \cdot \frac{1}{2} \cdot \frac{1}{n} \cdot \frac{n-k}{n} = \frac{1}{6} \cdot \frac{n-k}{n^2}.$$

Thus, the expected runtime of the second phase can then be bounded from above by

$$\sum_{k=1}^{n-1} 6\frac{n^2}{n-k} = 6n \sum_{k=1}^{n-1} \frac{n}{n-k} = O\left(n^2 \log n\right).$$

Hence, the expected optimisation time of the algorithm is upper bounded by $O\left(T_{init} + n^2 \log n\right)$. □

Theorem 6. *The expected optimisation time of (1+1)-GP-single on MO-INV is* $O\left(T_{init} + n^5\right)$.

Proof. For our analysis, we draw upon results from the Theorems 2 and 5. First, after $O\left(n \log n + T_{init}\right)$ steps, we arrive at a non-redundant tree. Next, as we can have at most n^2 fitness-improving insertions, the maximum tree size T_{max} is bounded by $O\left(n + n^2\right)$ after the initial trimming phase. Consequently, the probability for a fitness-improving mutation is bounded by $\Omega\left(\frac{1}{n^3}\right)$. Thus, we can now bound the overall optimisation time by

$$\begin{aligned} O\left(n \log n + T_{init}\right) + \sum_{k=0}^{n \cdot (n-1)/2} O\left(n^3\right) \\ = O\left(n \log n + T_{init}\right) + O(n^5) \\ = O\left(T_{init}\right) + O(n^5) \end{aligned}$$ □

Achieving a similar bound for the multi-mutation variant is not as easy, as the insertion of a missing element (i.e. a fitness improvement), may be accompanied by the insertion of many elements that are already present. Due to the Poisson distributed number of operations performed by HVL-Prime within (1+1)-GP-multi, the algorithm's *typical* local behaviour is difficult to predict.

Therefore, we take an alternate approach, by looking at a sequence of steps $t = poly(n)$. Let T_{init} to be a tree with size $|T_{init}| = poly(n)$. The failure probability for inserting at most n^ϵ in a single HVL-Prime operation is $e^{-\Omega(n^\epsilon)}$. Furthermore, given any initial tree, we can have at most n improvements of the sortedness when the measurements LAS and EXC are used. Now, we compute a bound of the tree size. Looking at n mutations that increase the fitness, the failure probability for adding at most $nn^\epsilon = n^{1+\epsilon}$ leaves in t time steps is exponentially small: $te^{-\Omega(n^\epsilon)} = e^{-\Omega(n^\epsilon)}$. Thus, the tree size does not exceed $T_{max} = T_{init} + n^{1+\epsilon}$ within $t = poly(n)$ time steps, with high probability.

Theorem 7. *Let $\epsilon > 0$ be a constant. The optimisation time of (1+1)-GP-multi on MO-LAS is $O\left(T_{init} + n^2 \log n\right)$, with probability $1 - o(1)$.*

Proof. We will split the proof into two parts: first, we bound the total time needed for deletions during a run, and second, we investigate the time needed to perform the necessary insertions to find the optimal solution.

First, given a solution where k_m elements have to be removed in order to arrive at a non-redundant tree after the m-th fitness-increasing insertion. In the following, let i be the number of redundant elements in the tree, and let j be the number of non-redundant elements in the tree.

Stage 1, $i \geq n+1$.

As the probability for a single operation is $\frac{1}{e}$, the probability for the deletion of a single redundant element at any time is lower bounded by

$$\frac{1}{3e}\frac{i}{i+j} \geq \frac{1}{3e}\frac{i}{i+n} \geq \frac{1}{3e}\frac{1}{2} = \frac{1}{6e}.$$

Then, the expected time to delete k_m elements is upper bounded by $6ek_m$. Furthermore, as we know that we can delete at most T_{max} leaves over a full optimisation run, $\sum_{i=1}^{n} k_i \leq T_{max}$. Thus, we can bound the expected time needed for *all* deletions (when $i \geq n+1$) by $6eT_{max}$.

Let $X_1, \ldots, X_d$ be independent random variables taking value 1 with $Prob(X_i = 1) = \frac{1}{6e}$ if an element is deleted (in time step $1 \leq t \leq d$), and 0 otherwise. With Chernoff's inequality (with $\delta = 1$, see Section 2.4) we get that

$$\begin{aligned} Prob\left(X \geq 12eT_{max}\right) &= Prob\left(X \geq 12e(T_{init} + n^{1+\epsilon})\right) \\ &\leq e^{-2e(T_{init}+n^{1+\epsilon})} \leq e^{-\Omega\left(n^{1+\epsilon}\right)}. \end{aligned}$$

Stage 2, $i \leq n$.

To bound the number of steps, we apply the technique of multiplicative drift with tail bounds (see Section 2.4).

In our case, $\Phi(x) = i$ is a feasible ν-drift function on the number of redundant elements (with implicit constant $\delta = 1$). For the optimal solutions ("no redundant elements left") $\Phi(x) = 0$ holds as required, $\Phi(x) \geq 1$ holds for all non-optimal solutions, and $E[\Phi(x_{new})] \leq \left(i - \frac{i}{6en}\right) = \left(1 - \frac{1}{\nu(n)}\right)\Phi(x)$. Thus, $\nu(n) = 6en$ and $\delta = 1$. Consequently, we get that the time needed for all deletions (when $i \leq n$) during a run exceeds $6en(\ln n + n \ln n)$ with probability at most n^{-c}. As these deletion phases take place at most n times, the resulting overall deletion time does not exceed $O(n^2 \log n)$ with probability $1 - n^{-c+1} = 1 - o(1)$.

Next, we consider the time necessary to perform the insertions of the missing elements, once the insertion was unblocked. We will again apply the multiplicative drift with tail bounds, as used above. Note, that the situation is very similar: instead of reducing the number of redundant elements, we are now reducing the number of missing elements.

Let j be the number of elements currently missing. As the probability for a single operation is $\frac{1}{e}$, the probability for a single insertion of a missing element to happen at the required position is lower bounded by $\frac{1}{3e}\frac{1}{2n}\frac{j}{n} = \frac{j}{6en^2}$. With $E[\Phi(x_{new})] \leq \left(j - \frac{j}{6en^2}\right) = \left(1 - \frac{1}{\nu(n)}\right)\Phi(x)$ we get, $\nu(n) = 6en^2$ and $\delta = 1$. Consequently, by applying Theorem 1, we get that the time needed for all insertions during a run exceeds $6en^2(\ln n + n \ln n)$ with probability at most n^{-c}. Thus, the resulting overall time needed for all insertions does not exceed $O(n^2 \log n)$ with probability $1 - n^{-c} = 1 - o(1)$.

To conclude, based on our analysis of the necessary deletions and insertions, the optimisation time of (1+1)-GP-multi on MO-LAS is $O\left(T_{init} + n^2 \log n\right)$, with probability $1 - o(1)$. □

3.2.2 Local Optima

In the following, we show that the parsimony approach can still lead to local optima for various types of sortedness measure.

Let $I_{w3} = (n, 2, 3, \ldots, n-3, n-2, n-1, 1)$ be the initial solution. We point out that this is a local optimum for (1+1)-GP-single on MO-EXC leading to an infinite optimisation time.

Theorem 8. *Let I_{w3} be an initial solution. Then the optimisation time of (1+1)-GP-single on MO-EXC is infinite.*

Proof. The individual I_{w3} has an EXC-value of 1 and a length of n. In order to reduce the fitness down to 0, it would be necessary to move the n from the head of the permutation to its end.

For this to happen, deletions and substitutions cannot be considered, as they would produce incomplete permutations, and incomplete permutations have EXC-values of at least 2.

Similarly, this situation cannot be solved using a single insertion: it is not possible to introduce n at its correct position within the permutation, as the existing n is preventing the new one from becoming expressed.

Therefore, none of the available operations can improve the number of elements sitting at their correct (relative) position via a single mutation. Thus, (1+1)-GP-single takes infinitely long, when initialised with I_{w3}. □

We continue by investigating the sortedness measure RUN. Without loss of generality, let n be even and $I_{w4} = \left(\frac{n}{2}+1, \frac{n}{2}+2, \ldots, n-1, n, 1, 2, \ldots, \frac{n}{2}-1, \frac{n}{2}\right)$ be an initial solution. The following theorem shows that I_{w4} is a local optimum for (1+1)-GP-single on MO-RUN.

Theorem 9. *Let I_{w4} be the initial solution. Then the optimisation time of (1+1)-GP-single on MO-RUN is infinite.*

Proof. The individual I_{w4} has a RUN-value of 2, which cannot be improved via a single insertion: $n/2$ elements have to change their positions in the inorder parsed list that is used for the computation of the RUN-value. Furthermore, a single deletion or substitution results in a worse sortedness value as one element is then missing (as defined in Section 2.2.2). Therefore, the runtime of the single-operation case of (1+1)-GP is infinite, when initialised with this particular individual. □

Finally, we consider the sortedness measure HAM and investigate the initial solution $I_{w5} = (1, n-2, 3, 4, 5, \dots, n-3, 2, n-1, n)$. We show that this is a local optimum for MO-HAM.

Theorem 10. *Let I_{w5} be the initial solution. Then the optimisation time of (1+1)-GP-single on MO-HAM is infinite.*

Proof. The individual I_{w5} has the elements 2 and $n-2$ at incorrect positions, resulting in a HAM-value of $n-2$. It is not possible to maintain the HAM-value (or improve it) via deletions, as they decrease the number of elements at correct positions. Substitutions can also not maintain the HAM-value. A substitution of the $n-2$ by 2 would result in the elements to the right to shift away from their correct position as in both cases the element at the third position would no longer get expressed. This leaves only the option of using insertions, in order to generate an individual that is accepted. The element $n-2$ cannot be introduced successfully at its correct position as its current occurrence is blocking a later expression. If 2 is introduced at its correct position, then the resulting permutation is $(1, 2, n-2, 3, 4, 5, \dots, n-3, n-1, n)$ as the second two no longer gets expressed, and the corresponding HAM-value for this permutation is $n-3$.

Thus, the runtime of the single-operation case of (1+1)-GP is infinite, when initialised with this particular individual. □

3.3 Multi-Objective Approach

In this section, we consider the Simple Evolutionary Multi-Objective Genetic Programming (SMO-GP) algorithm introduced by Neumann [62] and motivated by the SEMO algorithm for fixed length representations by Laumanns et al. [55]. Let us recall that SMO-GP (see Algorithm 2.7) is a population-based approach that starts with a single solution and keeps in each iteration a set of non-dominated solutions obtained during the optimisation run. This set of solutions constantly approximates the true Pareto front, i.e. the set of optimal trade-offs between fitness and complexity. In each iteration, it picks one solution uniformly at random and produces one offspring Y by mutation. Y is introduced into the population iff it is not weakly dominated by any other solution in P. If Y is added to the population all individuals that are dominated by Y are discarded.

In the following, we analyse the performance of the SMO-GP variants on each one of the fitness functions introduced in Section 2.2.2. In particular, we analyse the expected number of iterations before the set of non-dominated solutions becomes the true Pareto front. We call this the *expected optimisation time* of SMO-GP algorithms.

The following lemma bounds the expected time until the empty solution has been included into the population, when considering an arbitrary optimisation problem:

Lemma 1 (Neumann [62]). *Let I_{init} be the size of the initial solution and k be the number of different fitness values of a problem F. Then the expected time until the population of SMO-GP-single and SMO-GP-multi applied to MO-F contains the empty solution is $O\left(kI_{init}\right)$.*

Theorem 11. *The expected optimisation time of SMO-GP-single and SMO-GP-multi is $O(nI_{init} + n^3 \log n)$ on MO-EXC and MO-RUN, and $O(nI_{init} + n^4)$ on MO-HAM.*

Note that for all three problems, only solutions of complexity $2i - 1$, $1 \leq i \leq n$, and the empty solution can be Pareto optimal. For RUN, these are solutions X with $C(X) = 2i-1$ and $\text{RUN}(X) = n+1-i$, $1 \leq i \leq n$, and the empty tree with $C(X) = 0$ and $\text{RUN}(X) = n + 1$.[3]

Proof. In the following, we will first prove the theorem for MO-RUN. The proofs for MO-EXC and MO-HAM follow the same structure.

RUN has $n + 1$ different fitness values. Using Lemma 1, the empty solution is produced after an expected number of $O\left(nI_{init}\right)$ steps. Note that the empty solution will never be removed from the population as it is the unique solution having complexity zero.

In the following steps, we will bound the time needed to discover the whole Pareto front, once the empty solution is introduced into the population. Let us assume that the population contains all Pareto optimal solutions with complexities $2j-1$, $1 \leq j \leq i$. Then, a population that includes all Pareto optimal solutions with complexities $2j - 1$, $1 \leq j \leq i + 1$, can be achieved by producing a solution Y that is Pareto optimal and that has complexity $2(i + 1) - 1$. Y can be obtained from a Pareto optimal solution X with $C(X) = 2i - 1$ by inserting any of the $n - i$ missing elements into the correct position. This operation produces from a solution of complexity $2i - 1$ a solution of complexity $2(i + 1) - 1 = 2i + 1$, as one leaf node and one inner node are added.

Based on this idea we can bound the expected optimisation time once we can bound the probability for such steps to happen. Choosing X for mutation has probability at

[3]For the sake of readability, we will omit the empty solution in the remainder of this section when addressing sets of trees that cover an interval of complexities.

least $1/(n+1)$ as the population size is upper bound by $n+1$. Next, the mutation step carrying out just one operation happens with at least $1/e$, and the inserting operation of HVL is chosen with probability $1/3$. As $n-i$ out of the n elements are missing, any of those can be inserted. However, the correct position for such a randomly chosen element has to be chosen, in order to produce the Pareto optimal solution of complexity $i+1$. This probability is at least $1/2 \cdot 1/n$, as the number of leaf nodes is bound by n, and the probability to insert as the correct child of the newly introduced inner node is at least $1/2$. Thus, the total probability of such a generation is

$$\frac{1}{n+1} \cdot \frac{1}{3e} \cdot \frac{1}{2n} \cdot \frac{n-i}{n}.$$

Now, we use the method of fitness-based partitions (see Section 2.4) according to the $n+1$ different fitness values of i. Thus, as there are only n Pareto-optimal improvements possible once the empty solution is introduced into the population, the expected time until all Pareto optimal solutions have been generated is:

$$\begin{aligned}\sum_{i=0}^{n} \left(\frac{1}{n+1} \cdot \frac{1}{3e} \cdot \frac{1}{2n} \cdot \frac{n-i}{n} \right)^{-1} &= 6en^2(n+1) \cdot \sum_{i=0}^{n} \frac{1}{n-i} \\ &= O(n^3 \log n).\end{aligned}$$

Taking into account the expected time to produce the empty solution, the expected time until the whole Pareto front of MO-RUN has been computed is $O\left(nI_{init} + n^3 \log n\right)$.

The proof for MO-EXC follows the same structure. First, note that if the ordering within the permutation requires an exchange, then this individual is dominated by individuals of same complexity that require fewer exchanges. Just as with MO-RUN, let us assume that the population contains all Pareto optimal solutions with complexities $2j-1$, $1 \leq j \leq i$. Then, a population which includes all Pareto optimal solutions with complexities $2j-1$, $1 \leq j \leq i+1$, can be achieved by inserting any of the $n-i$ missing elements into the correct position of the Pareto optimal individual X with $C(X) = 2i-1$. The probability for such a step to happen is at least $\frac{1}{n+2} \cdot \frac{1}{3e} \cdot \frac{1}{2n} \cdot \frac{n-i}{n}$. Now, as $n+2$ different EXC-values are possible, and by summing up the waiting times as done for MO-RUN, the expected optimisation time is $O(nI_{init} + n^3 \log n)$.

Similarly, we can prove an upper bound for MO-HAM. First, note that each Pareto optimal solution with HAM-value i represents a perfectly sorted permutation of the i elements $1, \ldots, i$. Just as above, let us assume that the population contains all Pareto optimal solutions with complexities $2j-1$, $1 \leq j \leq i$. Then, a population which includes all Pareto optimal solutions with complexities $2j-1$, $1 \leq j \leq i+1$, can be achieved by inserting the element $i+1$ into its correct position (i. e., as the rightmost leaf) in the Pareto optimal individual X with $\text{HAM}(X) = C(X) = 2i-1$. The probability for such a step to happen is at least $\frac{1}{n+1} \cdot \frac{1}{3e} \cdot \frac{1}{2n} \cdot \frac{1}{n} = \Omega\left(\frac{1}{n^3}\right)$ and the corresponding waiting time is $O(n^3)$. There are $n+1$ different HAM-values. This implies that the expected optimisation time is $O(nI_{init} + n^4)$. $\square$

Theorem 12. *The expected optimisation time of SMO-GP-single and SMO-GP-multi is $O(n^2 I_{init} + n^5)$ on MO-INV, and $O(nI_{init} + n^3 \log n)$ on MO-LAS.*

Proof. Our analysis follows the proof of Theorem 11. First, as INV has $n(n-1)$ different fitness values, using Lemma 1, the empty solution is produced after an expected number of $O\left(n^2 I_{init}\right)$ steps. First, note that each Pareto optimal solution with complexity $2i-1$ has an INV-value of $\sum_1^{i-1} i$, if $i \geq 2$.[4]

Second, as above, we will bound the time needed to discover the whole Pareto front, once the empty solution is introduced into the population. Let us assume that the population contains all Pareto optimal solutions with complexities $2j-1$, $1 \leq j \leq i$. Then, a population that includes all Pareto optimal solutions with complexities $2j-1$, $1 \leq j \leq i+1$, can be achieved by producing a solution Y that is Pareto optimal and that has complexity $2(i+1)-1$. Y can be obtained from a Pareto optimal solution X with $C(X) = 2i-1$ by inserting an element that increases the INV-value by $i-1$. This operation produces from a solution of complexity $2i-1$ a solution of complexity $2(i+1)-1 = 2i+1$, as one leaf node and one inner node are added. The probability of this operation to happen can be bounded by $\frac{1}{n(n-1)/2+1} \cdot \frac{1}{3e} \cdot \frac{1}{2n} \cdot \frac{1}{n}$.

Thus, as there are only n Pareto-optimal improvements possible once the empty solution is introduced into the population, the expected time until all Pareto optimal solutions have been generated is:

$$\sum_{i=0}^{n} \left(\frac{1}{n(n-1)/2+1} \cdot \frac{1}{3e} \cdot \frac{1}{2n} \cdot \frac{1}{n} \right)^{-1} = 6en^5 = O(n^5).$$

Similarly, we can prove an upper bound for MO-LAS. First, note that each Pareto optimal solution with LAS-value i represents a perfectly sorted permutation of i elements. Next, as only n different LAS-values are possible, the empty solution is produced after an expected number of $O\left(nI_{init}\right)$ steps. Just as above, let us assume that the population contains all Pareto optimal solutions with complexities $2j-1$, $1 \leq j \leq i$. Then, a population that includes all Pareto optimal solutions with complexities $2j-1$, $1 \leq j \leq i+1$, can be achieved by inserting any of the missing $n-i$ elements into its correct position in the Pareto optimal individual X with $\text{LAS}(X) = C(X) = 2i-1$.

Thus, as there are only n Pareto-optimal improvements possible once the empty solution is introduced into the population, the expected time until all Pareto optimal solutions have been generated is:

[4]For the sake of readability, we will omit in the following the special cases for $i = 0$ and $i = 1$.

F(X)	(1+1)-GP*, F(X)		(1+1)-GP, F(X)
	single	multi	single/multi
INV	$O(n^3 T_{max})$	$O(n^3 T_{max})$	
LAS	∞	$\Omega\left(\left(\frac{n}{e}\right)^n\right)$	
HAM	∞	$\Omega\left(\left(\frac{n}{e}\right)^n\right)$	?
EXC	∞	$\Omega\left(\left(\frac{n}{e}\right)^n\right)$	
RUN	∞	$\Omega\left(\left(\frac{n}{e}\right)^n\right)$	

Table 3.2: Summary of computational complexity bounds for single-objective variants. The question mark indicates combinations for which we do not know any bounds.

$$\sum_{i=0}^{n}\left(\frac{1}{n+1}\cdot\frac{1}{3e}\cdot\frac{1}{2n}\cdot\frac{n-i}{n}\right)^{-1} = 6en^2(n+1)\cdot\sum_{i=0}^{n}\frac{1}{n-i}$$
$$= O(n^3\log n). \qquad \square$$

3.4 Experimental Supplements to the Computational Complexity Analysis

In this section, we carry out experimental investigations about the runtime of different variable-length algorithms over the presented fitness functions. The purpose of this analysis is threefold

- to complement the theoretical results with conjectures about the expected optimisation times for the variants lacking a formal proof,
- to assess the impact on the runtime of two collected measures, namely the maximal tree size T_{max} and, for SMO-GP, the maximal population size P_{max} encountered during an optimisation run, and
- to give useful insight for guiding further rigorous theoretical analysis.

A similar experimental approach has been employed by [8] and Lässig and Sudholt [53]. In addition, we have used the same approach to complement theoretical investigations on WORDER and WMAJORITY (see Chapter 4).

3.4.1 Theoretical results

As already stated, the computational complexity analysis of genetic programming analyses the so-called *expected optimisation time* of an algorithm. In single-objective contexts, such as the one of (1+1)-GP, the expected optimisation time measures the number of fitness evaluations until an algorithm has produced the first optimal solution,

F(X)	(1+1)-GP, MO-F(X)		SMO-GP, MO-F(X)
	single	multi	single/multi
INV	$O(T_{init} + n^5)$	?	$O\left(n^2 T_{init} + n^5\right)$
LAS	$O(T_{init} + n^2 \log n)$	$O(T_{init} + n^2 \log n)$ †	$O(nT_{init} + n^3 \log n)$
HAM	∞	?	$O(nT_{init} + n^4)$
EXC	∞	?	$O(nT_{init} + n^3 \log n)$
RUN	∞	?	$O(nT_{init} + n^3 \log n)$

Table 3.3: Summary of computational complexity bounds for multi-objective variants. † indicates a bound that holds with probability $1 - o(1)$. Question marks indicate combinations for which we do not know any bounds.

which is a common performance metric in evolutionary computation. Table 3.2 summarises the theoretical findings of Section 3.1 about our single-objective setups. In multi-objective algorithms, such as SMO-GP, however, the number of evaluations until the first optimal solution is not informative enough, as the goal is to reconstruct or approximate as much as possible the true Pareto front, i.e. the whole set of optimal trade-offs between objectives. In these context, the number of evaluations to produce the complete front is considered instead. Table 3.3 summarises the theoretical findings of Sections 3.2 and 3.3 for our multi-objective setups.

As can be observed from the tables, all bounds take into account tree sizes of some kind: either the maximum solution size T_{max}, or the size of the initial solution T_{init}. In particular, the runtime of (1+1)-GP, F(X) depends on the maximum tree size T_{max}, since the expected time to get to the optimal solution grows larger and larger as the tree grows in size, and the runtimes of several MO-F(X) variants are dependent on the initial tree size T_{init} as often the first step of the proof involves deconstructing the original solutions until a tree of size zero is found.

3.4.2 Experimental setup

In our experimental investigations, we consider (1+1)-GP on F(X), (1+1)-GP on MO-F(X), (1+1)-GP* on F(X), and SMO-GP on MO-F(X). Each GP algorithm is run in its single-operation and multi-operation variants, and we investigate problems of sizes $n = 20, 40, 60, \ldots, 200$. For the initialisation of the individuals, we consider the schemes init_0 (empty tree) and init_n (tree with n leaves constructed by applying n insertion mutations at random positions on an initially empty tree). In total, our experiments span ten problems: INV, HAM, RUN, LAS and EXC in their single and multi-objective variants. All the experiments were performed on AMD Opteron 250 CPUs (2.4GHz), on Debian GNU/Linux 5.0.8, with Java SE RE 1.6 and were given a maximum runtime of 3 hours and a budget of 10^9 evaluations each. Furthermore, each experiment has been repeated 200 times, which results in a standard error of the mean (the standard deviation of the sampling distribution) of $1/\sqrt{200} = 7\%$. As a curiosity, the whole set of experiments took about 30 CPU-years to complete.

The complete source code of the employed framework is available on BitBucket (Mercurial, at https://bitbucket.org/tunnuz/gpframework), on GitHub (Git, https://github.com/tunnuz/gpframework) and Google Code (Subversion at http://code.google.com/p/gpframework).

3.4.3 (1+1)-GP variants

We now analyse the experimental results on (1+1)-GP variants with respect to the maximum tree size obtained during execution and the required optimisation time.

Tree size

As we have seen, the known theoretical bounds for the (1+1)-GP variants presented in Section 3.2 depend on T_{max}, the maximal solution size that is encountered during the run of the algorithm. It is important to observe that the maximal solution size is not a parameter that is set in advance, but rather a measure that emerges from the nature of the employed fitness function and mutation operators. In addition, the can might involve a degree of randomness, which makes T_{max}, and thus bloating, extremely difficult to predict. For this reason, we investigate the maximum solution size experimentally in order to detect when bloat occurs within the analysed algorithms. As statistics, we employ the median (the second quartile) as a measure of central tendency and the interquartile range (iqr, the distance between the first and the third quartiles) as a measure of variance.

Table 3.4 reports results for $n = 40, 80, 160$, but similar results hold for the other input sizes. The missing data (-) represent experiments for specific input sizes where the algorithms did not make it to an optimal solution within the time or evaluations bound for more than 50% of the repetitions. For the sake of clarity we recall that (1+1)-GP*, F(X) accepts a new solution only if the fitness is strictly better than the previous one, while (1+1)-GP, F(X) always accepts a solution of the same value. (1+1)-GP, MO-F(X) accepts a solution of the same fitness only if the complexity is lower.

We first analyse T_{max} for the single-operation variant, where a single mutation operator is applied at each step (upper half of Table 3.4). Here (1+1)-GP* and (1+1)-GP, MO-F(X) share similar tree sizes of about $2n - 1$ (sometimes $2n + 1$), which is a minimum for the optimal solution, on all fitness values but INV, where (1+1)-GP, MO-F(X) obtains a tree size of about $2.3n$ on both initialization schemes and (1+1)-GP*

k	F(X)	n	(1+1)-GP*, F(X)				(1+1)-GP, F(X)				(1+1)-GP, MO-F(X)			
			init$_0$		init$_n$		init$_0$		init$_n$		init$_0$		init$_n$	
			m	iqr	m	iqr	m	iqr	m	iqr	m	iqr	m	iqr
		40	307	46	327	33.5	528	185.5	528	202.5	95	4	97	5.5
	INV	80	821	79	849	105	1259	472	1269	473	189	8	191	6
		160	-	-	-	-	2645	612	2688	627.5	375	10	381	14
		40	79	0	-	-	525	212	592	265.5	79	0	79	0
	LAS	80	159	0	-	-	1352	508.5	1401	526.5	159	0	159	0
		160	319	0	-	-	2670	527.5	-	-	319	0	319	0
		40	79	0	-	-	1665	1042.5	1672	723.5	79	0	79	0
k=1	HAM	80	159	0	-	-	-	-	-	-	159	0	159	0
		160	319	0	-	-	-	-	-	-	319	0	319	0
		40	81	0	-	-	1573	908	-	-	81	2	-	-
	EXC	80	161	0	-	-	-	-	-	-	161	0.5	-	-
		160	321	2	-	-	-	-	-	-	321	0	-	-
		40	79	0	-	-	-	-	-	-	79	0	-	-
	RUN	80	159	0	-	-	-	-	-	-	159	0	-	-
		160	319	0	-	-	-	-	-	-	319	0	-	-
		40	249	33	259	34	512	183	543	199.5	107	8	112	10
	INV	80	611	48	627	58	1245	490	1308	435.5	213	12.5	219	14
		160	-	-	-	-	2793	733	2821	700	419	18	437	22
		40	95	10	-	-	555	276.5	560	261.5	79	2	79	2
	LAS	80	187	10.5	-	-	1334	592	1382	420.5	159	2	159	2
		160	-	-	-	-	2893	698	2789	498	319	2	319	2
		40	87	6	-	-	1767	926	1833	1042	79	2	79	2
k=1+Pois(1)	HAM	80	177	8	-	-	-	-	-	-	159	0	159	2
		160	353	11.5	-	-	-	-	-	-	319	2	319	2
		40	93	6	-	-	1852	1042.5	1964	964.5	81	0	83	2
	EXC	80	-	-	-	-	-	-	-	-	161	2	-	-
		160	-	-	-	-	-	-	-	-	-	-	-	-
		40	-	-	-	-	-	-	-	-	-	-	-	-
	RUN	80	-	-	-	-	-	-	-	-	-	-	-	-
		160	-	-	-	-	-	-	-	-	-	-	-	-

Table 3.4: Maximum tree sizes encountered until the individual X_{opt} with optimal fitness is found. Shown are median m and median interquartile ranges iqr.

shows a tree size close to $10n$. On the other hand, (1+1)-GP, F(X) appears cursed by bloating in all fitness functions, with tree sizes above $12n$. Nonetheless, unlike

(1+1)-GP*, (1+1)-GP, F(X) seems independent on the employed initialization scheme and can reach optimal solutions for INV and LAS even with init$_n$ where (1+1)-GP* fails. As for the interquartile range, (1+1)-GP, MO-F(X) appears to be the most stable algorithm with an *iqr* of zero on all fitness functions but INV. Overall, the best algorithm with respect to tree size, interquartile range and robustness with respect to initialization schemes is (1+1)-GP with parsimony (MO-F(X) variant).

As for the multi-operation variant, i.e. where $k = 1 + Pois(1)$ applications of each mutation operator are executed at each step, tree sizes increase in every algorithmic variant on INV. On the other fitness functions, the negative impact of multiple operations appears especially on the F(X) variants, while (1+1)-GP, MO-F(X) is less susceptible to this parameter. Overall, the interquartile range in the tree size increases along with it. Again, with respect to tree size, interquartile range and initialization scheme independence the best algorithm is (1+1)-GP with parsimony (MO-F(X) variant).

The plots in the remainder of this chapter indicate the asymptotic behaviour of the investigated measures. In particular, they include

- the *distributions* of values, represented as box plots,
- the *failure rate*, i.e. the fraction of repetitions that did not make it to the end because of the imposed timeout or evaluations budget, represented as different colour tones of the box plots,
- two blue-toned lines representing, for each input size, the medians of the distributions divided by some polynomial, whose interpretation gives an indication of the asymptotic behaviour of the measure.

In order to deduce the asymptotic behaviour of a measure, one must look at the polynomial line that is closest to constant (i.e. the 'most horizontal' one). A horizontal line means that, barred a multiplicative factor, the measure behaves like the corresponding polynomial, at least for the analysed input sizes. In all plots, we have excluded the input sizes where a failure rate above 50% did not allow to compute a reliable median (and thus to obtain a reliable estimate on the asymptotic behaviour).

Figures 3.2 and 3.1 show the distribution of T_{max} (1+1)-GP for increasing values of n, respectively with the $init_0$ and $init_n$ initialization schemes.

From the figure T_{max} looks close to linear for (1+1)-GP, MO-F(X) variants on both initialization schemes and mutation variants, however its behaviour looks closer to $n \log n$ on INV and LAS and as n^2 for HAM and EXC in the $init_n$ initialization scheme with both mutation variants. As for (1+1)-GP with $init_0$, the variant accepting only strict improvement reaches a T_{max} linear in n for all fitness functions except INV and the one accepting neutral moves a T_{max} growing as $n \log n$ in INV and LAS. Overall

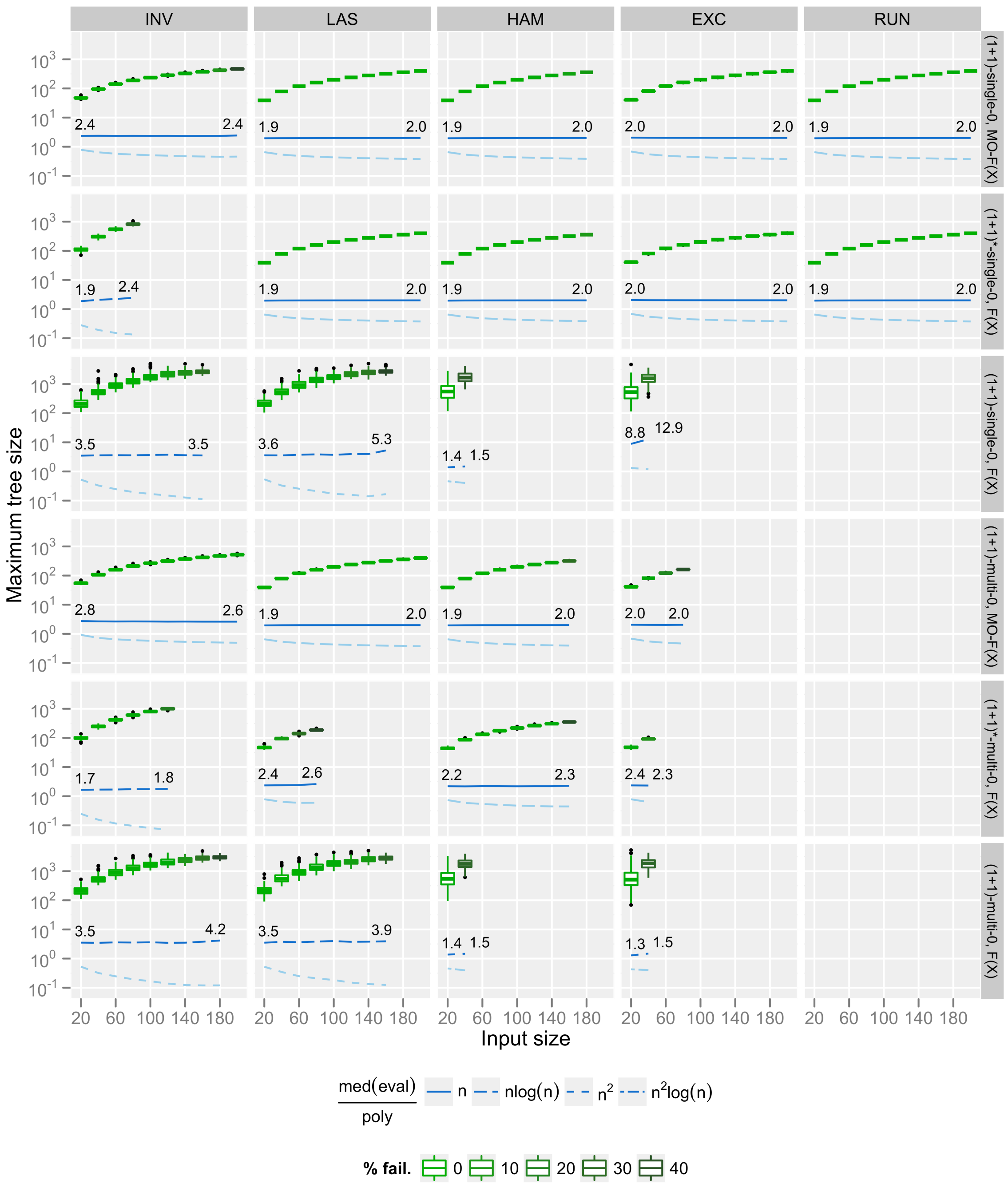


Figure 3.1: Maximum tree sizes T_{max} observed when init_0 is used. Note that no data is shown, when less than 50% of the runs were successful. Note that several algorithms have significant problems solving RUN, and as a consequence no distributions of T_{max} can be given. The same holds for larger instances of HAM and EXC.

the $init_0$ initialization scheme seems to be beneficial in terms of failure rate and growth of T_{max}.

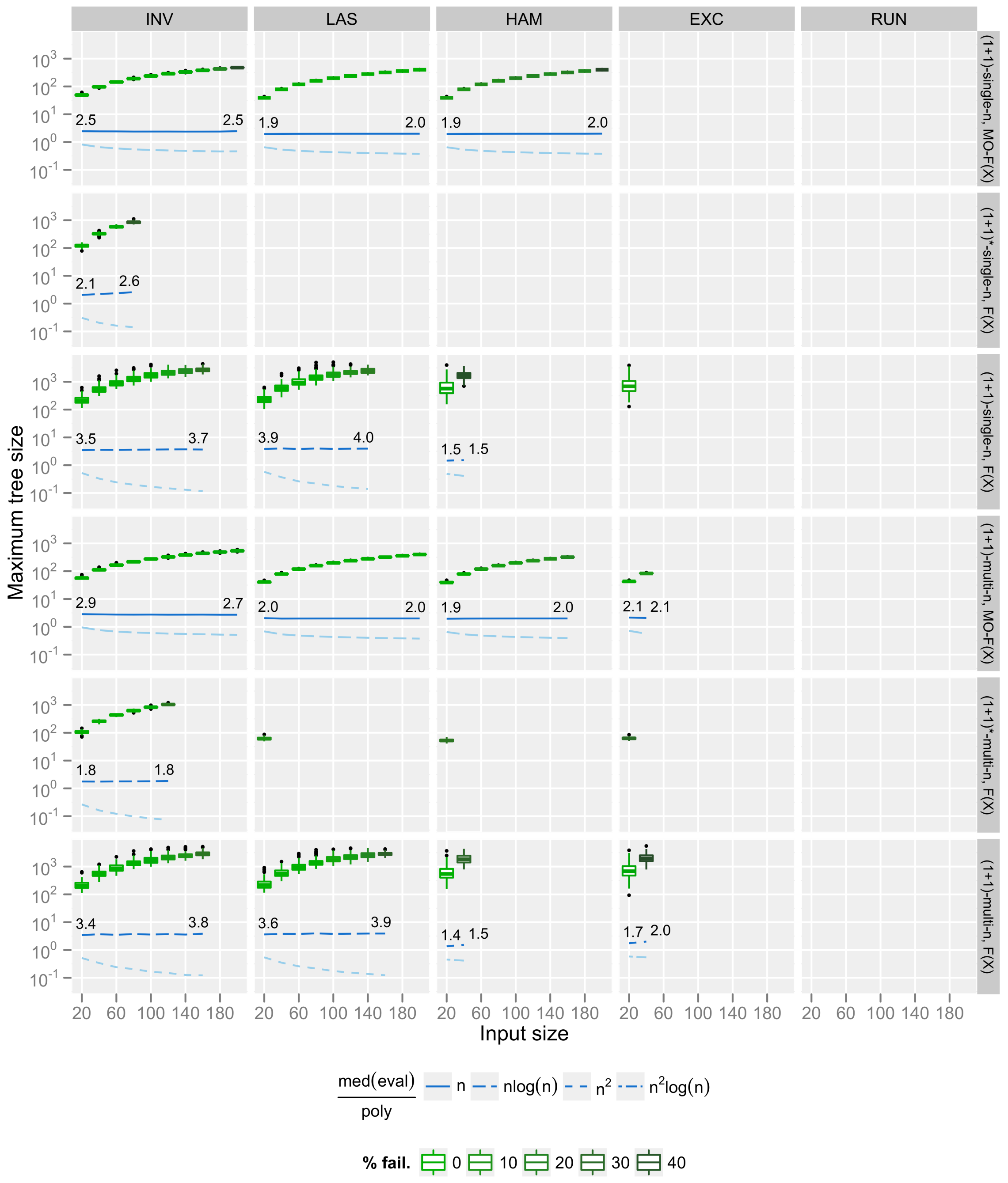


Figure 3.2: Maximum tree sizes T_{max} observed when init$_n$ is used. Note that no data is shown, when less than 50% of the runs were successful. When one compares these results with those of Figure 3.1, it becomes obvious that the initialisation with n leaf nodes can render the problem unsolvable.

Average case optimisation time

Figures 3.3 and 3.4 show the distributions of the required number of evaluations to reach the first optimal solution for the (1+1)-GP variants respectively in the init$_0$ and

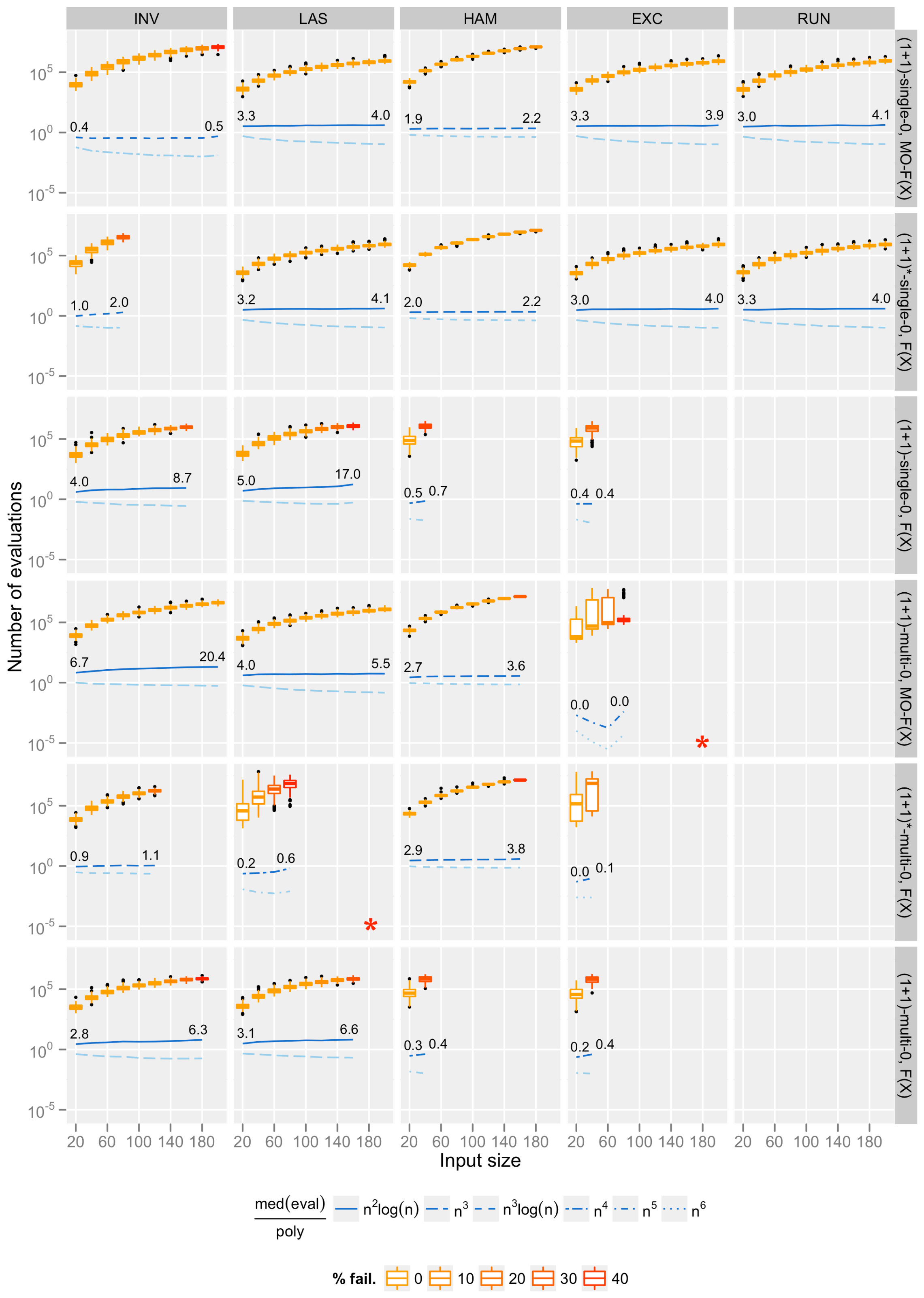


Figure 3.3: Number of evaluations required by (1+1)-GP (initialised with init$_0$) until the individual X_{opt} with optimal fitness is found, shown as box plots. Note that no data is shown, when less than 50% of the runs were successful. Consequently, one can see that the algorithms have problems solving even small instances of RUN. The * mark configurations in which the method to find the upper and lower polynomials is unreliable because of inflections in the number of evaluations for some input sizes.

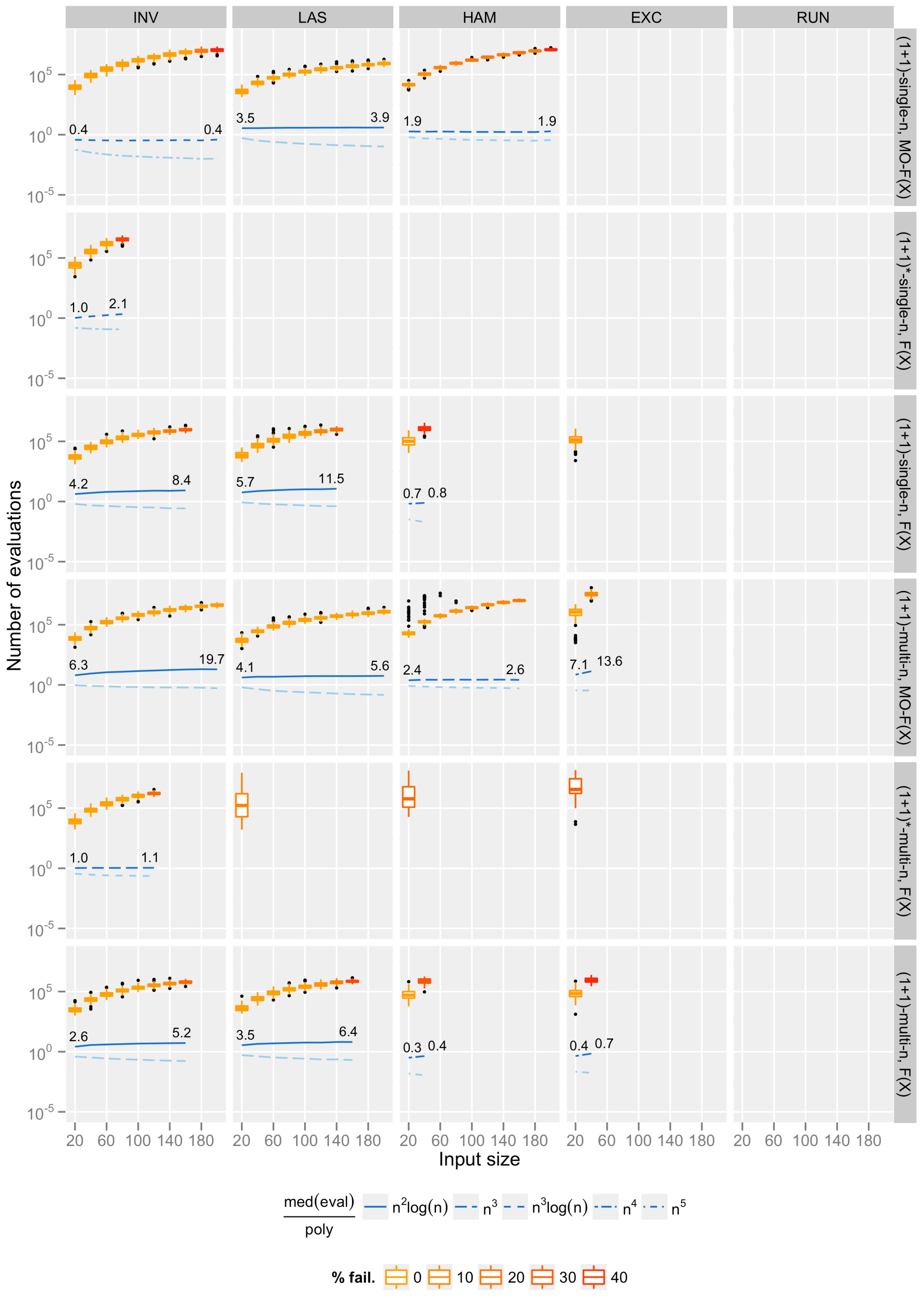


Figure 3.4: Number of evaluations required by (1+1)-GP (initialised with init_n) until the individual X_{opt} with optimal fitness is found, shown as box plots. Note that no data is shown, when less than 50% of the runs were successful. When one compares these results with those of Figure 3.3, it becomes obvious that the initialisation with n leaf nodes can render the problem unsolvable.

Line style	meaning
solid	$n^2 \log n$
long dashed	n^3
short dashed	$n^3 \log n$
dot dashed	n^4
short dot dashed	n^5
dotted	n^6

Table 3.5: Meaning of line styles in Figures 3.3 and 3.4.

init_n initialization schemes. The meaning of line styles in the figures is explained in Table 3.5.

By analysing the results it can be noted that overall the init_0 initialization scheme is beneficial for (1+1)-GP*, F(X) both in single- and multi-operation modes, allowing it to optimise every fitness function in single-operation mode and to reach some optima for all fitness functions except RUN for multi-operation mode. On the contrary, (1+1)-GP does not seem to be influenced significantly by a particular choice of initial individuals. The performances of the two initialization schemes are identical for INV across every algorithmic variant but consistently worse for init_n in all the other fitness values, at least in terms of failure rate. In general, initializing the population with full trees appears to be an obstacle to optimisation. Also, multi-operation when applied with the init_0 scheme appears to be detrimental.

The theoretical bounds are confirmed by the experiments, suggesting that they might be tight.

3.4.4 SMO-GP

We now analyse the experimental results on SMO-GP variants with respect to the maximum tree and population size obtained during execution and the expected optimisation time.

Table 3.6 shows the maximum tree sizes and maximum population sizes that were observed up to the following two events. First, until the individual X_{opt} with optimal fitness is found, and secondly, until the population represents the entire true Pareto front P_{Pareto}.

Tree size

With respect to maximum tree size we can note that with both init_0 and init_n initialization schemes and both single- and multi-operation variants the tree size is always very close to the theoretical minimum of $2n - 1$ except for INV in which we have an

		F(X)	n	maximum tree size				max. population size			
				to X_{opt}		to P_{Pareto}		to X_{opt}		to P_{Pareto}	
				m	iqr	m	iqr	m	iqr	m	iqr
SMO-GP, with k=1	init$_0$	INV	80	169	4	169	4	85	1	85	1
		LAS	80	159	0	159	0	81	0	81	0
		HAM	80	159	0	159	0	81	0	81	0
		EXC	80	159	2	159	2	81	1	81	1
		RUN	80	159	0	159	0	81	0	81	0
	init$_n$	INV	80	173	6	173	6	86	2	86	2
		LAS	80	159	0	159	0	81	0	81	0
		HAM	80	159	0	159	0	81	0	81	0
		EXC	80	159	2	159	2	81	1	81	1
		RUN	80	159	2	159	2	81	0	81	0
SMO-GP, with k=1+Pois(1)	init$_0$	INV	80	183	8	183	8	89	2	89	2
		LAS	80	159	2	159	2	81	0	81	0
		HAM	80	159	2	159	2	81	0	81	0
		EXC	80	161	2	161	2	81	1	81	1
		RUN	80	161	2	161	2	81	0	81	0
	init$_n$	INV	80	185	10	185	10	89	3	89	3
		LAS	80	159	2	159	2	81	0	81	0
		HAM	80	159	2	159	2	81	0	81	0
		EXC	80	161	0	161	0	81.5	1	81.5	1
		RUN	80	161	2	161	2	81	0	81	0

Table 3.6: Maximum tree sizes and maximum population sizes encountered for SMO-GP on the multi-objective problem variants: (1) until the individual X_{opt} with maximum fitness is found, (2) until the population represents the entire true Pareto front P_{Pareto}. Shown are median m and interquartile ranges iqr.

increase of about 6% in single operation mode and about $30 - 37\%$ in multi-operation mode. The interquartile range in this data is minimal, often zero in single operation mode and up to 5% in multi-operation mode. It is worth observing that the maximum attained tree size is quite independent on the used initialization scheme.

Population size

While T_{max} already appears as a factor in the computational complexity bounds for the (1+1)-GP variants, the impact of P_{max} on SMO-GP is not yet completely clear. It

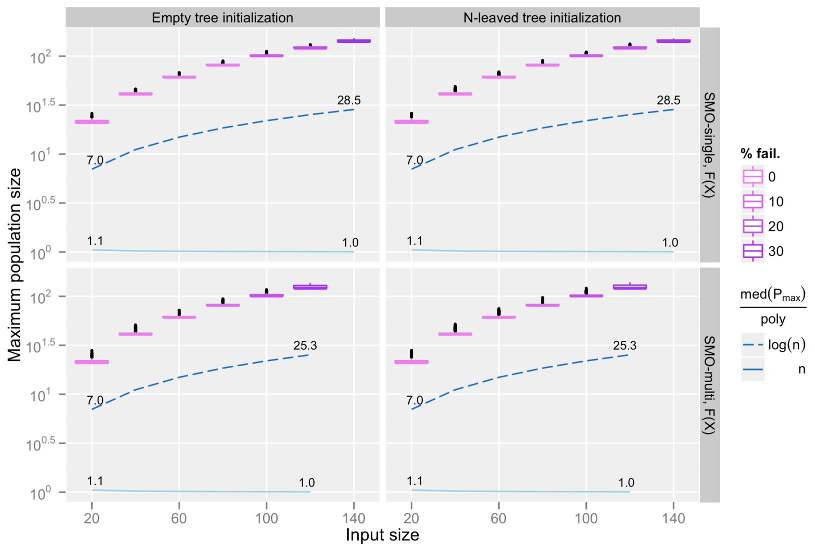


Figure 3.5: Maximum population size for INV in SMO-GP, possibly quadratic but practically linear in n.

Line style	meaning
solid	n^3
long dashed	$n^3 \log n$
dashed	n^4
dot dashed	n^5

Table 3.7: Meaning of line styles in Figure 3.6.

is reasonable to presume that for large populations, e.g. exponential in n, the expected optimisation time grows due to the lower probability of selecting the correct individual to improve. Unfortunately it is not obvious how often such a large population occurs since it depends on factors such as the number of different objectives and the fitness levels for each of these objectives.

For the sorting problem, four out of five of the considered sortedness measures yield a linear number of trade-offs, hence population individuals, between fitness value and complexity. Only one of the fitness functions, namely INV, can potentially generate a quadratic number of trade-offs. However, our experiments show that even in the case of INV the maximum population size is mostly *about* n and always linear in n (see Figure 3.5 where the population size has been divided by $\log n$ and n). As for the maximum tree size, there is no evident correlation between the choice of a particular initialization scheme and the maximum population size.

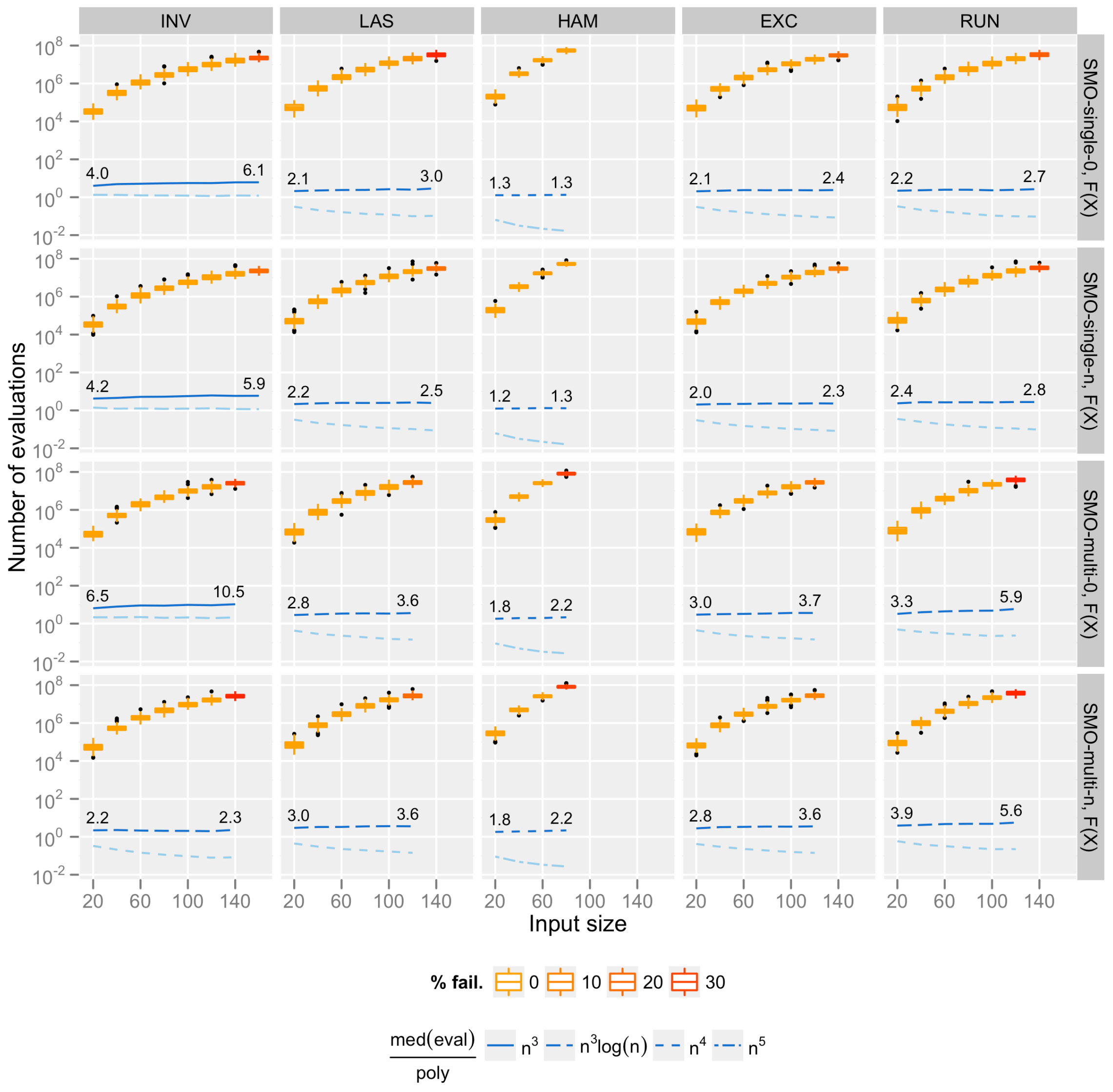


Figure 3.6: Shown as box plots is the number of evaluations required until the first individual with optimal fitness is found. Note that this multi-objective approach is "more reliable" (albeit slower) in solving the problem than the (1+1)-GP setups of Figures 3.3 and 3.4.

Average case optimisation time

Just as in the previous section, Figure 3.6 shows the distribution of the expected optimisation time for SMO-GP. Observe that for multi-objective algorithms the expected optimisation time is the number of evaluations to reach the true Pareto front, however, since for our experiments these two measure almost always coincided, we decided to drop the latter and promote the comparison between (1+1)-GP variants and SMO-GP variants. Table 3.7 summarises the meaning of the various line styles employed in the plot.

As can be seen from the plots, the theoretical bounds on HAM, EXC and RUN are always verified, this suggesting they are tight. As for INV and LAS, the polynomial lines show a strong indication towards a runtime in $\Omega(n^3 \log n)$, mostly close to

$O(n^3 \log n)$. Overall, when n is large, the single operation mode seems to yield better factors for the polynomials and a lower failure rate with respect to multi-operation mode.

3.5 Conclusions

Evolutionary algorithms with variable-length representations are frequently used, and the most prominent example using such a representation is genetic programming. Through our investigations, we contribute to their theoretical understanding. We discussed two methods for dealing with bloat that frequently occur when using such a representation. In order to point out the differences between these two approaches, we examined different measures of sortedness that have been analysed for evolutionary algorithms with fixed length representations. Interestingly, our analysis for the parsimony approach shows that variable-length representations might have difficulties when dealing with simple measures of sortedness due to the presence of local optima. Contrary to this, our runtime analysis for simple multi-objective algorithms shows that they compute the whole Pareto front for all examined sortedness measures in expected polynomial time.

Additionally, we carried out experimental investigations to complement the theoretical results. Crucial parameters in these theoretical analyses are the maximal solution size that is attained during the run of the algorithms, as well as the population size when dealing with multi-objective models. Furthermore, just a few theoretical results for the multi-operation variants are known to date, and for the existing bounds, it is also unknown how tight the given bounds are.

The analysis of our empirical investigations allowed us to fill in the gaps in the theory with conjectures about the average case complexities of these algorithms (see Tables 3.8 and 3.9):

- **(1+1)-GP, F(X):** When no bloat-control is applied, the algorithm fails regularly to solve RUN. INV and LAS appear to be solvable in $O(n^2 \log n)$, while EXC and HAM are solvable in $O(n^4)$.
- **(1+1)-GP*, F(X):** This situation changes quite dramatically for the worse, when introducing the minimal bloat-control mechanism of accepting new solutions only if they are of better fitness. INV is solved in $O(n^4)$ (as theory predicted), assuming a maximum tree size $T_{max} = O(n)$ (see Table 3.4). All other sortnedness measures are unsuccessful, when the initial tree already has n leaves. When initialising with the empty tree, the single-mutation variant achieves a runtime of $O(n^2 \log n)$ on LAS, EXC and RUN, and a runtime of $O(n^3)$ on HAM. These runtimes are in fact easy to prove, as the permutations can be build up one element at a time.

F(X)	(1+1)-GP*, F(X)		(1+1)-GP, F(X)	
	single	multi	single	multi
INV	$O(n^3 T_{max})$	$O(n^3 T_{max})$	$O(n \log n T_{max})$ †	$O(n \log n T_{max})$ †
LAS	∞	$\Omega\left(\left(\frac{n}{e}\right)^n\right)$	$O(n \log n T_{max})$ †	$O(n \log n T_{max})$ †
HAM	∞	$\Omega\left(\left(\frac{n}{e}\right)^n\right)$	$O(n^3 T_{max})$ †	$O(n^3 T_{max})$ †
EXC	∞	$\Omega\left(\left(\frac{n}{e}\right)^n\right)$	$O(n^3 T_{max})$ †	$O(n^3 T_{max})$ †
RUN	∞	$\Omega\left(\left(\frac{n}{e}\right)^n\right)$	$\Omega\left(\left(\frac{n}{e}\right)^n\right)$ †	$\Omega\left(\left(\frac{n}{e}\right)^n\right)$ †

Table 3.8: Single-objective problems: summary of our average case conjectures (†) and the proven bounds from Table 3.2.

- **(1+1)-GP, MO-F(X):** Here, the combination of applying just a single mutation at a time and initializing with the empty tree is the most successful one. When intialised with trees with $C(X) = 2n - 1$, then the algorithm has some chance to get stuck in a local optimum on MO-HAM, but still achieves an upper bound of $O(n^3)$ in the average case. As proven: MO-LAS is solved in $O(n^2 \log n)$, which confirms the proven upper bound.

- **SMO-GP, MO-F(X):** All problems are solved in $O(n^3 \log n)$, except for MO-HAM which is solved on average in $O(n^4)$, thus confirming the proofs. Regarding the missing proofs, it is now easy to show the $O(n^3 \log n)$ for MO-INV, assuming that the maximum population size $P_{max} = O(n)$, as supported by Figure 3.5.

Note that our results are based on an initial tree size, i.e. T_{init}, which is always linear in n, and thus the T_{init} term suggested by theoretical results is always dominated by the $O(n \log n)$ term. Nevertheless, it is easy to show that by using arbitrarily large initial tree sizes it is possible to obtain expected optimisation times in which the T_{init} term is relevant.

This work has been published in the book chapter of the post-conference book Genetic Programming Theory and Praxis IX in 2011 [65], and in the proceedings of the 12th International Conference on Parallel Problem Solving From Nature (PPSN) in 2012 [87].

F(X)	(1+1)-GP, MO-F(X)		SMO-GP, MO-F(X)
	single	multi	single/multi
INV	$O(T_{init} + n^5)$, $O(nT_{init} + n^3)$ †	$O(nT_{init} + n^3)$ †	$O\left(n^2 T_{init} + n^5\right)$, $O(nT_{init} + n^3 \log n)$ †
LAS	$O(T_{init} + n^2 \log n)$	$O(T_{init} + n^2 \log n)$	$O(nT_{init} + n^3 \log n)$ †
HAM	∞, $O(n^3)$ †	$O(n^3)$ †	$O(nT_{init} + n^4)$
EXC	∞	$O(nT_{init} + n^5)$ ‡	$O(nT_{init} + n^3 \log n)$
RUN	∞	$\Omega\left(\left(\frac{n}{e}\right)^n\right)$ †	$O(nT_{init} + n^3 \log n)$

Table 3.9: Multi-objective problems: summary of our average case conjectures (†) and the proven bounds from Table 3.3. ‡ is a conjecture based on the idea that a single *exchange* operation can be simulated with HVL-Prime in time $O(n^4)$.

Experience without theory is blind, but theory without experience is mere intellectual play.

Immanuel Kant

4 Experimental Analysis of ORDER and MAJORITY

In contrast to the previous chapter, where the focus was on theoretical investigations, we now carry out experimental investigations that complement recent theoretical investigations [29, 62]. Crucial measures in these theoretical analyses are the maximum tree size that is attained during the run of the algorithms as well as the population size when dealing with multi-objective models. Both measures can be very difficult to bound in a theoretical analysis. Therefore, we study those measures in detail by extensive experimental investigations and analyse the runtime of the different algorithms in a purely experimental way.

4.1 Introduction

The algorithms that we consider in the following are the stochastic hill-climber called (1+1)-GP (see Figure 2.6) and the population-based multi-objective programming algorithm called SMO-GP (see Figure 2.7). These algorithms have been analysed on problems with isolated program semantics taken from [39], which can be seen as the analogue of linear pseudo-Boolean functions [27] known from the computational complexity analysis of evolutionary algorithms working with fixed length binary representations.

The theoretical results provided in [29, 62] bring up several questions that remain unanswered in these papers. In particular, for different combinations of algorithms and problems no (or no exact) runtime bounds are given. In our paper, we explore the different open cases and questions in an experimental way. Similar to [8, 53], this may guide further rigorous analyses by exploring the important measures within a computational complexity analysis of the algorithms and give experimental estimates on the actual runtime of the algorithms on the different problems. Our experimental investigations, will concentrate on important measures such as the maximum tree size during the run of the single-objective algorithms analysed in [29] and the maximum population size of the multi-objective algorithm analysed in [62]. It can be observed from the analyses carried out in these two papers, that both measures have a different implication on the runtime of the analysed genetic programming algorithms. Other experimental results indicate that both measures do not grow large during the run of the algorithms, which would imply a fast optimisation process. Furthermore, our experimental results on the actual runtime of (1+1)-GP and SMO-GP indicate an efficient optimisation process.

This chapter is structured as follows. In Section 4.2, we summarise the computational complexity results from [29, 62]. (1+1)-GP is experimentally investigated in Section 4.3 and the behaviour of SMO-GP is examined in Section 4.4. We finish with some concluding remarks.

4.2 Preliminaries

In our experimental investigations, we will treat the algorithms and problems analysed in [29, 62]. The setup is similar to that of our investigations on SORTING (see Chapter 3). For the (1+1)-GP, we consider the problem of computing a solution X that maximises a given function $F(X)$. In the case of the parsimony approach, we additionally take into account the complexity $C(X)$ of a solution (measured as the total number of nodes in the tree). For SMO-GP, we will treat the two objectives F and C as equally important and use standard notations from the field of multi-objective optimisation (see Section 2.3). As before, (1+1)-GP and SMO-GP only use the mutation operator HVL-Prime (see Figure 2.4) to generate offspring. Lastly, depending on the number of operations used in the mutation operator, we get the algorithms (1+1)-GP-single and SMO-GP-single and their corresponding multi-mutation variants (1+1)-GP-multi and SMO-GP-multi.

4.2.1 Theoretical results

Let us recall that the computational complexity analysis of genetic programming analyses the expected number of fitness evaluations until an algorithm has produced an

F(X)	(1+1)-GP, F(X) [29]		(1+1)-GP, MO-F(X) [62]		SMO-GP, MO-F(X) [62]	
	k=1	k=1+Pois(1)	k=1	k=1+Pois(1)	k=1	k=1+Pois(1)
ORDER	$O(nT_{max})$	$O(nT_{max})$	$O(T_{init} + n \log n)$	?	$O(nT_{init} + n^2 \log n)$	
WORDER	?	?	$O(T_{init} + n \log n)$		$O(n^3)\star$	?
MAJORITY	$O(n^2 T_{max} \log n)$	?	$O(T_{init} + n \log n)$		$O(nT_{init} + n^2 \log n)$	
WMAJORITY	?	?	$O(T_{init} + n \log n)$		$O(n^3)\star$	?

Table 4.1: Computational complexity results from [29, 62]. Question marks indicate combinations for which we do not know any bounds.

optimal solution for the first time. This is called the *expected optimisation time*. In the case of multi-objective optimisation the number of fitness evaluations until the whole Pareto front has been computed is analysed and referred to as the expected optimisation time. The bounds from [29, 62] are listed in Table 4.1. As it can be seen, all results take into account tree sizes of some kind: either the maximum tree size T_{max} during the search plays a role in the bound, or the size of the initial tree T_{init} does. Furthermore, it is quite striking that virtually no results for the multi-operation variants are known to date. It is also unknown how tight the given bounds are. The maximum tree size for (1+1)-GP and the population size for SMO-GP play a relevant role in the theoretical analysis and will be further investigated in the rest of the paper. Lastly, note that the upper bounds marked with $\star$ hold only if the algorithm has been initialised in the particular, i.e. non-redundant, way described in [62].

4.2.2 Experimental setup

In the remainder of this chapter, we will empirically confirm and verify the theoretical results from [29, 62]. Additionally, by analysing the data gathered from the experiments, we will estimate asymptotic runtimes where theoretical results are missing. We consider (1+1)-GP and SMO-GP, each in their single and multi-operation variants, and investigate problems of sizes $n = 20, 40, 60, \ldots, 200$. For the initialisation, we consider the schemes init$_0$ (empty tree) and init$_{2n}$ (in which a $2n$ leaves tree is generated by applying $2n$ *insertion* mutations at random positions). In total, our experiments span twelve problems: WORDER and WMAJORITY (see Section 2.2 in their F(X) and MO-F(X) variants. The weight settings are set as follows:

- ORDER, MAJORITY: $w_i = 1$, $1 \leq i \leq n$
- WORDER-RAN, WMAJORITY-RAN: $w_i \in [0, 1]$ chosen uniformly at random, $1 \leq i \leq n$
- WORDER-BIN, WMAJORITY-BIN: $w_i = 2^{n-i}$, $1 \leq i \leq n$

The following experiments were performed on AMD Opteron 250 CPUs (2.4GHz), on Debian GNU/Linux 5.0.8, with Java SE RE 1.6 and were given a maximum runtime of 3

k	F(X)	n	(1+1)-GP, F(X)				(1+1)-GP, MO-F(X)			
			init$_0$		init$_{2n}$		init$_0$		init$_{2n}$	
			m	iqr	m	iqr	m	iqr	m	iqr
k=1	ORDER	100	519	94.5	593	100	199	0	399	2
	WORDER-RAN	100	513	85	594	90	199	0	399	0.5
	WORDER-BIN	100	513	94	591	88.5	199	0	399	0
	MAJORITY	100	507	78.5	563	72	199	0	399	0
	WMAJORITY-RAN	100	499	76.5	567	74.5	199	0	399	0
	WMAJORITY-BIN	100	499	74.5	567	75	199	0	399	0
k=1+Pois(1)	ORDER	100	670	138	742	143	223	12	399	6
	WORDER-RAN	100	667	136.5	713	131	229	12	399	6
	WORDER-BIN	100	665	150.5	735	132.5	231	12	399	4
	MAJORITY	100	624	96	668	102	239	14	401	8
	WMAJORITY-RAN	100	617	104	678	116.5	241	16	401	8
	WMAJORITY-BIN	100	635	114.5	671	116.5	243	14	401	8

Table 4.2: Maximum tree sizes encountered until the individual X_{max} with maximum fitness is found. Shown are median m and median interquartile ranges iqr. Here $k = 1$ and $k = 1 + Pois(1)$ refer respectively to the single and multi-operation variants.

hours and a budget of 10^9 evaluations. Furthermore, each experiment has been repeated 400 times, which results in a standard error of the mean (the standard deviation of the sampling distribution) of $1/\sqrt{400} = 5\%$. The source code of the framework is available online (see the URLs in Section 3.4.2).

4.3 (1+1)-GP

4.3.1 Tree size

The theoretical bounds for (1+1)-GP on ORDER and MAJORITY presented in [29] depend on the maximum tree size that is encountered during the run of the algorithms. We investigate the maximum tree size experimentally in order to see whether bloat occurs when applying the algorithms. For (1+1)-GP-single using the parsimony approach, i. e. using the function MO-F(X), the difference between the solution value S and the number of leaves not preceded by their complements can not increase during the run of the algorithm [62].

First, we investigate the tree sizes typically observed during the optimisation for the different (1+1)-GP algorithms. Table 4.2 reports results for $n = 100$, but similar results hold for the other input sizes. The maximum tree size observed during the run of (1+1)-GP on MO-F(X) when using single-operation and empty initialisation is

$2n-1$, which is the minimum possible size of an optimal solution. This was expected, since the algorithm can only increase the tree by a single leaf in every accepting step. These values increase by about 10-20% in the case of init_0, when multiple HVL-Prime applications are allowed per mutation step. When the acceptance criteria is weakened by switching to the F(X) variant (i.e. the current tree can be replaced by larger ones of identical fitness), then the tree sizes are about 2.5 times larger in the single-operation case, and about 3 times larger in the multi-operation case.

Similarly, when running (1+1)-GP on MO-F(X), if the population is initialised with trees of $2n$ leaves, the largest trees encountered are of size $2 \cdot (2n) - 1$, i.e. the tree size of the initial solution, in the single-operation case, and are just minimally larger (about 1%) in the multi-operation case.

For the non-parsimony variants, however, the largest trees are about 50% larger when solving ORDER, and almost 100% when solving MAJORITY.

4.3.2 Runtime

Figure 4.1 shows the distributions of the required evaluations for the (1+1)-GP variants as box plots. The line plots represent the medians divided by different polynomials and suggest the asymptotic behavior of the algorithms: the solid line is the median number of evaluations needed to produce the individual with the optimal fitness value divided by $n \log n$, and the dashed line is the same number, but divided by n^2.

For all combinations of algorithms and problems these plots indicate an expected optimisation time of $O(n \log n)$, as the solid lines closely resemble constant functions (see the y-values for $n = 20$ and $n = 200$), and the y-values of the dashed lines are decreasing with increasing values of n. The constant factor obtained by dividing the median number of evaluations by $n \log n$ is overall higher in the single-operation variants of the algorithm, suggesting that applying multi mutations can help getting earlier to the optimal solution.

One important observation is that the algorithms' asymptotic behaviour appears to be same, when initialised with the empty tree, and with trees with $2n$ leaves. For the setups where a theoretical bound in Table 4.1 is missing, the experimental results give a strong indication about the expected optimisation time being $O(n \log n)$.

4.4 SMO-GP

4.4.1 Tree size and population size

Table 4.3 shows the maximum tree sizes and maximum population sizes that were observed up to the following two events. Firstly, until the individual X_{max} with maximum

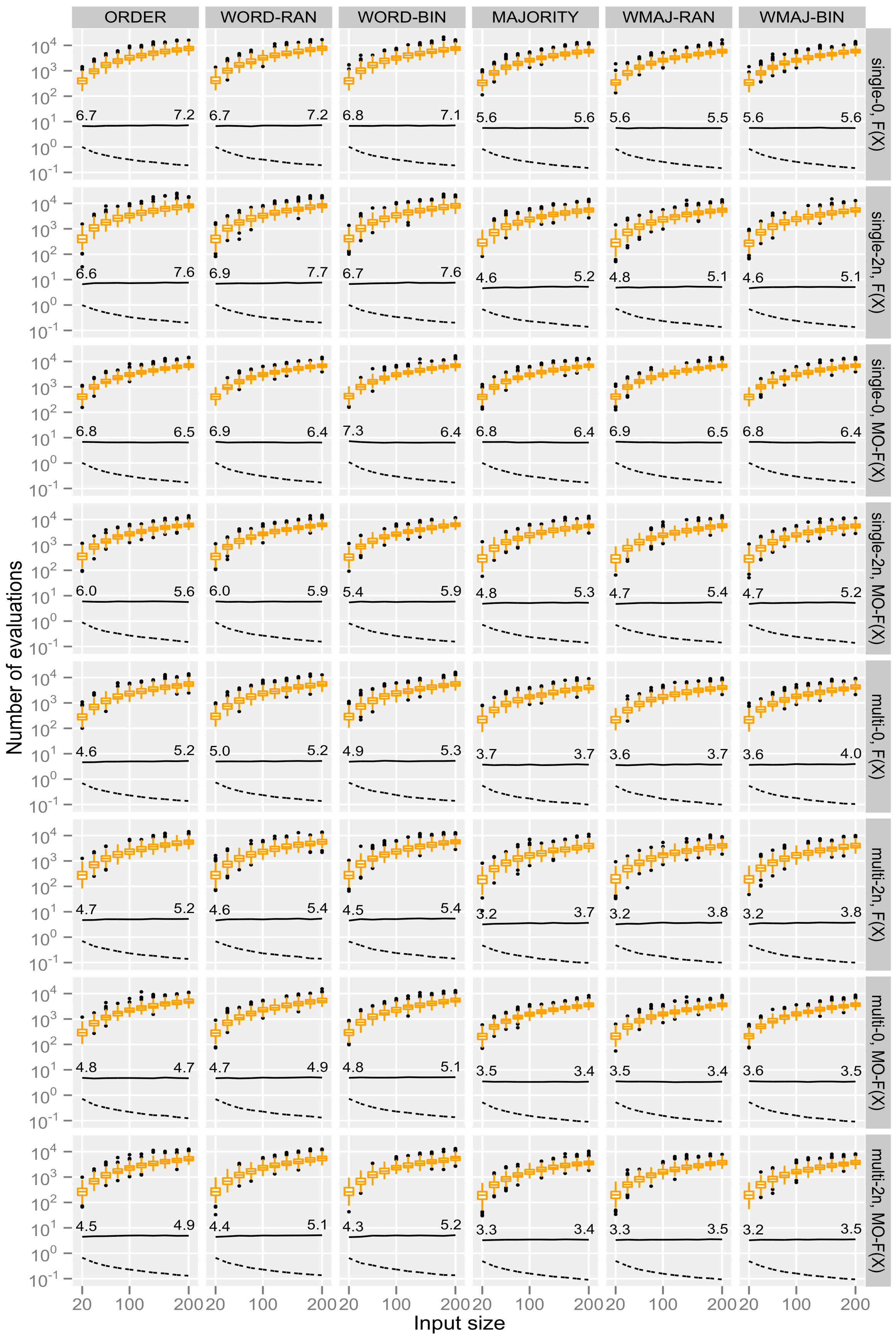


Figure 4.1: Number of evaluations required by (1+1)-GP until the individual X_{max} with maximum fitness is found, shown as box plots. The solid line is the median of the number of evaluations divided by $n \log n$, the dashed line is the same median divided by n^2.

		F(X)	n	maximum tree size				max. population size			
				to X_{max}		to P_{Pareto}		to X_{max}		to P_{Pareto}	
				m	iqr	m	iqr	m	iqr	m	iqr
SMO-GP, with k=1	init_0	ORDER	100	199	0	199	0	101	0	101	0
		WORDER-RAN	100	199	0	199	0	101	0	101	0
		WORDER-BIN	100	199	0	199	0	101	0	101	0
		MAJORITY	100	199	0	199	0	101	0	101	0
		WMAJORITY-RAN	100	199	0	199	0	101	0	101	0
		WMAJORITY-BIN	100	199	0	199	0	101	0	101	0
	init_{2n}	ORDER	100	399	0	399	0	101	0	101	0
		WORDER-RAN	100	399	0	399	0	101	0	101	0
		WORDER-BIN	100	399	0	399	0	101	0	101	0
		MAJORITY	100	399	0	399	0	101	0	101	0
		WMAJORITY-RAN	100	399	0	399	0	101	0	101	0
		WMAJORITY-BIN	100	399	0	399	0	101	0	101	0
SMO-GP, with k=1+Pois(1)	init_0	ORDER	100	207	6.5	207	6.5	101	0	101	0
		WORDER-RAN	100	211	8	211	8	102	2	102	1
		WORDER-BIN	100	211	6	211	6	102	1	102	2
		MAJORITY	100	215	10.5	215	10.5	101	0	101	0
		WMAJORITY-RAN	100	223	12	223	12	103	2	103	1
		WMAJORITY-BIN	100	219	10	219	10	102	2	103	2
	init_{2n}	ORDER	100	399	4	399	4	101	0	101	0
		WORDER-RAN	100	399	4	399	4	102	2	102	1
		WORDER-BIN	100	399	4	399	4	102	1	102	2
		MAJORITY	100	399	4	399	4	101	0	101	0
		WMAJORITY-RAN	100	400	6	400	6	103	2	104	2
		WMAJORITY-BIN	100	401	6	401	6	102	2	103	1

Table 4.3: Maximum tree sizes and maximum population sizes encountered for SMO-GP on the MO-F(X) problem variants: (1) until the individual X_{max} with maximum fitness is found, (2) until the population represents the entire true Pareto front P_{Pareto}. Shown are median m and interquartile ranges iqr. init_0 denotes the initialisation with the empty tree, and init_{2n} the one with randomly constructed trees with $2n$ leaf nodes.

fitness is found, and secondly, until the population represents the entire true Pareto front P_{Pareto}.

It can be seen that tree and population sizes observed by SMO-GP-single are independent of the initialisation. In all cases, no trees with more than the size of the Pareto optimal solution X with $F(X) = n$ (size $2n - 1$) (when using init_0) and the initial tree size $2 \cdot (2n) - 1$ (when using init_{2n}) ever belong to the population. In the multi-operation cases, the maximum population sizes are rarely higher, and the same holds for the maximum tree sizes.

4.4.2 Runtime

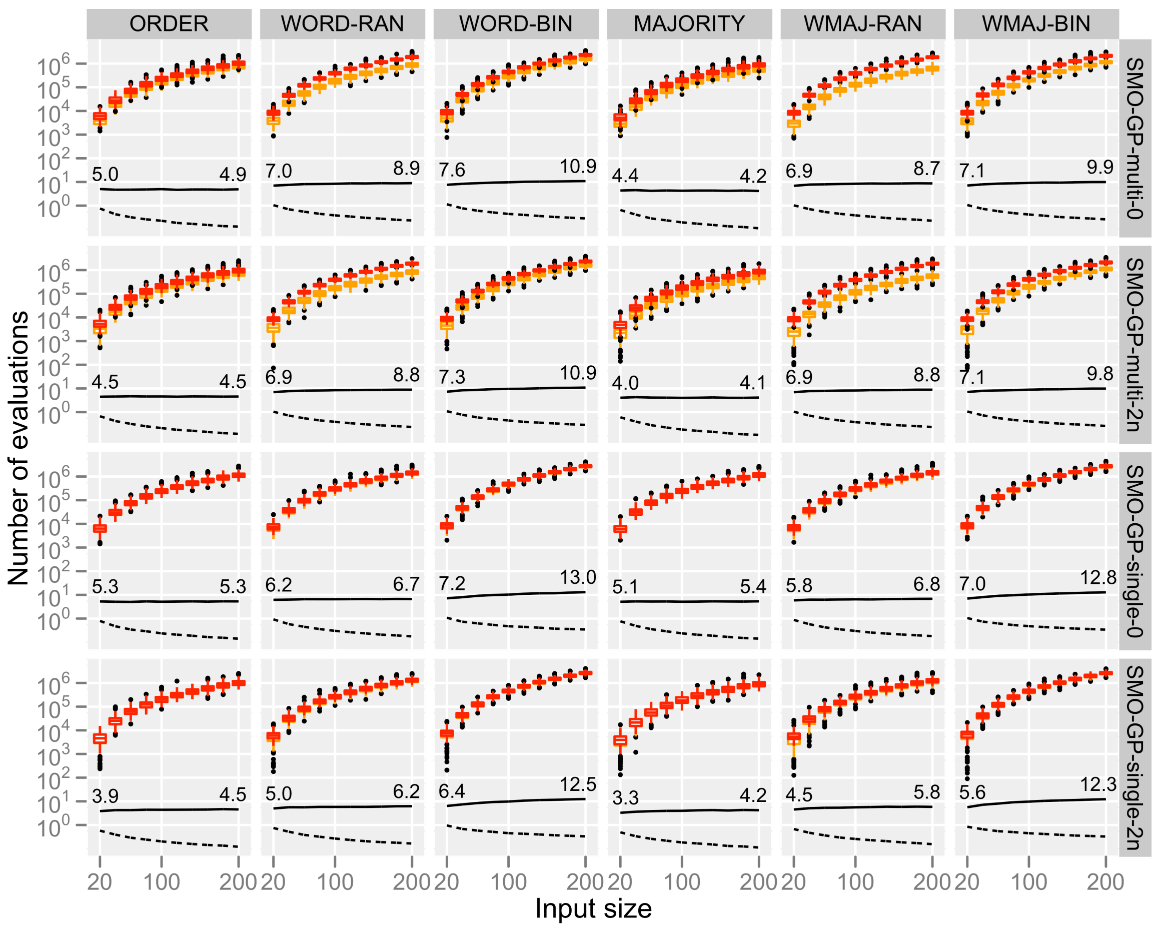


Figure 4.2: Shown as box plots is the number of evaluations required: (1) until the individual X_{max} with maximum fitness is found (orange), (2) until the population represents the entire true Pareto front P_{Pareto} (red). The solid line is the median of the latter number of evaluations divided by $n^2 \log n$, the dashed line is it divided by n^3.

Just as in the previous section, we show now the distributions of the required evaluations as box plots in Figure 4.2. As before, yellow box plots represent the number of evaluations to get to X_{max}, while red box plots represent now the number of evaluations to get to P_{Pareto}. In this plot, the lines are the medians divided by different polynomials and suggest the asymptotic behaviour of the algorithms: the solid line is the median number of evaluations needed to get to the Pareto front divided by $n^2 \log n$, and the dashed line is the same number, but divided by n^3. For all combinations of algorithms and the problems, these plots indicate an expected optimisation time of $O(n^2 \log n)$ for ORDER and MAJORITY, as the solid lines closely resemble constant functions, and the y-values of the dashed lines are decreasing with increasing values of n. For the weighted variants, however, the solid lines appear to be slowly rising, indicating a runtime in $\Omega(n^2 \log n) \cap O(n^3)$, although the runtime is extremely close to $O(n^2 \log n)$.

Furthermore, it can be observed that there is a significant time difference, for SMO-GP-multi, between finding the individual with the optimal fitness value and finding the entire Pareto front. For SMO-GP-single, this time difference is negligible, which is the reason why the corresponding orange box plots are scarcely identifiable behind the red ones.

4.5 Conclusions

In this chapter, we carried out experimental investigations to complement recent theoretical results on the runtime of two genetic programming algorithms [29, 62]. Crucial measures in these theoretical analyses are the maximum tree size that is attained during the run of the algorithms, as well as the population size when dealing with multi-objective models. Furthermore, virtually no theoretical results for the multi-operation variants are known to date. It is also unknown how tight the given bounds are. The analysis of our empirical investigations allowed us to fill in the gaps in the theory with conjectures about the expected optimisation time (see Tables 4.4 and 4.5) of these algorithms.

Our experimental evaluation shows that the expected optimisation time of (1+1)-GP on the single-objective F(X) is very close to $O(n \log n)$. Our results, however, are based on an initial tree size, i.e. T_{init}, which is always linear in n, and thus the T_{init} term suggested by theoretical results is always dominated by the $O(n \log n)$ term. Nevertheless, it is easy to show that by using arbitrarily large initial tree sizes it is possible to obtain expected optimisation times in which the T_{init} term is relevant. For this reason we conjecture an expected optimisation time of $O(T_{init} + n \log n)$. Following the same reasoning for SMO-GP, we conjecture a runtime of $O\left(nT_{init} + n^2 \log n\right)$ by noting that the observed runtimes are very close to $O(n^2 \log n)$ and that the algorithm has to evolve a population of size $O(n)$.

As a further development for this line of research, it would be interesting to prove these conjectured bounds theoretically and to show how they are related to maximum population size reached during an optimisation run.

This work has been published in the proceedings of the 12th International Conference on Parallel Problem Solving From Nature (PPSN) in 2012 [84].

F(X)	(1+1)-GP, F(X)		(1+1)-GP, MO-F(X)	
	k=1	k=1+Pois(1)	k=1	k=1+Pois(1)
ORDER	$O(nT_{max})$ [29] $O(T_{init} + n \log n)$ †	$O(nT_{max})$ [29] $O(T_{init} + n \log n)$ †	$O(T_{init} + n \log n)$[62]	$O(T_{init} + n \log n)$ †
WORDER	$O(T_{init} + n \log n)$ †	$O(T_{init} + n \log n)$ †	$O(T_{init} + n \log n)$[62]	$O(T_{init} + n \log n)$ †
MAJORITY	$O(n^2 T_{max} \log n)$ [29] $O(T_{init} + n \log n)$ †	$O(T_{init} + n \log n)$ †	$O(T_{init} + n \log n)$[62]	$O(T_{init} + n \log n)$ †
WMAJORITY	$O(T_{init} + n \log n)$ †	$O(T_{init} + n \log n)$ †	$O(T_{init} + n \log n)$[62]	$O(T_{init} + n \log n)$ †

Table 4.4: Summary of our conjectures (†) and the existing upper bounds from Table 4.1.

F(X)	SMO-GP, MO-F(X)	
	k=1	k=1+Pois(1)
ORDER	$O(nT_{init} + n^2 \log n)$[62]	$O(nT_{init} + n^2 \log n)$[62]
WORDER	$O(n^3)\star$ [62] $O(nT_{init} + n^2 \log n)$ †	$O(nT_{init} + n^2 \log n)$ †
MAJORITY	$O(nT_{init} + n^2 \log n)$[62]	$O(nT_{init} + n^2 \log n)$[62]
WMAJORITY	$O(n^3)\star$ [62] $O(nT_{init} + n^2 \log n)$ †	$O(nT_{init} + n^2 \log n)$ †

Table 4.5: Summary of our conjectures (†) and the existing upper bounds from Table 4.1.

Part II

Design of Evolutionary Multi-Objective Algorithms

The majority see the obstacles; the few see the objectives; history records the successes of the latter, while oblivion is the reward of the former.

Alfred A. Montapert

5 Evolutionary Multi-Objective Optimisation

Multi-objective optimisation problems arise frequently in applications. For example, let us consider the scenario of producing energy with the help of wind turbines. There, in the planning phase of the wind farm, several properties have to be considered. In Figure 5.1 we indicate the trade-offs between the occurring wake effects (which are detrimental to the energy production) and the area dimensions (which incur varying initial set-up costs): smaller areas cost less initially, but the wake effects on the other hand can significantly reduce the efficiency of the wind farm. In order to eventually make an informed decision for a particular wind farm configuration, the decision maker should be aware of different possible trade-offs. For more real-world examples, we refer the interested reader to the surveys [15, 57, 59, 83].

As we see in the example of the wind farm planning, the objectives of real-world problems can often be in conflict with each other. The goal of solving a multi-objective optimisation (MOO) problem is to find a (not too large) set of compromise solutions. The Pareto front of a MOO problem consists of the function values representing the different trade-offs with respect to the given objective functions. In practice, it is impossible to compute the whole Pareto front, and MOO problems can often only be solved approximately by heuristic approaches. Evolutionary algorithms have been widely used to tackle multi-objective problems. These algorithms use different measures to ensure diversity in the objective space but are not guided by a formal notion of approximation.

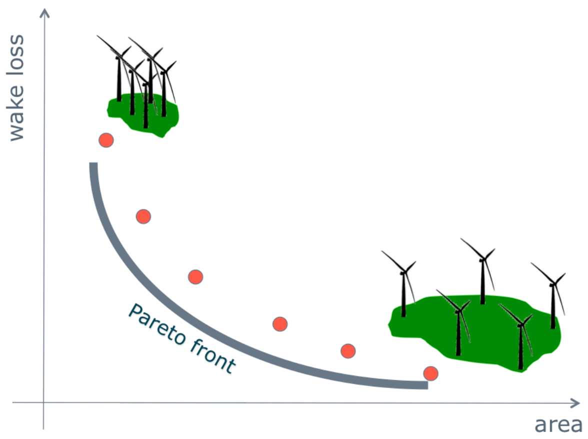


Figure 5.1: Sketch: search space for the multi-objective optimisation of wind farms with respect to "wake effects vs. area requirements". The red dots represent actual layouts with different trade-offs.

In this part of the thesis, we present a new framework of an evolutionary algorithm for multi-objective optimisation that allows to work with a formal notion of approximation. Our experimental results show that our approach outperforms state-of-the-art evolutionary algorithms in terms of the quality of the approximation that is obtained in particular for problems with many objectives.

5.1 Introduction

Multi-objective optimisation is assumed to be more (or at least as) difficult as single-objective optimisation due to the task of computing several solutions. From a computational complexity point of view even simple single-objective problems on weighted graphs like shortest paths or minimum spanning trees become NP-hard when they encounter at least two weight functions [30]. In addition, the size of the Pareto front is often exponential for discrete problems and even infinite for continuous ones.

Due to the hardness of almost all interesting multi-objective problems, different heuristic approaches have been used to tackle them. Among these methods evolutionary algorithms are frequently used as they work at each time step with a set of solutions called population. The population of an evolutionary algorithm for a MOO is used to store desired trade-off solutions for the given problem.

As the size of the Pareto front is often very large, evolutionary algorithms and all other algorithms for MOO have to restrict themselves to a smaller set of solutions. This set of solutions should be a good approximation of the Pareto front. The main question is now how to define approximation. The literature (see, e. g., [19]) on evolutionary multi-objective optimisation (EMO) just states that the set of compromise solutions

- should be close to the true Pareto front,
- should cover the complete Pareto front, and

- should be uniformly distributed.

Many approaches try to produce good approximations of the true Pareto front by incorporating different preferences. For example, the environmental selection in the Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [20] first ranks the individuals using non-dominated sorting. Then, in order to distinguish individuals with the same rank, the crowding distance metric is used, which prefers individuals from less crowded sections of the objective space. The metric value for each solution is computed by adding the edge lengths of the cuboids in which the solutions reside, bounded by the nearest neighbours.

The Strength Pareto Evolutionary Algorithm 2 (SPEA2) [98] works similarly. The raw fitness of the individuals according to Pareto dominance relations between them is calculated, and then a density measure to break the ties is used. The individuals that reside close together in the objective space are less likely to enter the archive of best solutions.

In contrast to these two algorithms, the Indicator-Based Evolutionary Algorithm (IBEA) [96] is a general framework, which uses no explicit diversity preserving mechanism. The fitness of individuals is determined solely based on the value of a predefined indicator. Typically, implementations of IBEA come with the epsilon indicator or the hypervolume indicator, where the latter measures the volume of the dominated portion of the objective space.

The S-metric Selection Evolutionary Multi-Objective Algorithm (SMS-EMOA) [32] is a frequently used IBEA, which uses the hypervolume indicator directly in the search process. It is a steady-state algorithm that uses non-dominated sorting as a ranking criterion, and the hypervolume as the selection criterion to discard that individual, which contributes the least hypervolume to the worst-ranked front. While SMS-EMOA often outperforms its competition, its runtime unfortunately increases exponentially with the number of objectives. Nevertheless, with the use of fast approximation algorithms (e.g., [5, 10, 48]), this algorithm can be applied to solve problems with many objectives as well.

However, the above notion of approximation is not a formal definition. Having no formal definition of approximation makes it hard to evaluate and compare algorithms for MOO problems. Therefore, we think that it is necessary to use a formal definition of approximation in this context and evaluate algorithms with respect to this definition.

Different formal notions of approximation have been used to evaluate the quality of algorithms for multi-objective problems from a theoretical point of view. The most common ones are the multiplicative and additive approximation (see [18, 22, 72]). Laumanns et al. [54] have incorporated this notion of approximation in an evolutionary algorithm for MOO. However, this algorithm is mainly of theoretical interest as the desired approximation is determined by a parameter of the algorithm and is not improved over time. Another approach related to a formal notion of approximation is the popular hypervolume indicator [97] that measures the volume of the dominated portion of the objective space. Hypervolume-based algorithms such as the Multi-Objective Covariance Matrix Adaptation Evolution Strategy (MO-CMA-ES) [47] or SMS-EMOA [7] are well-established for solving MOO problems. They do not use a formal notion of approximation but it has recently been shown that the worst-case approximation obtained by optimal hypervolume distributions is asymptotically equivalent to the best worst-case approximation achievable by all sets of the same size [11, 12]. The major drawback of the hypervolume approach is that it cannot be computed in time polynomial in the number of objectives unless $\mathrm{P} = \mathrm{NP}$ [9].

In this part of the thesis, we introduce an efficient framework of an evolutionary algorithm for MOO that works with a formal notion of approximation and improves the approximation quality during its iterative process. The algorithm can be applied to a wide range of notions of approximation that are formally defined. As the algorithm does not have complete knowledge about the true Pareto front, it uses the best knowledge obtained so far during the optimisation process.

The intuition for our framework is as follows. During the optimisation process, the current best set of compromise solutions (usually called "population") gets closer and closer to the Pareto front. Similarly, the set of all non-dominated points seen so far in the objective space (we call this "archive") is getting closer to the Pareto front. Additionally, the archive is getting larger and larger and becoming an increasingly good approximation of the true Pareto front. Assuming that the archive approximates the Pareto front, we then measure the quality of the population by its approximation with respect to the archive. In our algorithm

- any set of feasible solutions constitutes a (potentially bad) approximation of the true Pareto front, and

- we optimise the approximation with respect to all solutions seen so far.

We show that this approach is highly successful in obtaining approximations according to the formal definition. Comparing our results to state of the art approaches such as NSGA-II, SPEA2, and hypervolume based algorithms, we show that our algorithm gives significantly better approximations fulfilling the formal definition.

The outline of this part is as follows. We introduce some basic definitions in Section 5.2. In Chapter 6 we present the basic algorithm and first speed-up techniques. In Chapter 7 we present the improved version of the algorithm, which incorporates additional speed-up techniques.

5.2 Preliminaries

In multi-objective optimisation the task is to optimise a function $f = (f_1, \dots, f_d)\colon S \to \mathbb{R}^d_+$ with $d \geq 2$, which assigns to each element $s \in S$ a d-dimensional objective vector. Each objective function $f_i\colon S \mapsto \mathbb{R}$, $1 \leq i \leq d$, maps from the considered search space S into the positive real values. Elements from S are called search points and the corresponding elements $f(s)$ with $s \in S$ are called objective vectors.

Throughout this part of the thesis, we consider the minimisation of d objectives. As already said, the given objective functions f_i are usually conflicting in multi-objective optimisation, which implies that there no single optimal objective vector. Instead of this the Pareto dominance relation is defined, which is a partial order. In order to simplify the presentation we only work with the Pareto dominance relation on the objective space and mention that this relation transfers to the corresponding elements of S.

The Pareto dominance relation $\preceq$ between two objective vectors $x = (x_1, \dots, x_d)$ and $y = (y_1, \dots, y_d)$, with $x, y \in \mathbb{R}^d$ is defined as

$$x \preceq y \;:\Leftrightarrow\; x_i \leq y_i \text{ for all } 1 \leq i \leq d.$$

We say that x dominates y iff $x \preceq y$. If

$$x \prec y \;:\Leftrightarrow\; x \preceq y \text{ and } x \neq y$$

holds, we say that x strictly dominates y as x is not worse than y with respect to any objective, and at least better with respect to one of the d objectives.

The objective vectors x and y are called incomparable if

$$x \parallel y \;:\Leftrightarrow\; \neg(x \preceq y \vee y \preceq x)$$

holds. Two objective vectors are therefore incomparable if there are at least two (out of the d) objectives where they mutually beat each other. An objective vector x is called Pareto optimal if there is no $y = f(s)$ with $s \in S$ for which $y \prec x$ holds. The set of all Pareto optimal objective vectors is called the Pareto front of the problem given by f. Note that the Pareto front is a set of incomparable objective vectors.

Even for two objectives the Pareto front might grow exponentially with respect to the problem size. Therefore, algorithms for multi-objective optimisation usually have to restrict themselves to a smaller set of solutions. This smaller set is then the output of the algorithm.

Algorithm 5.1: Measure approximation quality of a population

```
input  : Archive $A$, Population $P$
output: Indicator $S_\alpha(A, P)$
1  $S \leftarrow \emptyset$;
2  foreach $a \in A$ do
3      $\delta \leftarrow \infty$;
4      foreach $p \in P$ do
5          $\rho \leftarrow -\infty$;
6          for $i \leftarrow 1$ to $d$ do
7              $\rho \leftarrow \max\{\rho, a_i - p_i\}$;
8          $\delta \leftarrow \min\{\delta, \rho\}$;
9      $S \leftarrow S \cup \{\delta\}$;
10 sort $S$ decreasingly;
11 return $S$;
```

We make the notion of approximation precise by considering a weaker relation on the objective vectors called additive ϵ-dominance. It is defined as

$$x \preceq_{\epsilon^+} y \;:\Leftrightarrow\; x_i + \epsilon \le y_i \text{ for all } 1 \le i \le d.$$

Furthermore, we also define additive approximation of a set of objective vectors T with respect to another set of objective vectors S.

In order to judge the quality of a population P with respect to a given archive A, we will use Definition 3. In this way, we have a measure on how good the current population is with respect to the search points seen during the run of the algorithm.

Definition 3. *For finite sets $S, T \subset \mathbb{R}^d$, the additive approximation of T with respect to S is defined as*

$$\alpha(S, T) := \max_{s \in S} \min_{t \in T} \max_{1 \le i \le d} (s_i - t_i).$$

Although, we are only using the notion of additive approximation, we would like to mention that our approaches can be easily adapted to multiplicative approximation (e.g., as defined in [72]). To do this, we only need to adjust the definitions accordingly.

Note that this indicator is sensitive to outliers. We prefer this over taking the average of the approximations: the resulting indicator would become very similar to the generational distance [85], and it would lose its motivation from theory.

Our aim is to minimise the additive approximation of the population P we output with respect to the archive A of all points seen so far, i.e., we want to minimise

$\alpha(A, P)$. The problem is that $\alpha(A, P)$ is not sensitive to local changes of P. $\alpha(A, P)$ only measures improvements of points that are currently worst approximated.

To get a sensitive indicator that can be used to guide the search, we consider instead the set $\{\alpha(\{a\}, P) \mid a \in A\}$ of all approximations of the points in A. We sort this set decreasingly and call the resulting sequence

$$S_\alpha(A, P) := (\alpha_1, \ldots, \alpha_{|A|}).$$

The first entry α_1 is again $\alpha(A, P)$. Our new goal it then to minimise $S_\alpha(A, P)$ *lexicographically.*[1] Note that this is an augmentation of the order induced by $\alpha(A, P)$: If we have $\alpha(A, P_1) < \alpha(A, P_2)$ then we also have $S_\alpha(A, P_1) <_{\text{lex}} S_\alpha(A, P_2)$. Moreover, this indicator is locally sensitive. Algorithm 5.1 describes how to calculate it.

[1]In lexicographic minimisation, the reduction of larger approximation values (placed towards the left in the sorted sequence) is regarded as "infinitely more important" than the reduction of smaller approximation values (placed towards the right in the sorted sequence).

An approximate answer to the right problem is worth a good deal more than an exact answer to an approximate problem.

John Tukey

6 Approximation-Guided Evolution

So far, we have defined different concepts around multi-objective optimisation, but we have not presented our algorithm yet. In this chapter, we will present the first algorithm called Approximation-Guided Evolution (AGE), which uses the approximation quality indicator $S_\alpha(A, P)$ (see Definition 3 and Algorithm 5.1).

6.1 Simple Algorithm

Given the definition of $S_\alpha(A, P)$, it is easy to come up with an algorithm that minimises it lexicographically. Algorithm 6.1 presents such an algorithm. It maintains a population of μ individuals. In each generation, it generates λ new offspring. From the union of the old population and the offspring generation it iteratively removes the individual p for which $S_\alpha(A, P \setminus \{p\})$ is lexicographically smallest. This approach is greedy and does not guarantee to achieve the best possible approximation among all the $\binom{\mu+\lambda}{\mu}$ possible sets. Lastly, every new individual is added to the archive A such that the archive only contains non-dominating solutions. As described in Algorithm 6.2, this means that (i) a new offspring is only added if it is not dominated by another individual already in A and (ii) individuals in A that are dominated by a new individual are removed. Note that in contrast to many other algorithms (like Laumanns et al. [54] or all hypervolume-based algorithms), our new algorithms needs no meta-parameters.

Algorithm 6.1: Simple $(\mu + \lambda)$-Approximation-Guided EA

```
1  Initialise population $P$ with $\mu$ random individuals;
2  Set archive $A \leftarrow P$;
3  foreach generation do
4      Initialise offspring population $O \leftarrow \emptyset$;
5      for $j \leftarrow 1$ to $\lambda$ do
6          Select two random individuals from $P$;
7          Apply crossover and mutation;
8          Add new individual to $O$;
9      foreach $p \in O$ do
10         Insert offspring $p$ in archive $A$ with Algorithm 6.2;
11     Add offsprings to population, i.e., $P \leftarrow P \cup O$;
12     while $|P| > \mu$ do
13         foreach $p \in P$ do
14             Compute $S_\alpha(A, P \setminus \{p\})$ with Algorithm 5.1;
15         Remove $p$ from $P$ for which $S_\alpha(A, P \setminus \{p\})$ is lexicographically smallest;
```

Algorithm 6.2: Insert point into archive

```
   input  : Archive $A$, Point $p \in \mathbb{R}^d$
   output: Archive consisting of the Pareto optimal points of $A \cup \{p\}$
1  dominated $\leftarrow$ false;
2  foreach $a \in A$ do
3      if $p \prec a$ then delete $a$ from $A$;
4      if $a \preceq p$ then dominated $\leftarrow$ true;
5  if not dominated then add $p$ to $A$;
6  return $A$;
```

We now give an upper bound for the runtime of Algorithm 6.2. One generation consists of producing and processing λ offspring. The main part of the runtime is needed for the $\mathcal{O}(\lambda(\mu+\lambda))$ computations of $S_\alpha(A, P \setminus \{p\})$, each costing $\mathcal{O}(d\,|A|\,(\mu+\lambda) + |A| \log |A|)$. Hence, we get a runtime of $\mathcal{O}(\lambda(\mu+\lambda)\,|A|\,(d\,(\mu+\lambda) + \log |A|))$ for generating an offspring of λ points. This means for N function evaluations, that is, N generated points overall, we get a total runtime of

$$\mathcal{O}(N\,(\mu+\lambda)\,|A|\,(d\,(\mu+\lambda) + \log |A|))$$

This algorithm works well for small population and offspring sizes $\mu + \lambda$, but for e.g. $\mu + \lambda = 100$, it becomes very slow due to the $(\mu+\lambda)^2$ factor.

Algorithm 6.3: Fast $(\mu + \lambda)$-Approximation-Guided EA

```
1–11 See lines 1–11 of Algorithm 6.1
12     foreach $a \in A$ do
13         $p_1(a) \leftarrow \operatorname{argmin}_{p \in P} \alpha(\{a\}, \{p\})$;
14         $p_2(a) \leftarrow \operatorname{argmin}_{p_1(a) \neq p \in P} \alpha(\{a\}, \{p\})$;
15         $\alpha_1(a) \leftarrow \min_{p \in P} \alpha(\{a\}, \{p\})$;
16         $\alpha_2(a) \leftarrow \min_{p_1(a) \neq p \in P} \alpha(\{a\}, \{p\})$;
17     foreach $p \in P$ do
18         $\beta(p) \leftarrow \max_{a \in A} \{\alpha_2(a) \mid p_1(a) = p\}$;
19     while $|P| > \mu$ do
20         Remove $p^*$ from $P$ with $\beta(p)$ minimal;
21         foreach $a \in A$ with $p_1(a) = p^*$ do
22             Compute $p_1(a), p_2(a), \alpha_1(a), \alpha_2(a)$ as done above in lines 13–16;
23             $\beta(p_1(a)) \leftarrow \max\{\beta(p_1(a)), \alpha_2(a)\}$;
```

6.2 Fast Algorithm

We now show how to speed-up our approach. Let us first assume that the approximations $\alpha(\{a\}, \{p\})$ are distinct for all $a \in A$ and $p \in P$. For all $a \in A$ we denote the point $p \in P$ that approximates it best by $p_1(a)$ and the second best by $p_2(a)$. The respective approximations we denote by $\alpha_i(a) := \alpha(\{a\}, \{p_i(a)\})$ for $i \in \{1, 2\}$. Now, let $p \neq q \in P$ and consider $S_p := S_\alpha(A, P \setminus \{p\})$ and $S_q := S_\alpha(A, P \setminus \{q\})$. Significant for the comparison of the two are only the positions $a \in A$ where S_p or S_q differ from $S := S_\alpha(A, P)$. This is the case for all positions in $B := \{a \in A \mid p_1(a) \in \{p, q\}\}$. If we delete p from P, then the worst approximation of one of the $a \in B$ is the maximum of $\max\{\alpha_2(a) \mid p_1(a) = p\}$ and $\max\{\alpha_1(a) \mid p_1(a) = q\}$. Now observe that if

$$\beta(p) := \max_{a \in A} \{\alpha_2(a) \mid p_1(a) = p\}$$

is smaller than the respective $\beta(q)$, then also the larger term above is smaller, as $\max\{\alpha_1(a) \mid p_1(a) = q\} < \max\{\alpha_2(a) \mid p_1(a) = q\}$. Hence, we end up with the fact that we only have to compare $\beta(p)$ and throw out the point p with $\beta(p)$ minimal (see Algorithm 6.3).

Recall that we assumed that all approximations $\alpha(\{a\}, \{p\})$ with $a \in A$, $p \in P$ are distinct. If this does not hold, we can simply change the indicator $S_\alpha(A, P)$ slightly and insert symmetry breaking terms $a \cdot \varepsilon$, where $\varepsilon > 0$ is an infinitesimal small number. This means that we treat equal approximations as not being equal and hence in some arbitrary order.

We now give an upper bound for the runtime of Algorithm 6.3. For one generation, i.e., for producing and processing λ offspring, the algorithm needs a runtime of $\mathcal{O}(d\,(\mu+\lambda)\,|A|)$ for computing the values $p_1(a), p_2(a), \alpha_1(a), \alpha_2(a)$ and $\beta(p)$ initially. Then we repeat λ times: We delete the point $p^* \in P$ with $\beta(p)$ minimal in $\mathcal{O}(\mu+\lambda)$, after which we have to recompute the values $p_1(a), p_2(a), \alpha_1(a), \alpha_2(a)$, but only for $a \in A$ with $p_1(a) = p^*$. Observe that we can store a list of these a's during the initial computation and keep these lists up to date with no increase of the asymptotic runtime. Also note that we would expect to find $\mathcal{O}(|A|/|P|)$ points with $p_1(a) = p^*$, while in the worst cases there may be up to $\mathcal{O}(|A|)$ such points. Summing up, we get a heuristic runtime for one generation of $\mathcal{O}(d\,(\mu+\lambda)\,|A| + \lambda((\mu+\lambda) + d|P|\cdot|A|/|P|))$, which simplifies to $\mathcal{O}(d(\mu+\lambda)|A|)$ as $|A| \geq \mu+\lambda$. In the worst case we replace $\mathcal{O}(|A|/|P|)$ by $\mathcal{O}(|A|)$ and get a runtime for one generation of $\mathcal{O}(d\lambda(\mu+\lambda)|A|)$. For N fitness evaluations we, therefore, get a runtime of $\mathcal{O}(d(1+\mu/\lambda)|A|N)$ heuristically, and $\mathcal{O}(d(\mu+\lambda)|A|N)$ in the worst case. Note that $|A| \leq N$. For any $\lambda = \mathcal{O}(\mu)$, e.g. $\lambda = 1$ or $\lambda = \mu$, this can be simplified to $\mathcal{O}(d\mu|A|N)$ in both cases, while for $\lambda = \Omega(\mu)$, e.g. $\lambda = \mu$, we get a reduced heuristic runtime of $\mathcal{O}(d|A|N)$.

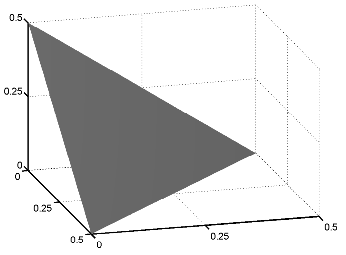
6.3 Experimental Study

The fast $(\mu + \lambda)$-version of our algorithm was implemented in the jMetal framework [28].[1] We compared the performance of our AGE algorithm to the established MOO algorithms IBEA [96], NSGA-II [20], SMS-EMOA [32], and SPEA2 [98] on the DTLZ benchmark family [21]. We used the functions DTLZ 1-4, each with $n = 30$ function variables and between 2 to 20 objective values/dimensions.[2] Figure 6.1 shows the Pareto fronts of DTLZ 1 and DTLZ 2 for three objectives. The fronts of DTLZ 3 and DTLZ 4 are equivalent to DTLZ 2; they only differ in the mapping from the search space to the objective space. We limit the calculations of the algorithms to a maximum of 100,000 fitness evaluations *and* a maximum computation time of four hours per run. Note that the time restriction had to be used as the runtime of some algorithms increases exponentially with respect to the size of the objective space.

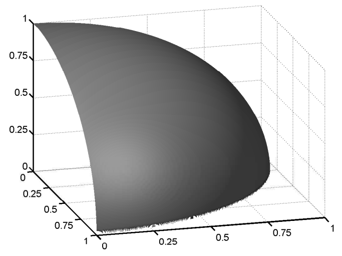
The further parameter setup of the algorithms is as follows. Parents are selected through a binary tournament, in which we select the individual out of two randomly drawn ones with the better approximation of the archive. As variation operators, the polynomial mutation and the simulated binary crossover [1] are applied, which are both used widely in MOO algorithms [20, 40, 98]. The distribution parameters associated with the operators are $\eta_m = 20.0$ and $\eta_c = 20.0$. The crossover operator is biased towards the creation of offspring that are close to the parents, and was applied with

[1]The code is available here `http://cs.adelaide.edu.au/~ec/research/age.php`.

[2]It is important to note that we use the terms ‘objective values’ and ‘dimensions’ interchangeably, as our focus is solely on the dimensionality of the objective space.



(a) DTLZ 1.



(b) DTLZ 2.

Figure 6.1: Visualization of the Pareto fronts for $d = 3$.

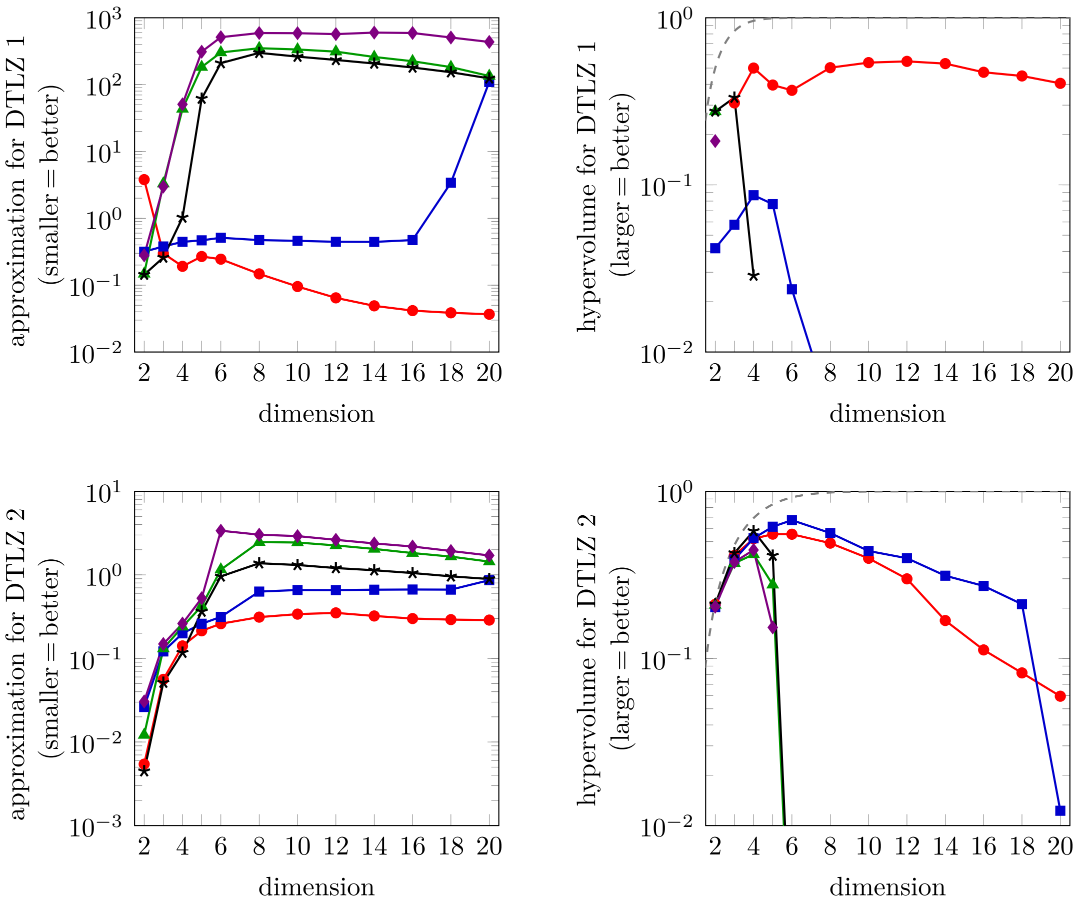


Figure 6.2: Comparison of the performance of our algorithm AGE (—●—) with IBEA (■), NSGA-II (▲), SMS-EMOA ($\star$), and SPEA2 (♦) on the test functions DTLZ 1 and DTLZ 2 with varying dimension d. The figures show the average of 100 repetitions each. Only non-zero hypervolume values are shown. For reference, we also plot (- - -) the maximum hypervolume achievable for $\mu \to \infty$.

$p_c = 0.9$. The mutation operator has a specialised explorative effect for MOO problems, and is applied with $p_m = 1/n$, where n denotes the number of variables.

Figures 6.2 and 6.3 present our results for population size $\mu = 100$ and $\lambda = 100$, averaged over 100 independent runs. We performed the same experiments also for

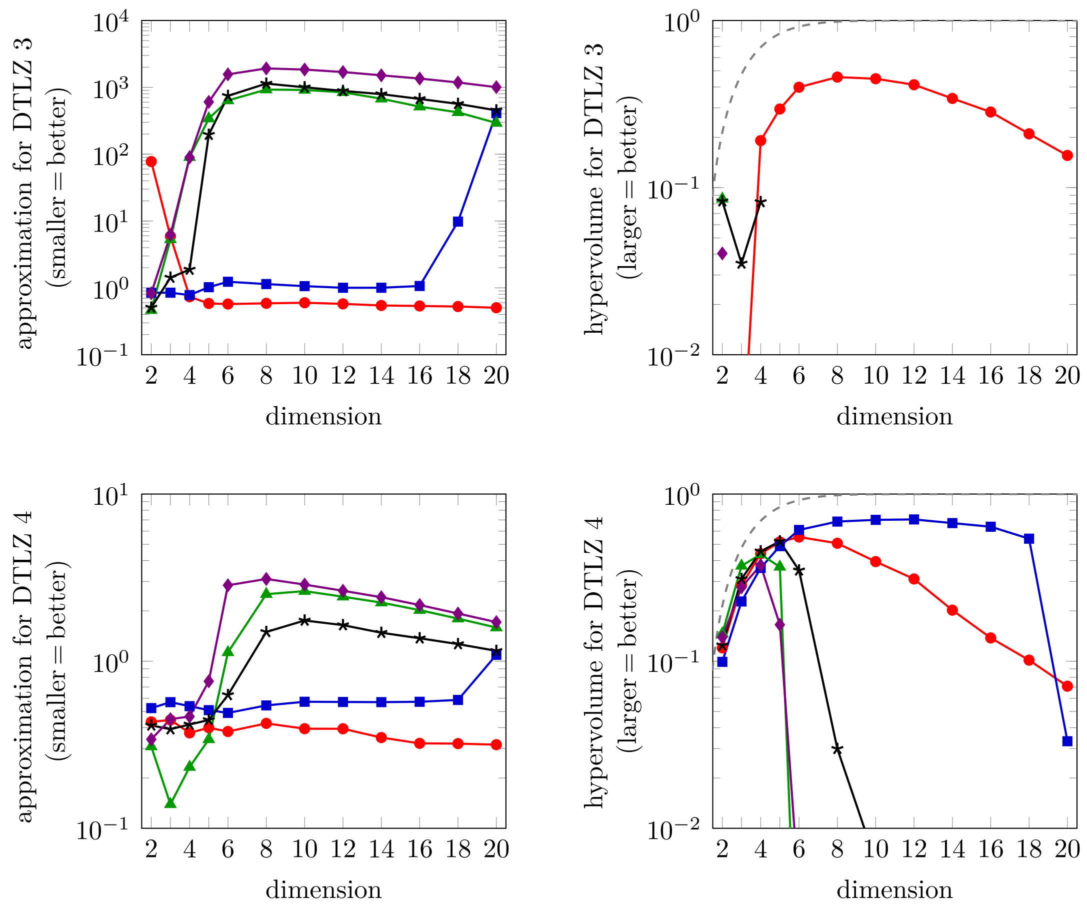


Figure 6.3: Comparison of the performance of our algorithm AGE (—●—) with IBEA (■), NSGA-II (▲), SMS-EMOA (⋆), and SPEA2 (♦) on the test functions DTLZ 3 and DTLZ 4 with varying dimension d. The figures show the average of 100 repetitions each. Only non-zero hypervolume values are shown. For reference, we also plot (- - -) the maximum hypervolume achievable for $\mu \to \infty$.

$\mu \in \{25, 50\}$ and observed similar behaviors. We assess the algorithms using the following measures:

- *Approximation:* We approximate the achieved additive approximation of the known Pareto fronts by first drawing one million points of the front uniformly at random and then computing the approximation which the final population achieved for this set with Algorithm 5.1.
- The *hypervolume* [97] is a popular performance measure which measures the volume of the dominated portion of the objective space relative to a reference point r. For DTLZ1 we choose $r = 0.5^d$, otherwise $r = 1^d$. We approximate the achieved hypervolume with an FPRAS [9] which has a relative error of more than 2% with probability less than 1/1000. The volumes shown for DTLZ 1 are normalised by the factor 2^d.

As it is very hard to determine the minimum approximation ratio achievable or the maximum hypervolume achievable for all populations of a fixed size μ, we only plot the theoretical maximum hypervolume for $\mu \to \infty$. For this, a simple geometric calculation gives a maximum (rescaled) hypervolume of $1 - 1/d!$ for DTLZ1 and a maximum hypervolume of $1 - 2^{-d}\pi^{d/2}/(d/2)!$ for DTLZ 2 (with $n! := \Gamma(n+1)$).

For all test functions, our new algorithm AGE (—●—) achieves the best approximation among the competing algorithms for dimensions $d > 5$. We first discuss DTLZ 1 and DTLZ 3 who are known to be hard to analyze as they contain a very large number of local Pareto-optimal fronts [21]. For both functions, we achieve the best approximation among all tested algorithms for $d > 3$. Remarkably, all other algorithms (besides IBEA (■)) are unable to find the front at all for these instances. This results in extremely large approximations and zero hypervolumes. The reason for IBEAs decreasing behaviour for very large dimension ($d \geq 18$) is that it was stopped after four hours and it could not perform 100,000 iterations. The same holds already for much smaller dimensions in the case of SMS-EMOA ($\star$), which uses an exponential-time algorithm to internally determine the hypervolume. It did not finish a single generation for $d \geq 8$ and only performed around 5,000 iterations within four hours for $d = 5$. This implies that the higher-dimensional approximations plotted for SMS-EMOA ($\star$) actually show the approximation of the random initial population. Interestingly, the approximations achieved by NSGA-II (▲) and SPEA2 (♦) are even worse as they are tuned for low-dimensional problems and move their population too far out to the boundaries for high dimensions. Our algorithm (—●—) and also NSGA-II (▲) and SPEA2 (♦) always finished in less than four hours.

The plots of DTLZ 2 and DTLZ 4 reveal other properties. Here, an approximation of the front seems generally much easier. For small dimensions ($d = 2, 3, 4$), all algorithms find acceptable solutions. However, for larger dimensions again SMS-EMOA ($\star$), NSGA-II (▲), and SPEA2 (♦) fail for the said reasons. In these cases, our algorithm (—●—) still achieves the best approximation, but for $4 \leq d \leq 18$ (DTLZ 2) and $6 \leq d \leq 18$ (DTLZ 4), the solutions found by IBEA (■) (which uses the hypervolume as an indicator) have a larger hypervolume. The hypervolume of IBEA is only worse for $d = 20$ because it could only perform a few hundred iterations within the four hour time limit.

6.4 Conclusions

Evolutionary algorithms are frequently used to solve multi-objective optimisation problems. Often, it is very hard to formally define the optimisation that current state-of-the-art approach work with. We have presented a new evolutionary multi-objective algorithm that works with a formal notion of approximation. The framework of our algorithm allows to work with various formal notions of approximations. Our experimental results show that given a fixed time budget it outperforms current state-of-the-art approaches in terms of the desired additive approximation as well as the covered hypervolume on standard benchmark functions. This holds, in particular, for problems with many objectives, which most other algorithms have difficulties dealing with.

This work has been published in the proceedings of the 21nd International Joint Conference on Artificial Intelligence (IJCAI) in 2011 [13].

Not with whom you are born, but with whom you are bred.

Miguel de Cervantes Saavedra, Don Quixote

7 Approximation-Guided Evolution II

Approximation-Guided Evolution (AGE) outperforms state-of-the-art multi-multi-objective algorithms in terms of approximation quality, as we have seen in the previous chapter. In particular, this holds for problems with many objectives, which most other algorithms have difficulties dealing with. Quite surprisingly, it is the other way around when the problems have just very few objectives. As can be seen in Figure 7.5 (that will serve us for our final evaluation), the original AGE (—●—) is clearly outperformed by other algorithms in several cases when the problem has just two to three objectives.

We identified the following two important and disadvantageous properties of AGE:

1. A new but incomparable point is added to the archive independent of how *different* it is (see Line 10, Algorithm 6.3). These unconditional insertions can lead to huge archives that consequently slow down the algorithm. In Section 7.1.1, we introduce a technique to approximate the set of incomparable solutions seen.

2. The parents for the mating process are selected uniformly at random (see Line 6, Algorithm 6.3). Interestingly, this random selection does not seem to be detrimental to the algorithm's performance on problems with many objectives. However, realising that the selection process might be improved motivated us to investigate algorithm-specific selection processes (see Section 7.1.2).

Of course, it is not clear whether approximating the archive gives an approximation of the Pareto front. However, the intuition is that after some time the archive approximates the front quite well, so that an approximation of the archive directly yields an approximation of the front. The experiments presented later-on show that this intuition is right, as our algorithm indeed finds good approximations of the fronts.

We propose a fast and effective approximation-guided evolutionary algorithm called AGE-II. It is fast and performs well for problems with many as well as few objectives. It can be seen as a generalization of AGE, but it allows to trade-off archive size and speed of convergence. To do so, we adapt the ϵ-dominance approach [54] in order to approximate the different points seen so far during the run of the algorithm (for similar approaches see, e.g., [77, 78]). Furthermore, we change the selection of parents being used for reproduction such that the algorithm is able to achieve a better spread along the Pareto front. Our experiments show that AGE-II performs very well for multi-objective problems having few as well as many objectives. It scales well with the number of objectives and enables practitioners to add objectives to their problems at small additional computational cost.

The outline of this chapter is as follows. In Section 7.1 we introduce our new algorithm. In Section 7.2 we showcase the computational speed-up. In Section 7.3 we report on our experimental investigations. Finally, we finish with some conclusions.

7.1 AGE-II

In this section, we present the improvements to AGE that lead to AGE-II. We show how we adapt the ϵ-dominance approach Laumanns et al. [54] in order to approximate the different points seen so far during the run of the algorithm. Subsequently, we briefly motivate our parent selection strategy.

7.1.1 Archive Approximation

The size of the archive can grow to sizes that slow down the original AGE tremendously. Interestingly, we are thus facing a problem that is similar to the original problem of multi-objective optimisation: a set of solutions is sought that nicely represents the true set of compromise solutions. In order to achieve this, we reuse AGE's own main idea of maintaining a small set that approximates the true Pareto front. By approximating the archive as well in a controlled manner, we can guarantee a maximum size of the archive, and thus prevent the archive from slowing down the selection procedure. We achieve this based on the idea of ϵ-dominance introduced in Laumanns et al. [54]. Instead of using an archive A^t that stores at any point in time t the whole set of non-dominated objective vectors, we are using an archive $A^{(t)}_{\epsilon_{grid}}$ that stores an additive ϵ-approximation of the non-dominated objective vectors produced until time step t.

In order to maintain such an approximation during the run of the algorithm, a grid on the objective space is used to pick a small set of representatives (based on

Algorithm 7.1: Outline of Approximation-Guided EA II

```
1  Initialize population $P$ with $\mu$ random individuals;
2  Set $\epsilon_{grid}$ the resolution of the *approximative archive* $A_{\epsilon_{grid}}$;
3  foreach $p \in P$ do
4      Insert offspring *floor*$(p)$ in the *approximative archive* $A_{\epsilon_{grid}}$ such that only
       non-dominated solutions remain;
5  foreach *generation* do
6      Initialize offspring population $O \leftarrow \emptyset$;
7      for $j \leftarrow 1$ to $\lambda$ do
8          Select two individuals from $P$ (see Section 7.1.2);
9          Apply crossover and mutation;
10         Add new individual to $O$;
11     foreach $p \in O$ do
12         Insert offspring *floor*$(p)$ in the *approximative archive* $A_{\epsilon_{grid}}$ such that only
           non-dominated solutions remain;
13         Discard offspring $p$ if it is dominated by any point *increment*$(a)$, $a \in A$;
14     Add offsprings to population, i.e., $P \leftarrow P \cup O$;
15     while $|P| > \mu$ do
16         Remove $p$ from $P$ that is of least importance to the approximation (for
           details on this step see [13]);
```

Algorithm 7.2: Function *floor*

```
   input  : $d$-dimensional objective vector $x$, archive parameter $\epsilon_{grid}$
   output: Corresponding vector $v$ on the $\epsilon$-grid
1  for $i = 1$ to $d$ do  $v[i] \leftarrow \left\lfloor \frac{x[i]}{\epsilon_{grid}} \right\rfloor$ ;
```

ϵ-dominance). We reuse the *update*-mechanism from [54], and thus can maintain the ϵ-Pareto set $A^{(t)}_{\epsilon_{grid}}$ of the set $A^{(t)}$ of all solutions seen so far. Due to [54], the size is bounded by

$$\left|A^{(t)}_{\epsilon_{grid}}\right| \leq \prod_{j=1}^{m-1} \left\lfloor \frac{K}{\epsilon_{grid}} \right\rfloor$$

where

$$K = \max_{i=1}^{d} \left(\max_{s \in S} f_i(s) \right)$$

is the maximum function value attainable among all objective functions.

Algorithm 7.3: Function *increment*

```
input  : $d$-dimensional vector $x$, archive parameter $\epsilon_{grid}$
output: Corresponding vector $v$ that has each of its components increased by 1
1 for $i = 1$ to $d$ do  $v[i] \leftarrow o[i] + 1$ ;
```

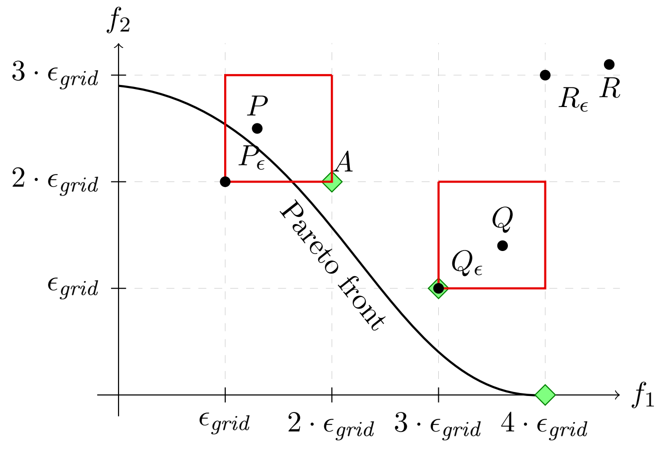

f_2
$3 \cdot \epsilon_{grid}$
$2 \cdot \epsilon_{grid}$
ϵ_{grid}
f_1
ϵ_{grid}
$2 \cdot \epsilon_{grid}$
$3 \cdot \epsilon_{grid}$
$4 \cdot \epsilon_{grid}$
P
P_ϵ
A
Q
Q_ϵ
R_ϵ
R
Pareto front

Figure 7.1: The newly generated points P, Q, and R are shown with their corresponding *additive ϵ-approximations* P_ϵ, Q_ϵ, and R_ϵ. Both objectives f_1 and f_2 are to be minimised, and the current *approximative archive* is represented by ◆. Only P_ϵ will be added to the *approximative archive*, replacing A. Both P and Q will be candidates for the selection process to form the next population.

Our new algorithm called AGE-II is parametrised by the desired approximation quality $\epsilon_{grid} \geq 0$ of the archive with respect to the seen objective vectors. AGE-II is shown in Algorithm 7.1, and it uses the helper functions given in Algorithms 7.2 and 7.3. The latter is used to perform a relaxed dominance check on the offspring p in Line 13. A strict dominance check here would require an offspring to be not dominated by any point in the entire archive. However, as the archive approximates all the solutions seen so far (via the flooring), it might be very unlikely, or even impossible, to find solutions that pass the strict dominance test.

7.1.2 Higher Performance for Lower Dimensions

Quite interestingly, and despite AGE's good performance on problems with many objective, it is clearly outperformed by other algorithms in several cases, when the problem has just two or three objectives. The key discovery is that the random parent selection of AGE is free of any bias. For problems with many objectives, this is not a problem, and can even be seen as its biggest advantage. For problems with just a few objectives, however, it is well known that one can do better than random selection, such as selection based on crowding distance, hypervolume contribution, etc. Such strategies then select potential candidates based on their relative position in the current population. For AGE, the lack of this bias means that solutions can be picked for parents that are not necessarily candidates with high potential. Consequently, it is not surprising to see that the original AGE is outperformed by algorithms that do well with their parent selection strategy, *if* their strategy is effective in the d-dimensional objective space.

Based on previous experiments [86], we choose the following parent selection strategy ‘ for the final comparison against the established algorithms. Firstly, the population is reduced: solutions in the front i have a probability of $1/i$ of staying in the population. Secondly, a binary tournament on two randomly selected solutions from the reduced pool is performed for the parent selection, where solutions of higher crowding distance are preferred. The consequence of the reduction is that all solutions that form the first front are kept in the population, so are the extreme points. Additionally, solutions that are dominated multiple times are less likely to be selected as a potential parent. The use of the crowding distance then helps with maintaining a diverse set of solutions in low-dimensional objective space. Both steps taken together significantly increase the selection pressure over the original random selection in AGE. At the same time, they are quick to compute and their effects diminish when the number of objectives increases.

7.2 Speed-up through approximative archives

AGE-II works at each time step t with an approximation $A^{(t)}_{\epsilon_{grid}}$ of the set of non-dominated points A^t seen until time step t. Note, that setting $\epsilon_{grid} = 0$ implies the original AGE approach that stores every non-dominated objective vector. In this section, we want to investigate the effect of working with different archives sizes (determined by the choice of ϵ_{grid}) in AGE-II. Our goal is to understand the effect of the choice of this parameter on the actual archive size used during the run of the algorithm as well as on the approximation quality obtained by AGE-II.

Next, we outline the results of our experimental investigation of the influence of approximative archives on the runtime and the solution qualities. Note, that the computational complexity of the original AGE is linear in the number of objectives, and this holds for AGE-II, too. The algorithm was implemented in the jMetal framework [28] and is publicly available[1].

The parameter setup of AGE-II is as follows. As variation operators, the polynomial mutation and the simulated binary crossover [1] were applied, which are both used widely in MOO algorithms [20, 40, 98]. The distribution parameters associated with the operators were $\eta_m = 20.0$ and $\eta_c = 20.0$. The crossover operator is biased towards the creation of offspring that are close to the parents, and was applied with $p_c = 0.9$.

[1] http://cs.adelaide.edu.au/~ec/research/age.php

The mutation operator has a specialized explorative effect for MOO problems, and was applied with $p_m = 1/n$, where n denotes the number of variables. Population size was set to $\mu = 100$ and $\lambda = 100$, and each setup was given a budget of 100,000 evaluations. We assess the selection schemes and algorithms using the additive approximation measures ([13]): we approximate the achieved additive approximation of the known Pareto fronts by first drawing one million points of the front uniformly at random and then computing the additive approximation that the final population achieved for this set.

Figure 7.2 shows the results averaged over 100 independent runs. Note how different the archive growth is for the different selected functions in the cases of $\epsilon_{grid} = 0$, where every non-dominated point is stored. For DTLZ 1, $d = 2$ the archive stays very small, with about 80 solutions in the end. Even in the case of DTLZ 3, $d = 10$ (a function with a similar objective space to that of DTLZ 1) only about every tenth solution is kept in the archive, which eventually contains about 9,000 solutions. For the similar objective spaces of DTLZ 2, $d = 3$ and DTLZ 4, $d = 20$ this situation is significantly different, and the rate of producing non-dominated points is significantly higher. In the case of the latter, over 90% of all generated solutions are added to the archive, if the insertion is just based on non-dominance. This situation changes only slightly, when a relatively "coarse" $\epsilon_{grid} = 0.1$ is used. For DTLZ 3, $d = 10$, the same value of the grid results in an enormous reduction in archive size.

Consequently, the choice of ϵ_{grid} has a significant impact on the runtime and even on the solution quality. For DTLZ 1, $d = 2$ the quality of the final population can be increased, whereas the use of an approximative archive has little impact on the archive size in this case. Of the tested values for DTLZ 2, $d = 3$ the choice of $\epsilon_{grid} = 0.01$ offers a speed-up by a factor of eight. Additional speed-ups can be achieved, but they come at the cost of worse final approximations. For DTLZ 3, $d = 10$ a similar observation can be made: if a minor reduction in quality is tolerable, then a speed-up by a factor of four can be achieved. The situation is very different for the 20-dimensional DTLZ 4, where a speed-up by a factor of over 250 can be achieved, while achieving even better quality solutions as well.

7.3 Benchmark Results

In this section, we compare AGE-II to well known evolutionary multi-objective algorithms including the original AGE on commonly used benchmark functions.

7.3.1 Experimental Setup

We use the jMetal framework [28] to compare our AGE-II with the original AGE, and with the established algorithms IBEA [96], NSGA-II [20], SMS-EMOA [32], and SPEA2 [98] on the benchmark families WFG [46], LZ [56], and DTLZ [21]. The test setup is identical to that of [13], and to the already outlined setup of Section 7.2. It is important to note that we limit the calculations of the algorithms to a maximum of 50,000/100,000/150,000 fitness evaluations for WFG/DTLZ/LZ *and* to a maximum computation time of 4 hours per run, as the runtime of some algorithms increases exponentially with respect to the size of the objective space.[2] The further parameter setup of the algorithms is as follows. Parents were selected through a binary tournament (unless further specified). We will present our results for population size $\mu = 100$ and $\lambda = 100$, averaged over 100 independent runs.

We assess the algorithms by taking their final populations, and then using the afore-described additive approximation measure and the *hypervolume* [97]. The latter is a popular performance measure that measures the volume of the dominated portion of the objective space relative to a reference point r. For the quality assessment on the WFG and LZ functions, we computed the achieved additive approximations and the hypervolumes with respect to the Pareto fronts given in the jMetal package. For DTLZ 1 we choose $r = 0.5^d$, otherwise $r = 1^d$. We approximate the achieved hypervolume with an FPRAS [9], which has a relative error of more than 2% with probability less than 1/1000. The volumes shown for DTLZ 1 are normalized by the factor 2^d. As it is very hard to determine the minimum approximation ratio achievable or the maximum hypervolume achievable for all populations of a fixed size μ, we only plot the theoretical maximum hypervolume for $\mu \to \infty$ as a reference.

Note that, by the design of the additive approximation indicator, the approximation values indicate the distributions of the solution and their distances from the Pareto front, as no point on the Pareto front is approximated worse than the indicator value.

7.3.2 Experiment results

The benchmarking results for the different algorithms are shown in Figures 7.3, 7.4, and 7.5. In summary, AGE-II ranks among the best algorithms on the low-dimensional WFG and LZ functions. This holds for both the additive approximation quality, as well as for the achieved hypervolumes. Interestingly, NSGA-II (—▲—) that normally performs rather well on such problems, is beaten in almost all cases. SPEA2 (—◆—) and IBEA (—■—) on average perform better. AGE (—●—), SMS-EMOA (—⋆—), and AGE-II ($\epsilon_{grid} = 0.1$: ∘, $\epsilon_{grid} = 0.01$: ∘) often perform very similarly.

Our investigations on the DTLZ family prove to be more differentiating. As these can be scaled in the number of objectives, the advantages and disadvantages of the algorithms' underlying mechanisms become more apparent:

[2]Again, it is important to note that we use the terms 'objective values' and 'dimensions' interchangeably, as our focus is solely on the dimensionality of the objective space.

- AGE-II ($\epsilon_{grid} = 0.1$: ∘, $\epsilon_{grid} = 0.01$: ∘) shows a significantly improved performance on the lower-dimensional DTLZ 1, DTLZ 3, and DTLZ 4 variants. Furthermore, it is either the best performing algorithm, or in many cases, it shows at least competitive performance.

- It is interesting to see that our AGE-II incorporates the crowding distance idea from NSGA-II (—▲—) for a fitness assignment, but is not influenced by its detrimental effects in higher dimensional objective spaces. This is thanks to the way how the next generation is formed (i.e., based on contributions to the approximation achieved of the archive, see Line 16 of Algorithm 7.1).

- When compared with the original AGE (—●—), then our modification does exhibit a performance improvement in all cases. Still, as AGE-II shows a consistent performance across all scaled functions, we deem the minimal loss in quality (in our experimental setup) as negligible.

- Remarkably, NSGA-II (—▲—), SMS-EMOA (—⋆—), and SPEA2 (—◆—) are unable to find the front of the high-dimensional DTLZ 1 and DTLZ 3 variants. This results in extremely large approximations and zero hypervolumes.

- The reason for IBEA's decreasing behaviour for very large dimension ($d \geq 18$) is that it was stopped after 4 hours and it could not perform $100,000$ iterations. The same holds already for much smaller dimensions for SMS-EMOA (—⋆—), which uses an exponential-time algorithm to internally determine the hypervolume. It did not finish a single generation for $d \geq 8$ and only performed around $5,000$ iterations within four hours for $d = 5$. This implies that the higher-dimensional approximations plotted for SMS-EMOA actually show the approximation of the random initial population.

- Interestingly, the approximations achieved by NSGA-II (—▲—) and SPEA2 (—◆—) are even worse as they are tuned for low-dimensional problems and move their population too far out to the boundaries for high dimensions.

7.4 Conclusions

Approximation guided evolutionary algorithms work with a formal notion of approximation and have the ability to work with problems that have with many objectives. Our new approximation-guided algorithm called AGE-II efficiently solves problems with few and with many conflicting objectives. Its computation time increases only linearly with the number of objectives. We control the size of the archive which mainly determines its computational cost, and thus observed runtime reductions by a factor of up to 250 over its predecessor, without a sacrifice of final solution quality.

Our experimental results show that given a fixed time budget it outperforms current state-of-the-art approaches in terms of the desired additive approximation on standard benchmark functions for more than four objectives. On functions with two and three objectives, it lies level with the best approaches. Additionally, it also performs competitive or better regarding the covered hypervolume, depending on the function. This holds in particular for problems with many objectives, which most other algorithms have difficulties dealing with.

In summary, AGE-II is an efficient approach to solve multi-problems with few and many objectives. It enables practitioners now to add objectives with only minor consequences, and to explore problems for even higher dimensions.

This work has been accepted for publication in the proceedings of the Genetic and Evolutionary Computation Conference (GECCO) in 2013 [88], and for publication in the proceedings of the 2013 IEEE Congress on Evolutionary Computation (CEC) [86].

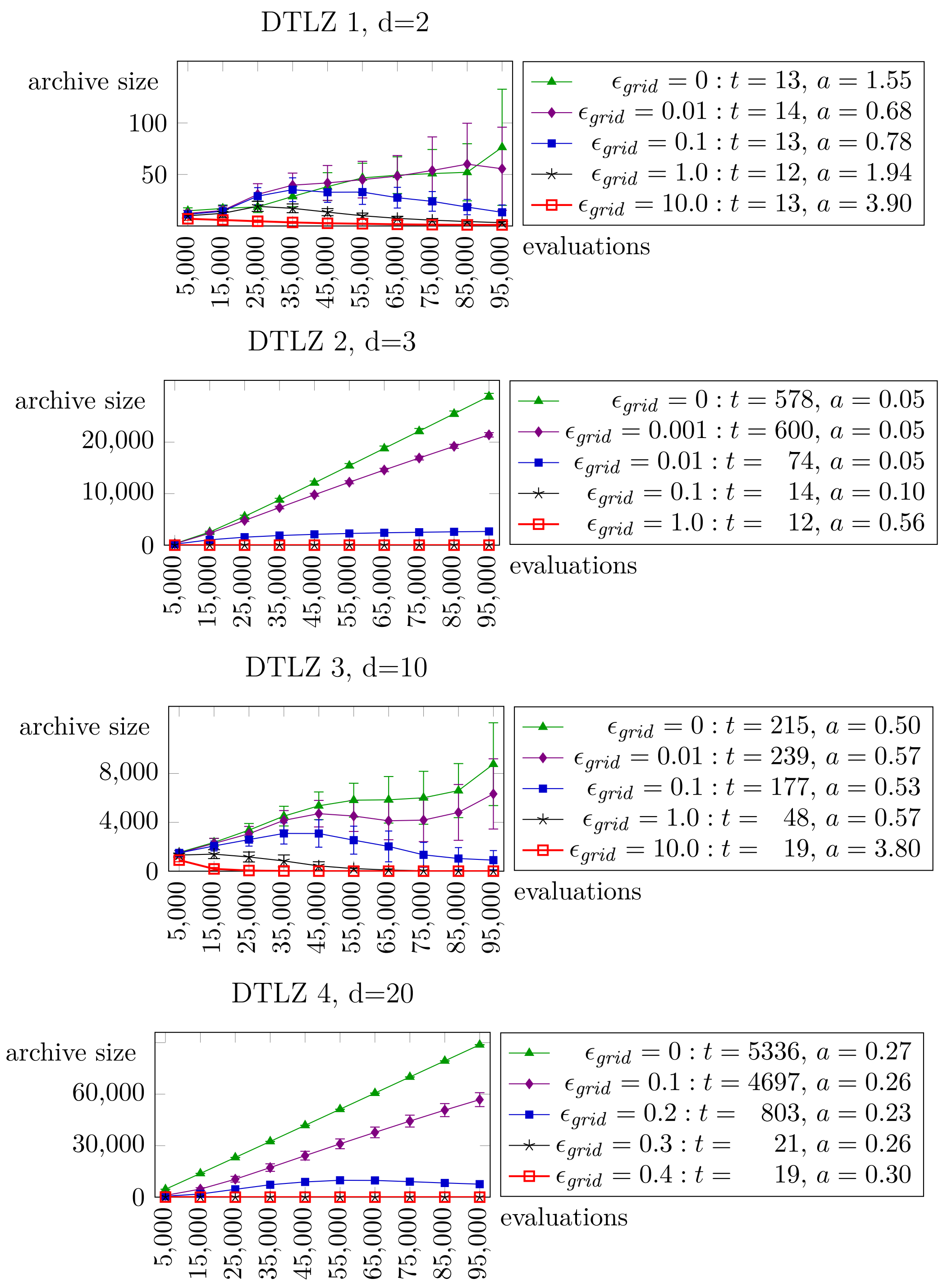


Figure 7.2: Influence of ϵ_{grid} on the archive size, the runtime, and the final quality. Shown are the means of the archive sizes, and their standard deviation is shown as error bars. Additionally, the means of the runtime t in seconds and the achieved additive approximation a of the true Pareto front are listed (smaller values are better). Note: the archive can grow linearly with the number of solutions generated, even when problem have just $d = 3$ objectives. Furthermore, if the front is small compared to the volume of the objective space (see DTLZ 1 and 3), then the archive can grow and shrink during the optimisation.

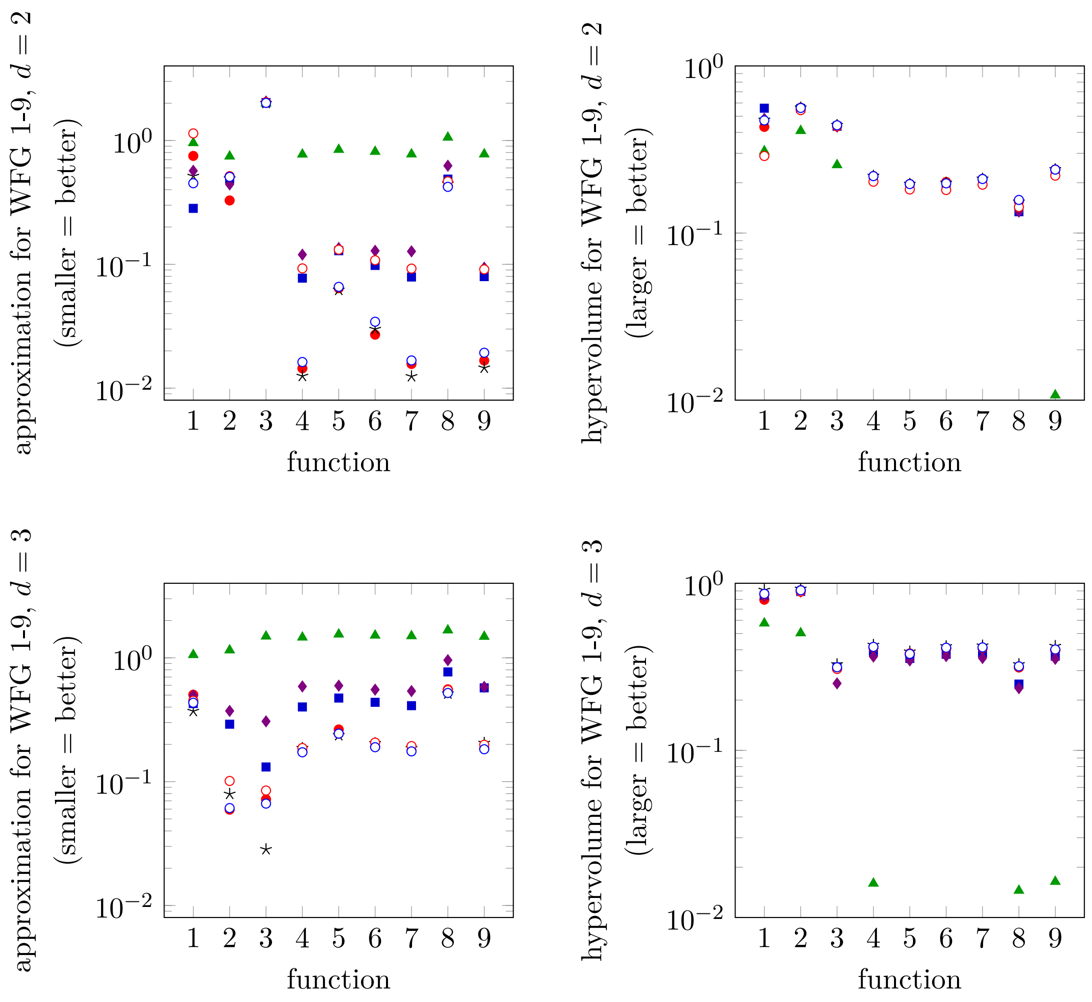


Figure 7.3: Comparison of the performance of our AGE-II ($\epsilon_{grid} = 0.1$: ○, $\epsilon_{grid} = 0.01$: ○) with the original AGE (●), IBEA (■), NSGA-II (▲), SMS-EMOA ($\star$), and SPEA2 (♦) with varying dimension d. The figures show the average of 100 repetitions each. Only non-zero hypervolume values are shown.

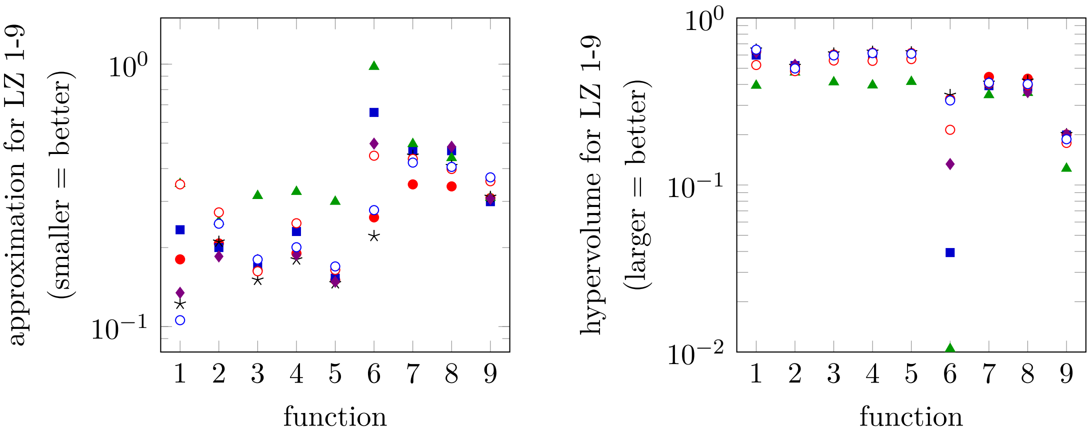


Figure 7.4: Comparison of the performance of our AGE-II ($\epsilon_{grid} = 0.1$: ○, $\epsilon_{grid} = 0.01$: ○) with the original AGE (●), IBEA (■), NSGA-II (▲), SMS-EMOA ($\star$), and SPEA2 (♦) with varying dimension d. The figures show the average of 100 repetitions each. Only non-zero hypervolume values are shown.

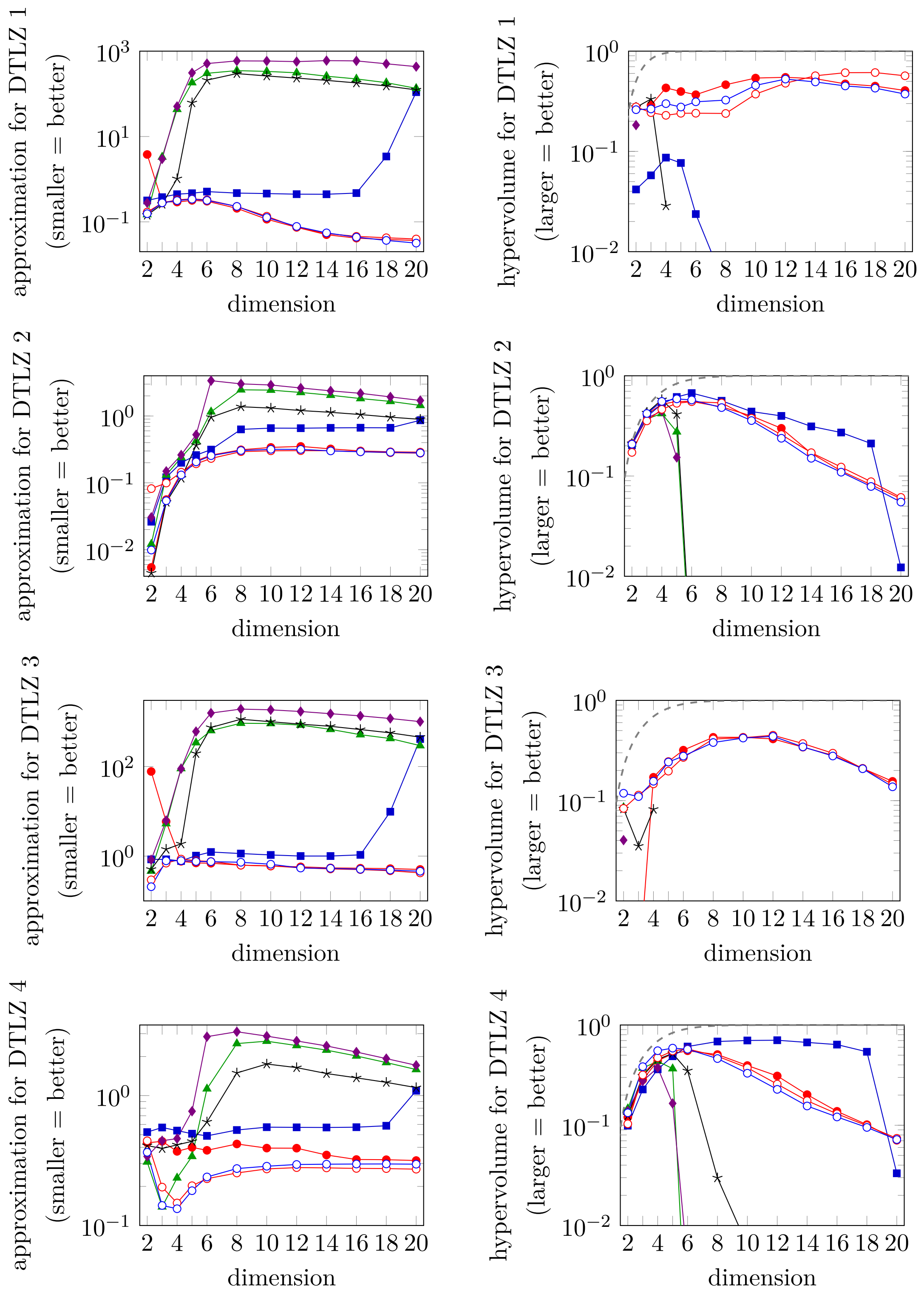


Figure 7.5: Comparison of the performance of our AGE-II (ϵ_{grid} = 0.1: ─○─, ϵ_{grid} = 0.01: ─○─) with the original AGE (─●─), IBEA (─■─), NSGA-II (─▲─), SMS-EMOA (─⋆─), and SPEA2 (─◆─) with varying dimension d. The figures show the average of 100 repetitions each. Only non-zero hypervolume values are shown. For reference, we also plot (- - -) the maximum hypervolume achievable for $\mu \to \infty$.

Part III

Applications to Wind Farm Optimisation

I was born on the prairies where the wind blew free and there was nothing to break the light of the sun. I was born where there were no enclosures.

Geronimo

8 Wind Farm Optimisation

Renewable energy is energy that comes from natural resources, such as sunlight, wind, tides, and geothermal heat. With renewable energy being a 'hot topic' right now and the renewable energy market rapidly expanding worldwide, the rapid growth of the renewable energy industry has led to cost reduction challenges. In order to deal with those complex challenges bio-inspired algorithms have been applied several times. However, the results achieved so far are not very satisfying, as most of the underlying models make inaccurate assumptions in order to make the optimisation process computationally feasible. Moreover, there is often significant room left for improvements of the results as well.

We are interested in applications of computational intelligence methods to the production of renewable energy. Specifically, we are pursuing optimisation challenges arising in wind power generation. The problem we study in this part of the thesis arises during the preliminary phase of a wind turbine farm project.

8.1 Introduction

The *wind farm layout problem* entails the process of planning the placement of turbines on a potential wind farm site, and the layout design of a wind farm is an important component of ensuring the profitability of a wind farm project [95]. There, an inadequate design would lead to lower than expected wind power capture, increased maintenance costs, etc. The creation of a 'good' farm layout involves the invocation of a software optimisation module, which attempts to efficiently place the turbines while adhering to the constraints and optimising the stated objectives. Often, this module is embedded within a specialised tool provided by wind power consultants such as Garrad Hassan or AWS TruePower, who offer a product such as OpenWind [3]. One of the problems that such tools have to deal with, apart from the actual optimisation, is the scaling cost of computing wake effects when estimating energy capture for increasing numbers of turbines. To estimate the energy capture of a layout the optimisation module models the free stream wind flowing through the site in and out of the turbines. Some degree of non-linear wind turbulence occurs at the outflow of a turbine and affects the inflow to turbines close enough behind it. Modeling this effect is necessary because wake has a great effect on the actual energy output of a wind farm. However, the time for computing the wake effect with respect to a given wake model such as the Park wake model [66] takes time $\Omega(n^2)$, where n is the number of turbines on the wind farm. This computational effort is significant if one is applying iterative search algorithms such as local search, simulated annealing or evolutionary algorithms for the optimisation of the placement. Such methods would need to evaluate each farm layout with respect to the wake model under consideration and would therefore require time $\Omega(n^2)$ for each solution that is considered during the optimisation process.

The goal of our research on the applications of bio-inspired algorithms in renewable energy production is to create models that are as accurate as possible, and to improve their computational complexity in order to make the optimisation of complex real-life scenarios computationally feasible. Additionally, specialised algorithms are designed that consider the models' special characteristics.

8.2 State of the Art

The optimal siting of wind turbines on a given area of land is a complex optimisation problem that is hard to solve by exact methods. The decision space is non-linear with respect to how sited turbines interact, when considering wake loss and energy capture.

Several bio-inspired computation techniques such as evolutionary strategies [79] (ES), genetic algorithms [38] (GA) and particle swarm optimisation [49] (PSO) have been used for the optimisation.

The different approaches for the wind farm layout problem are summarised in [74]. Wan et al. [91–93], use cell-based approaches and compare different bio-inspired algorithms, each applied to the same set of wind farm models and parameters. They use successively more expressive layout representations (and algorithms)[1] to relax where in a cell a turbine can be located: strictly in the middle, anywhere, or anywhere subject to proximity constraints with neighbouring turbines. An alternative to cell placement was explored in [51]: each turbine's location is a decision variable pair of real-valued,

[1]from a binary to a real-coded genetic algorithms, to particle swarm optimisation

spatial (x,y) coordinates. In that paper a simple ES is applied to optimise the placement of the turbines. In general, an ES is effective because it is easily parallelised and it self-adapts the extent to which it perturbs decision variables when generating a new potential solution. However, the algorithm given in [51] is only able to deal with problems that have just a small number of turbines.

For example, in the optimisation formulation in [61], the turbines are placed in regular grids, and wake is considered by enforcing minimal and maximal distances between the turbines, thus effectively neglecting physical effects. In [82], distance-dependent wakes are considered, but the underlying wind scenario was randomly generated, and is the result of unrealistic assumptions. On the other hand, the above-mentioned industry tool AWS OpenWind contains elaborate wake models that are used for real-world scenarios, but its simple optimiser neither considers the turbines' vicinities when perturbing layouts, nor adapts parameters for the perturbation during the run.

Note that, placing turbines on a defined area is loosely related to the difficult problem of packing discs in shapes [42, 80]. There, the task is to arrange n identical discs without overlap entirely inside a square on the plane, such that the discs have the largest possible diameter. As the turbines influence each other via non-linear wake-effects, these "influence areas" can be assumed to be circular for very simple scenarios of uniform wind distributions.[2] The naive goal then can be to minimise the overlap of the influence discs. However, the theoretical results from the disc packing are not applicable in our context, as it would be extremely simplifying to assume that these discs are circular due to non-uniform wind distributions. Additionally, the sizes and irregular shapes are highly dependent on the interaction of the turbines: due to interactions, the wind resources available at a turbine change, and so does then the influence area around a turbine.

In summary, related work typically has the following three drawbacks:

- Cell based approach: It allows limited flexibility in the layout optimisation because a layout consists of cells, which each have a turbine at the centre of the cell.

- Small number of turbines: It usually optimises for a farm size of 8-30 turbines.

- Limited Scalability: Previous approaches have been very time consuming, requiring up to several weeks for a single optimisation run for large wind farm layouts.

In the subsequent Chapters 9 and 10 we present approaches that solve these issues.

[2]Note that these discs are not necessarily of infinite size, as a cut-off distance can be set once the influence becomes negligible.

Table 8.1: Wake modelling: symbol definitions

Number of turbines	n
Wind velocity Wind direction	v $0^0 < \theta < 360^0$
Farm radius	r
Rotor diameter	R
Weibull distribution for wind speed Weibull shape parameter Weibull 1scale parameter Wind direction distribution	$p_v(v,k,c) = \frac{k}{c}\left(\frac{v}{c}\right)^{k-1} e^{-(v/c)^k}$ k c $P(\theta)$
Expected power of a single turbine i	$E^i[\eta]$
Piecewise power curve of turbine	$\beta(v) = \begin{cases} 0 & v < v_{cut\_in} \\ \lambda v + \gamma & v_{cut\_in} \leq v \leq v_{rated} \\ P_{rated} & v_{rated} < v < v_{cut\_out} \end{cases}$

8.3 Wake Modelling

Before we can present our algorithms for breaking the 1000 turbine barrier, we need to establish the conditions under which we evaluate the layouts. We follow Kusiak and Song [51] and formulate the layout problem as follows. Let $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_n\}$ be the x and y coordinates of the n turbines.

Our goal is to identify a layout that maximises the energy capture from a given farm

$$\arg\max_{(X,Y)} \eta(X, Y, v, \beta(v)) \tag{8.1}$$

where v is the wind speed, and the function $\beta(v)$, known as a power curve, gives the power generated by a specific turbine for a given wind speed. Wind speed v however is a random variable with a Weibull distribution, $p_v(v, c, k)$, which is estimated from wind resource data. This distribution also changes as a function of direction, θ which varies from $0^0 - 360^0$, yielding a probability density function for different θ given by $p_v(v(\theta), c(\theta), k(\theta))$. Additionally, wind flows from a certain direction with some probability $P(\theta)$. These different pieces of information are inputs to the algorithm and are summarised in Table 8.1. Due to the random nature of wind velocity, the objective function in Equation 8.1 is transformed to evaluate the *expected* value of the energy capture for a given wind resource and turbine positions. For a single turbine, this value can be calculated using

$$E^i[\eta] = \int_\theta P(\theta) \int_v p_v(v(\theta), c(\theta), k(\theta))\beta^i(v). \tag{8.2}$$

Equation 8.2 evaluates the overall average energy over all wind speeds for a given wind direction, and then averages this energy over all the wind directions. However, during the resource assessment, the wind speed distributions are estimated for discrete wind direction bins. Hence the above integral is discretised along the wind direction. Furthermore, the wind speed is discretised in order to proceed with numerical integration. For more details, we refer the interested reader to [51].

8.3.1 Wake Modelling

The above formulation of expected energy capture, assumes identical wind resources, i.e., $p_v(v(\theta), c(\theta), k(\theta))$ and $P(\theta)$ at each turbine. However, a significant factor that diminishes efficient energy capture is the wake effect: the so-called "down wind exhaust" from one turbine alters the free stream inflow into a turbine behind it. When optimising a layout, the wake affect is calculated as a modification of the estimated wind resource that is available for a turbine i due to its location and the location of other turbines. Like others [51], we make some simplifying assumptions for illustrative purposes in this paper. We use the modified Park wake model [66]. We are aware that, for a large number of turbines, it is not as appropriate as the deep array wake model [6]. The latter could be additionally imposed without modifications to our algorithms. The procedure for the evaluation of the wake effects due to the Park model is shown below in Algorithm 8.1.

As can be seen from Algorithm 8.1, the wake effects on a turbine i change the wind resource available to it along different directions by reducing the *scale* parameter of the Weibull distribution estimated for the entire farm, which is also called the freestream wind resource. This is dependent on its location and the location of the rest of the turbines. Hence, Equation 8.2 is modified to reflect this to

$$E^{farm}[\eta] = \sum_i \int_\theta P(\theta) \int_v p(v(\theta), c_i(\theta, X, Y), k(\theta))\beta^i(v). \tag{8.3}$$

In this equation v is the wind speed, and the function $\beta^i(v)$ defines the power curve for turbine i. Wind speed v however is a random variable with a Weibull distribution, $p(v(\theta), c_i(\theta, X, Y), k(\theta))$, which is estimated from wind resource data and considers the wake effect using X and Y. This distribution is also a function of the wind direction, θ which varies from $0^0 - 360^0$. Note that the *shape parameter* of the Weibull distribution is not influenced by the Wake effects here. Additionally, wind flows from a certain direction with some probability $P(\theta)$.

The goal of the optimisation problem is to maximise Equation 8.3. In the following subsection, we present the constraints and assumptions we made for the optimisation problem.

8.3.2 Constraints and Assumptions

The first constraint enforces an upper bound on the area of the farm. This constraint ensures that we can only place a turbine within a certain area, which is a realistic constraint for most layout problems. For a circular farm with radius r and the origin as the centre, this constraint is satisfied *iff*

$$sqrt(x_i^2 + y_i^2) \leq r, \forall i. \tag{8.4}$$

For a rectangular farm with length l and width w this constraint is satisfied *iff*

$$0 \leq x_i \leq l \;\&\; 0 \leq y_i \leq w, \forall i. \tag{8.5}$$

The second constraint regulates the spatial proximity, as it dictates the minimal distance within which two turbines can be set up. It is satisfied *iff* $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \geq MR, \forall i \forall j$ where R is the rotor radius and M is a proximity factor usually decided ahead of the optimisation based on the make and model of the turbines used. We use $M = 8$ based on the industry standard.

In addition to the above constraints, we assume that all turbines have the same power curves (approximated as piecewise linear functions) and that the same wind resource spans the entire farm.[3] The assumptions can be revised in a very straight forward manner to generate more realistic scenarios.

Note that our implementation of the Park wake model and of the constraint checker is publicly available, in order to facilitate comparability of future results: `http://cs.adelaide.edu.au/~ec/research/windfarmlayout.php`

[3]To increase accuracy, these resources can be estimated for different parts in the farm.

Algorithm 8.1: Procedure for evaluation of wake effects due to the Park model [51]

```
1  Given $\{X, Y\}$ as turbine locations, turbine thrust coefficient $C_T$, rotor diameter
   $R$, landscape-specific wake spreading factor $\kappa$ ;
2  $a = 1 - \sqrt{1 - C_T}$, $b = \kappa/R$, $u \Leftarrow$ unit step function,
   $o = (x_i - x_j)cos\theta + (y_i - y_j)sin\theta$;
3  $d_{i,j} = \|o\|$, $\alpha = tan^{-1}\kappa$;
4  for $i = 1$ to number of turbines do
5      for $\theta = 0^0$ to $360^0$ do
6          for $j = 1$ to n-1 and $j \neq i$ do
7              $\delta_{i,j} = cos^{-1}\left(\frac{o+R/\kappa}{\sqrt{(x_i-x_j+\frac{R}{\kappa}cos\theta)^2+(y_i-y_j+\frac{R}{\kappa}sin\theta)^2}}\right)$;
8              $Vdef_{(i,j)} = u(\delta_{i,j} - \alpha)\frac{a}{(1+bd_{i,j})^2}$;
9          $Vdef_i^\theta = \sqrt{\sum_j (Vdef_{(i,j)}^\theta)^2}$;
10         $c_i(\theta) = c_i(\theta) \times (1 - Vdef_i)$;
```

I hear, I know.
I see, I remember.
I do, I understand.

Confucius

9 Optimising the Layout of 1000 Wind Turbines

For the layout of hundreds, then 1000, turbines, we demonstrate an accurate, efficient, and parallelisable optimisation algorithm in this chapter. It is modular and therefore allows different wake effect models to be incorporated. Its computational cost is a relation that depends upon how many candidate layouts it investigates and the complexity of its wake loss calculation. We demonstrate how well it maximises energy capture and show how it allows one to examine how wake loss scales with energy capture and number of turbines.

9.1 CMA-Evolutionary Strategy

The Covariance Matrix Adaptation based evolutionary strategy (CMA-ES) was developed by Hansen [43] and it forms the basis of our first approach. It is summarised in Algorithm 9.1. Over the course of an optimisation, CMA-ES self-adapts the covariance matrix of a multivariate normal distribution in order to guide the search. This normal distribution is then used to sample from the multidimensional search space where each variate is a search variable. The co-variance matrix allows the algorithm to respect the correlations between the variables making it a powerful evolutionary search algorithm. Consider a representation x_k for the k^{th} solution to the optimisation problem that

Algorithm 9.1: Covariance Matrix Adaptation Based Evolutionary Strategy [43]

```
1 for $t = 1$ to maxiter do
2     Sample $\mathbf{x}_i^{(t)}$ using Equation 9.6, evaluate energy capture for $\mathbf{x}_i, \forall i$
3     Select $\mu$ members of the population
4     Update the mean using
          $$\mathbf{m}^{(t+1)} = \sum_{i=1}^{\mu} w_i \mathbf{x}_i^{(t+1)}, \text{ such that } \sum_{i=1}^{\mu} w_i = 1 \text{ and } w_i > 0 \quad (9.1)$$
5     Update standard deviation $\sigma^{(t+1)}$ using
          $$\mathbf{p}_\sigma^{(t+1)} = (1 - c_\sigma)\mathbf{p}_\sigma^{(g)} + \sqrt{(c_\sigma(2 - c_\sigma)\mu_{eff}}\mathbf{C}^{(t)\frac{-1}{2}} \frac{\mathbf{m}^{(t+1)} - \mathbf{m}^{(t)}}{\sigma^{(t)}} \text{ and} \quad (9.2)$$
          $$\sigma^{(t+1)} = \sigma^{(t)} exp(\frac{c_\sigma}{d_\sigma}(\frac{||\mathbf{p}_\sigma^{(t+1)}||}{E||\mathcal{N}(0, I)||} - 1)) \quad (9.3)$$
6     Update covariance matrix using
          $$\mathbf{p}_c^{(t+1)} = (1 - c_c)\mathbf{p}_c^{(t)} + h_\sigma^{(t+1)} \sqrt{c_c(2 - c_c)\mu_{eff}} \frac{\mathbf{m}^{(t+1)} - \mathbf{m}^{(t)}}{\sigma^{(t)}} \quad (9.4)$$
          $$\mathbf{C}^{(t+1)} = (1 - c_{cov})\mathbf{C}^{(t)} + \frac{c_{cov}}{\mu_{cov}}(\mathbf{p}_c^{(t+1)}\mathbf{p}_c^{(t+1)^T}) + c_{cov}(1 - \frac{1}{\mu_{cov}})\mathbf{C}_\mu^{(t+1)} \quad (9.5)$$
```

attempts to minimise the objective function $f(x)$. In each iteration, t, the algorithm samples λ number of solutions from a multivariate normal distribution given by

$$\mathbf{x}_k^{(t+1)} = \mathcal{N}(\mathbf{m}^{(t)}, (\sigma^{(t)})^2\mathbf{C}^{(t)})\forall k. \tag{9.6}$$

where $\mathbf{m}^{(t)}$ is the mean, $\sigma^{(t)}$ is the standard deviation and $\mathbf{C}^{(t)}$ is the covariance matrix for a multivariate normal distribution represented by $\mathcal{N}$. t represents the iteration index. The goal of the algorithm is to then adapt $\mathbf{m}$, σ and $\mathbf{C}$ as optimisation progresses. The simplest type of adaptation can be achieved by selecting a subset of μ solutions that perform the best in terms of the objective function, and estimating the parameters of the multivariate normal distribution based on these solutions. This can be simply done using

$$\mathbf{C}_\mu^{(t+1)} = \sum_{i=1}^{\mu} w_i \frac{\left(\mathbf{x}_i^{(t+1)} - \mathbf{m}^{(t)}\right)}{\sigma^{(t)}} \left(\frac{\mathbf{x}_i^{(t+1)} - \mathbf{m}^{(t)}}{\sigma^{(t)}}\right)^T \tag{9.7}$$

More sophistication to this adaptation can be added as shown in Equation 9.5, such as using weighted sums of the matrices, and the adaptation of the step-size.
Rank μ update: This is summation of two terms, i.e., Equation 9.7 weighted by $c_{cov}(1 - \frac{1}{\mu_{cov}})$ and $(1 - c_{cov})(\mathbf{C})^{(t)}$. Thus this generates a weighted sum of covariance matrix from previous iteration, and the estimate of the covariance from μ best performing samples in the current iteration.
Cumulation: This captures the direction of the movement of mean as iterations progress. This is calculated using Equation 9.4 and setting $\mathbf{p}_c^{(1)} = 0$ initially. Note that the contribution of the previous iterations is controlled using a weight $1 - c_c$. $h_\sigma^{(t+1)}$ is a Heaviside function that stalls the update of $\mathbf{p}_c^{(t)}$ if $||\mathbf{p}_\sigma^{(t+1)}||$ is large [43].
Step size control: This provides a mechanism to control the variation in $\sigma^{(t)}$. To achieve this an evolution path $\mathbf{p}_\sigma$ is evaluated using eigen decomposition of $\mathbf{C}^{(t)}$, which is $\mathbf{C}^{(t)\frac{-1}{2}}$, and the change in the means. This value is then used to determine the new value of $\sigma^{(t+1)}$ as shown in Equation 9.3. $E||\mathcal{N}(\mathbf{0}, \mathbf{I})||$ is the expectation of Euclidean norm of a $\mathcal{N}(\mathbf{0}, \mathbf{I})$.
Table 9.2 summarises the CMA-ES parameters we selected. Initial values are set as follows: $\mathbf{p}_\sigma^{(0)} = \mathbf{0}$, $\mathbf{p}_c^{(0)} = \mathbf{0}$, $\mathbf{C}^{(0)} = \mathbf{I}$. Values for parameters $w_i, c_\sigma, d_\sigma, c_c, \mu_{cov}, c_{cov}$ are set to their default values as described in [43]. For a more detailed intuition about these parameters the reader is referred to [43].
Constraint handling: Our algorithm takes care of the constraints in the following ways. Initially, it places the n turbines on a regular $s \times t$ grid. There, the grid is constructed in such a way that the distance between the rows and columns is maximal, including the placement of turbines on the borders of the wind farm area.[1] This is a straightforward approach of placing a number of turbines within an area, which can be observed to be frequently used in practice. Additionally, this approach proved to serve as a very good starting point, as the initial distance between the turbines is maximised in a naive way. Thus, the wake effect is already reduced to some extent (when compared to tighter layouts), even without considering the directional distribution of the wind.

Furthermore, when turbines violate the wind farm border constraint, we fix such placements by setting these right back onto the borders. This repair significantly reduces the time that is spent on the creation of a new layout, as the high likelihood of violating this constraint would otherwise require a significant number of repeated trials of creation.

Finally, when a layout has a turbine which violates the proximity constraint, we replace the layout entirely with a new, randomly generated feasible layout which allows the optimisation to continue. During our experiments, this circumstance was rare, as the turbines tend to be moved primarily by small distances per iteration.

[1] If $s \cdot t > n$, then the last column is not filled completely.

l	θ^l	θ^{l+1}	Scenario 1			Scenario 2			l	θ^l	θ^{l+1}	Scenario 1			Scenario 2		
			k	c	$P(\theta)$	k	c	$P(\theta)$				k	c	$P(\theta)$	k	c	$P(\theta)$
0	0	15	2	13	0	2	7	0.0002	12	180	195	2	13	0.01	2	10	0.1839
1	15	30	2	13	0.01	2	5	0.008	13	195	210	2	13	0.01	2	8.5	0.1115
2	30	45	2	13	0.01	2	5	0.0227	14	210	225	2	13	0.01	2	8.5	0.0765
3	45	60	2	13	0.01	2	5	0.0242	15	225	240	2	13	0.01	2	6.5	0.008
4	60	75	2	13	0.01	2	5	0.0225	16	240	255	2	13	0.01	2	4.6	0.0051
5	75	90	2	13	0.2	2	4	0.0339	17	255	270	2	13	0.01	2	2.6	0.0019
6	90	105	2	13	0.6	2	5	0.0423	18	270	285	2	13	0.01	2	8	0.0012
7	105	120	2	13	0.01	2	6	0.029	19	285	300	2	13	0.01	2	5	0.001
8	120	135	2	13	0.01	2	7	0.0617	20	300	315	2	13	0.01	2	6.4	0.0017
9	135	150	2	13	0.01	2	7	0.0813	21	315	330	2	13	0.01	2	5.2	0.0031
10	150	165	2	13	0.01	2	7	0.0994	22	330	345	2	13	0.01	2	4.5	0.0097
11	165	180	2	13	0.01	2	9.5	0.1394	23	345	360	2	13	0	2	3.9	0.0317

Table 9.1: Wind Scenario 1 and Scenario 2

9.2 Results and Discussion

We use the wind resources *Scenario 1* and *Scenario 2* as defined in [51] (see Table 9.1). The wind direction is binned in 15^0 intervals. Scenario 1 has the same *scale* and *shape* parameters for the Weibull distribution for all bins. Scenario 2 has the same *shape* parameter for the bins, but different *scale* parameters. The *shape* parameter k increases the spread of the Weibull distribution as it gets larger. In Scenario 1, the dominant wind directions are 75^0 - 90^0 (with $P(\theta) = 0.2$) and 90^0- 105^0 (with $P(\theta) = 0.6$). There is no wind coming from 0^0- 15^0, and 345^0- 360^0. For the rest of the bins the $P(\theta) = 0.01$.

Scenario 2 is more complex and realistic. The *shape* parameter is the same for all bins, however, the *scale* parameter is different for different bins and ranges from 4 - 10. Similarly, $P(\theta)$ also varies over the range 0.001 to 0.1839. It is more difficult to nominally identify competent layouts as there is no prominent wind direction. In Scenario 1 one can optimise for the prominent directions and not lose significant efficiency. In Scenario 2, one has to optimise the layout to work with minimum wake loss along all the wind directions.

The different metrics that we use to evaluate multiple layouts are presented in Table 9.3.

	n=2...9	n=10...100	n=200...500	n=1000
(μ, λ)	(20, 120)	(10, 20)	(10, 20)	(10, 20)
(generations, runs)	(100, 30)	(10000, 30)	(10000, 5)	(20000, 1)
farm size (km)	r=0.5	l=w=3	l=10, w=20	l=10, w=20

Table 9.2: CMA-ES and experiment parameters. Population is expressed with two variables, μ defines the parent population size and λ the number of offsprings generated from the parent population each generation.

Metric	Definition
$\mathcal{E}_{\mathbf{wlf}}$	Wake loss free power
$\mathcal{E}$	power achieved by layout optimiser
$\mathcal{E}_{\mathbf{loss}}$	Power loss due to wakes
$\mathcal{G}_{\mathbf{n}}$	Power gain achieved by layout optimiser via adding n turbines
$\mathcal{G}_{\mathbf{n}}^{\mathbf{wlf}}$	Wake loss free power capture of adding n turbines
$\mathcal{G}_{\mathbf{n}}^{\mathbf{loss}}$	$\mathcal{G}_n^{wlf} - \mathcal{G}_n$

Table 9.3: Metrics used to evaluate multiple layouts.

9.2.1 Results

Case A: 2-6 turbines

In this case, we validate the accuracy of CMA-ES by showing how it is comparable to small scale results of [51]. Each optimisation run of CMA-ES evaluated the same number of candidate layouts as [51] for fairness. Due to the stochastic property of ES, we run the ES multiple times and report 'best of runs' meaning the energy loss of the best layout found when all runs are compared and 'average best', which is the average energy loss of the best layout in each run. Using [51]'s scenarios, the plots of Figure 9.1 show that the Kusiak et al algorithm, called "SPEA-2" [51, 98], and our modified CMA-ES are equivalently effective. We plot the maximum energy capture (before wake loss is subtracted), and the net energy capture (after subtracting wake loss). With 6 turbines, using scenario 2, the energy capture without wake effects would be 43894 kW. SPEA-2's layout loses 698 kW to wake effects and CMA-EA's layout loses, on average 440 kW (approximately 36% improvement).

Case B: 10-100 turbines

We choose Scenario 2 because it is a more complex wind resource. Then we attempt to place 10 to 100 turbines at 10 turbine increments in a 9 km^2 rectangular area.

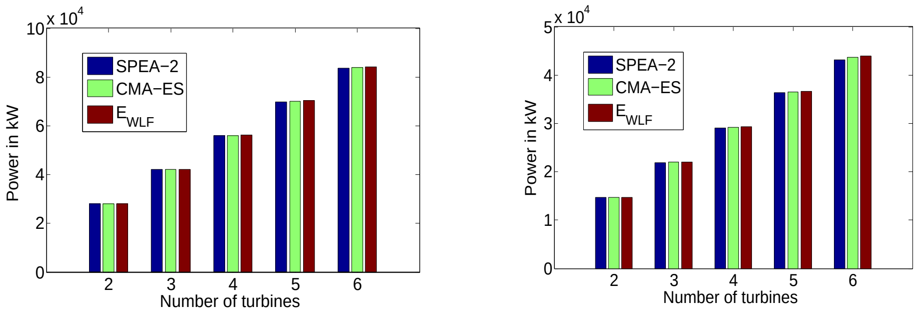


Figure 9.1: Comparison of Kusiak et al.'s SPEA-2 algorithm [51] and our adjusted CMA-ES, left: Scenario 1, right: Scenario 2. The results are not significantly different and are comparable.

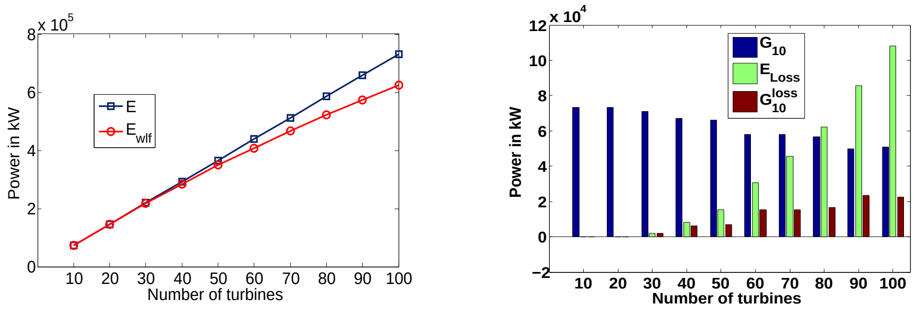


Figure 9.2: Performance of CMA-ES for 10 to 100 turbines under Scenario 2. Left plot shows energy capture climbs up as we add turbines. Right plot shows how adding each new set of 10 turbines helps despite the increase in wake losses.

Figure 9.2(left) shows the energy capture under wake loss. Wake loss is indicated by the gap between max energy and energy capture. Figure 9.2(right) and Figure 9.4(left) show that the value of adding each additional set of 10 turbines slowly declines, while the total wake loss rises considerably. The decline may be explained by additional interference due to squeezing more turbines into the farm. The net energy capture and wake loss rise from 73154 kW and zero respectively with 10 turbines to 619133 kW and 112405 kW respectively with 100 turbines. As in the validation case, packing turbines more tightly into the same area creates higher wake loss. Figure 9.5(left) shows the displacement of 50 turbines from their initial positions at the end of a CMA-ES run. The turbines were placed in a grid initially.At the end of the run the turbines were displaced by a few meters. Figure 9.5(right) summarises the displacements of turbines from their initial positions for 30 independent runs of CMA-ES. The turbines moved 15-20 meters from their initial placements.

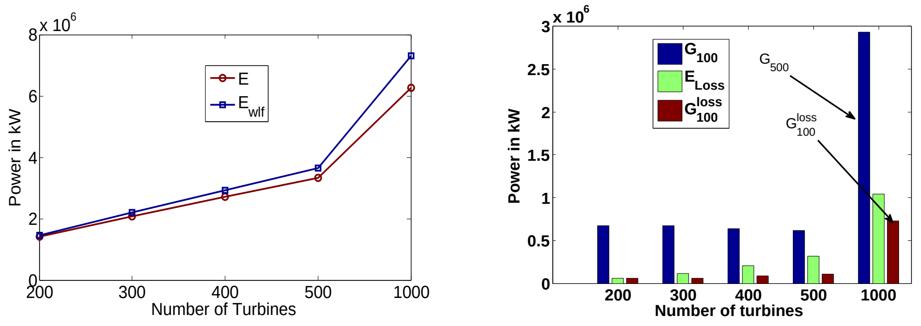


Figure 9.3: Performance of CMA-ES for 200 to 1000 turbines under Scenario 2. Left plot shows energy capture climbs up as more turbines are added. Right plot shows how adding each new set of 100 (500 between $n = 500$ and $n = 1000$) turbines helps despite the increase in the wake losses.

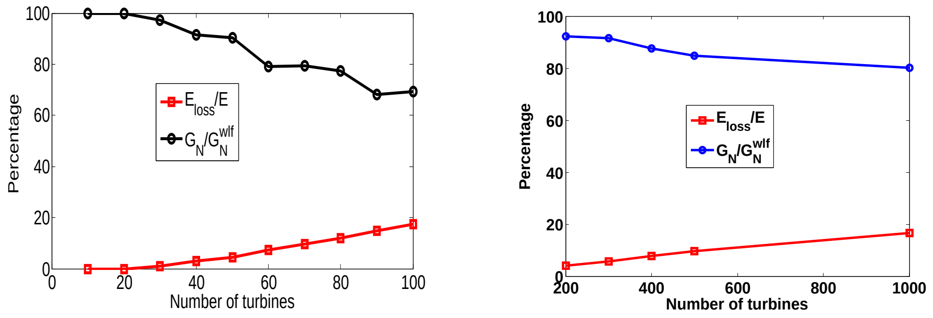


Figure 9.4: (a) The plot shows how the ratio of energy loss due to wake to total capture increases with each additional set of 10 turbines. As well as the gain achieved by adding each additional turbines starts to decrease. This is characteristic of the layout problem when more turbines are squeezed in the same area. (b) This plot shows the same metric evaluated for layouts consisting of 200 -1000 turbines.

Case C: Breaking the 1000 turbine barrier

What happens when 200 to 1000 turbines are located in a rectangle of 200 km^2? Figures 9.3 and Figure 9.4(right) show, for this turbine range, information similar to that of Figures 9.2 and 9.4(left). The net energy capture ascends in sequence $(1440, 2130, 2813, 3465)$ MW when turbine number grows from 200 to 500 by 100 turbine increments. The corresponding wake loss sequence is $(23, 64, 113, 193) MW$. At 1000 turbines, the net energy capture is just less than double that of 500 turbines: 6554 MW because the wake loss rises from 193 MW to 761 MW. The non-linear trend in wake loss, again, arises from packing more turbines into the same area.

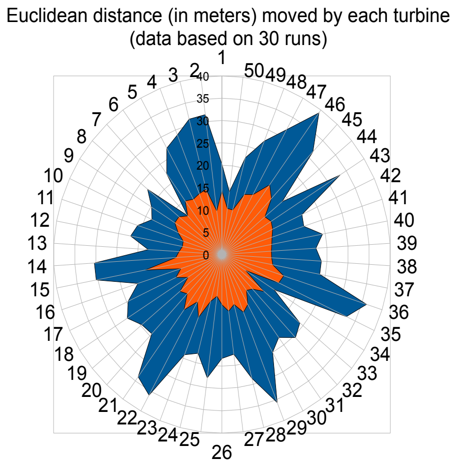


Figure 9.5: Mean and standard deviation of displacement of turbines for 30 independent runs of CMA-ES (right)

9.2.2 Cost of the algorithm

One metric of evaluation is elapsed time. We ran a parallelised version of our modified CMA-ES on 20 cores when running large layouts (for 1000 turbine problem). This is crucial because just one wake loss calculation for a 1000 turbine layout takes about 30 seconds (on an Intel Xeon E7530, 1.87GHz). A realistic version of this optimiser would account for many additional details, such as cabling costs, but the dominating factor in expense would still be calculating wake loss when evaluating the net energy capture of a layout as other become asymptotically insignificant. The wake loss calculation scales quadratically with the number of turbines. The cost of one layout evaluation must be multiplied by the total layout evaluations run by the optimiser. For CMA-ES this latter factor is the product of offspring pool size, μ, and the number of generations. For example, each layout of 1000 turbines, on average requires 30 seconds to evaluate net energy capture. If we run CMA-ES with an offspring pool of 20 for 20000 generations so the run requires, serially, roughly 12,000,000 CPU seconds or about 140 days. With parallelisation, the elapsed time of the optimisation was approximately 12 days implying the speedup is sub-linear. To optimise 200 and 500 turbines serially, it would have taken about 3 and 19 days respectively but, with parallelisation on 2 processors, this averaged to 1.3 and 13 days.

9.3 Conclusions

In this chapter, we have presented an advanced evolutionary algorithmic approach that learns the statistical properties of the better layouts and makes use of them to generate even better layouts. This property is advantageous for layout optimisation because the optimal position of a turbine depends upon its neighbours' positions.

We demonstrated the algorithm on layout problems involving 100's and even 1000 turbines. The stochastic nature of the algorithm demands performing multiple independent trials. One fortuitous feature of this requirement is that the multiple trials provided different layouts that were equally competent in their energy capture. This allows one to late impose additional objectives to make a selection among these competent designs. The algorithm was parallelised on multiple cores to achieve significant speed-ups.

This work has been published in the proceedings of the European Wind Energy Association Event 2011 [89].

The pessimist complains about the wind; the optimist expects it to change; the realist adjusts the sails.

William Arthur Ward

10 A Problem Specific Local Search Method

OUR FIRST APPROACH TO THE PROBLEM PERFORMED REASONABLY WELL, but was computationally expensive. In the following, we describe the design of a fast and effective randomised local search algorithm to act as a farm layout optimisation module. This new algorithm

1. uses a fast approach for evaluating new layouts,
2. makes effective use of the problem characteristics to produce new layouts, and
3. can efficiently handle infeasible regions in order to respect geographical constraints.

To be precise, where previous algorithms required time $\Omega(n^2)$ for the computation of the wake effects within a new layout, the algorithm that we present in this chapter requires $\Theta(kn)$, where n is the number of turbines on the wind farm, and k the number of turbines whose location was changed over the previous layout.

The new local search algorithm is capable of generating good layouts for hundreds of turbines quickly on standard computers, without the need of specialised computing hardware. This allows it to accommodate the emerging requirements of larger farms. For example, the Horse Hollow Wind Energy Center in Texas, USA operates with 735.5 megawatt capacity and consists of more than 300 turbines spread over nearly 47,000 acres (190 km^2). We compare our new algorithm to two approaches: the CMA-ES approach presented in Chapter 9 and the local search approach used in the industry tool AWS OpenWind. Our experiments show that the new algorithm outperforms both approaches in terms of the quality of the results and the running time.

In this chapter, we proceed as follows. Section 10.1 describes our algorithm. Section 10.2 gives details about our experiments, and discusses the outcomes. Then, the capability to deal with infeasible areas is shown, using a scenario based on an existing near-shore wind farm. Finally, Section 10.4 summarises our findings and outline potential future work. Note that the optimisation task at hand is identical to that of Chapter 8, and we refer to reader to said chapter for the description of the wind farm layout problem.

10.1 Turbine Distribution Algorithm

In the following, we describe our algorithm for the optimisation of wind farm layouts. It includes a problem specific local search operator and a faster evaluation, which makes use of the fact that we always change the position of exactly one turbine.

10.1.1 Computational Speedup

First, we describe how computation time can be saved, if the new layout differs from the old one only in the location of a single turbine.

For the computation of the (turbine-specific) wind resources, the Weibull scale parameters are adjusted up to $24 \times n^2$ times (line 10 of Algorithm 8.1, Section 8.3.1), as the wind direction is discretised into 24 wind directions in our model, and the mutual influence of the n turbines has to be considered. The computation of the influence matrix results in an evaluation time that is quadratic with the number of turbines, causing evaluation times of up to 30 seconds on standard hardware for a 1000 turbine layout.[1]

As our optimisation algorithm modifies only a single turbine of the current layout, the evaluation can be speeded up by updating the velocity deficits in line 10 of Algorithm 8.1 intelligently. First, for a moved turbine i the influence of all other turbines on i has to be computed conventionally, requiring n-1 influence checks. Second, the other turbines' velocity deficits have to be updated as i's influence on these may have changed. This can be done as follows. For each such unmoved turbine j:

$$Vdef_j^{\theta,NEW} = \sqrt{\left(Vdef_j^{\theta,OLD}\right)^2 - \left(Vdef_{(j,i)}^{\theta,OLD}\right)^2 + \left(Vdef_{(j,i)}^{\theta,NEW}\right)^2} \qquad (10.1)$$

where $Vdef_j^{\theta,OLD}$ is the velocity deficit the turbine j experiences in the old layout, and $Vdef_{(j,i)}^{\theta,OLD}$ is the influence that i had on j, for a given wind direction θ.

[1]This test and all subsequent experiments were performed on AMD Opteron 250 CPUs (2.4GHz), on Debian GNU/Linux 5.0.8, with Java SE RE 1.6.

Over the course of the algorithm's run, we only have to initially create the three dimensional array carrying all mutual influences for all wind directions once. In subsequent evaluations, we can use and update this matrix to speed-up the evaluations. Generalised, this allows for a resulting runtime for such a subsequent evaluation of only $\Theta(kn)$, when the locations of k out of n turbines were changed.[2]

10.1.2 The Algorithm

It is clear that the constraints discussed in Section 8.3.2 are vital to the construction of the algorithm. To ensure constraint handling in the optimisation, a random local search algorithm was purpose-built for the application. The turbine distribution algorithm (TDA) described in Algorithm 10.1 and Figure 10.1 iteratively displaces a single turbine in order to increase the energy gain while ensuring constraints are upheld.

In order to ensure that the turbines' initial placement respects the safety distance constraint, our TDA follows the approach taken in Chapter 9. There, the algorithm deterministically initializes the turbines in a grid formation of greatest space. The grid is constructed in such a way that the distance between the columns and rows is maximised, including the placement of turbines on the borders of the wind farm area. This is a straightforward approach, which is used frequently in practice, as the wake effect is already reduced to some extent (when compared to tighter layouts), even without considering the directional distribution of the wind.

Over the course of the optimisation, new layouts are created based on the best-so-far turbine configuration. For the next layout, a copy of the best-so-far is made, and a modification is then applied, in which a single turbine is selected uniformly at random and is shifted by a displacement vector $\vec{v}'$. In the following, we motivate our way to compute this displacement vector.

Initially, the vector $\vec{v}$ is determined by the normalised sum of the difference in distance between the current turbine and its nn nearest neighbours. The displacement of the current turbine by $\vec{v}$ would move the turbine away from the nearest neighbours. In order to ensure non-deterministic performance within sensible limits, TDA applies a normal distribution with a fixed standard deviation to the direction of the vector and uses an adaptable standard deviation on a 0 mean over a normal distribution to determine the distance for the vector. As a superior energy output for the wind farm could be obtained by moving turbines closer together, this new vector may be randomly reversed with a preset probability.

In the next step, the new location of the turbine after displacement by the resultant vector $\vec{v}'$ is investigated to ensure that it would not be placed in an illegal position, currently defined as outside the wind farm bounds or too close to another turbine. If

[2]We use $k = 1$ in our experiments, which results in a significant speed-up (see Table 10.1).

Algorithm 10.1: Turbine Distribution Algorithm (TDA)

```
1  Given number of turbines $n$, map bounds $x_{max}$, $y_{max}$, nearest neighbours $nn$,
   reversal probability $p$, displacement distance standard deviation $\sigma_{\{dis\}_k}$ for
   $k \in \{1, \dots, n\}$, and direction standard deviation $\sigma_{\{dir\}_k}$ for $k \in \{1, \dots, n\}$, place
   $\{X, Y\} = \{[0, \dots, x_{max}], [0, \dots, y_{max}]\}^n$ in the grid formation of greatest space;
2  Determine $f(\{X, Y\})$ as per Equation 8.3;
3  for $i = 1$ to number of evaluations do
4      Set $\{X, Y\}' = \{X, Y\}$;
5      Select turbine $k$ uniformly at random for the $i^{th}$ modification, denote as
       $\{x'_k, y'_k\}$;
6      Determine nearest neighbours: $\{x''_1, y''_1\}, ..., \{x''_{nn}, y''_{nn}\}$;
7      $\vec{v} = \left(\sum_{j=1}^{nn} x'_k - x''_j, \sum_{j=1}^{nn} y'_k - y''_j\right)$;
8      $\vec{v} = \vec{v}/||\vec{v}||$;
9      Select $\theta$ normally distributed over $\mu = \angle\vec{v}$ and $\sigma = \sigma_{\{dir\}_k}$;
10     Select $d$ normally distributed over $\mu = 0$ and $\sigma = \sigma_{\{dis\}_k}$;
11     $\vec{v}' = \theta * d$;
12     Set $\vec{v}' = -\vec{v}'$ with probability $p$, as tighter groups may increase the farm's
       output;
13     If applying $\vec{v}'$ would place $\{x'_k, y'_k\}$ in illegal area, reduce length of $\vec{v}'$ until
       legal;
14     Displace $\{x'_k, y'_k\}$ by $\vec{v}'$;
15     Determine $f(\{X, Y\}')$ as per Equations 8.3 and 10.1;
16     if $f(\{X, Y\}) \leq f(\{X, Y\}')$ then
17         Set $\{X, Y\} = \{X, Y\}'$;
18         Increase $\sigma_{\{dis\}_k}$;
19     else
20         Decrease $\sigma_{\{dis\}_k}$;
```

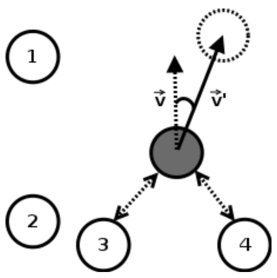


Figure 10.1: Illustration of vector displacement using TDA with $nn = 2$.

the turbine would be in an illegal position, the distance of $\vec{v}'$ is reduced to a legal value. An example of this displacement using $nn = 2$ may be seen in Figure 10.1. Note that, in this example, the two nearest neighbours of the current turbine are turbines 3 and 4, and hence these are the only turbine locations that affect the new position of the current turbine.

At the conclusion of this layout modification, the quality of the new layout is compared against that of the best-so-far, using Equation 8.3 and the computational speedup presented above. If the new potential solution is of higher quality, it replaces the previous best.

Furthermore, when the quality of the layout improves due to the displacement of a turbine, the above-mentioned adaptable standard deviation for the distance of that particular turbine increases to allow for increased exploration. Similarly, when the new layout is of lower quality due to the displacement of a turbine, the standard deviation of this particular turbine decreases to allow displacement exploitation. Effectively, this gives TDA the ability to autonomously switch between exploration and exploitation. There will be local optima due to the local search properties of the algorithm, and our strategy to increase the potential of escape is the combination of n adaptable displacement parameters, in combination with values drawn randomly from different normal distributions.

10.2 Experimental Investigations

In order to justify our design decisions, we performed experimental investigations on which we report in the following. First, we introduce the scenario that defines the wind resource present at an imaginary prospective site of a wind farm. Then, we describe TDA's parameter settings and the test scenarios for the subsequent comparative study. There, we compare the solution quality and runtime of our algorithm to a tuned version from Chapter 9 and the industry tool AWS OpenWind [3].

10.2.1 Algorithm Settings and Scenario Information

To evaluate our algorithm's performance we set up a realistic scenario with the wind resources defined as *Scenario 2* in [51] (see Table 9.1), which allows for a direct comparison with the results reported in Section 9.2. Again, the prevailing wind direction covers a broad sector of about 105°, and the wind intensity per direction is given by Weibull distributions. This results in non-zero probabilities for wind coming from any direction, and therefore, one has to optimise the layout to work with minimum wake loss along all the wind directions.

For the subsequent experiments, the internal parameters of our algorithm were set as follows.

The direction standard deviation is set to $\sigma_{\{dir\}_i} = \pi/6$ for all i turbines. Thus, the resulting displacement direction will be within $\pm 30^0$ with a probability of 68.2% of all

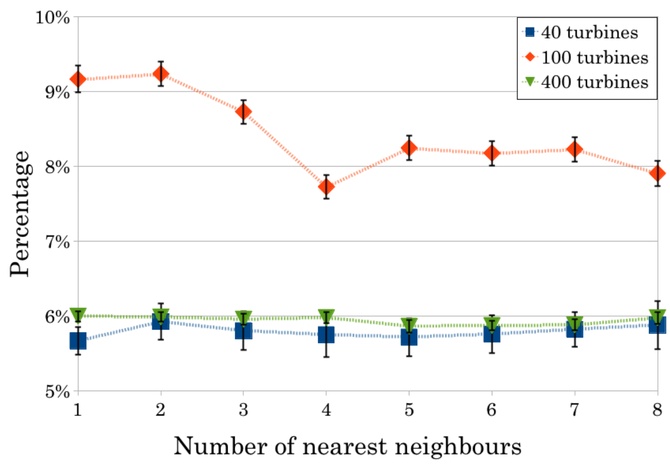


Figure 10.2: Study: influence of the number of considered nearest neighbours on the performance. Shown is the energy gain and its standard deviation achieved over the initial layout.

directional adjustments, and within $\pm 60^0$ with a probability of 95.4%. Furthermore, the reversal probability of the displacement was set to $p = 0.2$.

The displacement distance standard deviation $\sigma_{\{dis\}_i}$ is set and adjusted by the algorithm for each turbine individually, depending on the initial layout. If the minimal initial distance between turbines is d, then $\sigma_{\{dis\}_i} = (d - 8R)/3$. Thus, we restrict the displacements to small distances, while considering the safety distance of $8R$ between the turbines.

As described in Section 10.1, these parameters are adjusted over the course of the optimisation depending on whether recent displacements were successful or not. This allows for an efficient exploration as well as exploitation. The number of turbines n, the map bounds x_{max}, y_{max}, and the number of nearest neighbours nn are specific to the scenario.

10.2.2 Impact of the Nearest Turbines

In order to understand the influence of the number of nearest neighbouring turbines nn that influence the relocation of a chosen turbine, we ran an initial set of experiments. We chose the scenarios with 40, 100, and 400 turbines, and ran each experiment with a budget of 10,000 evaluations. nn varied from 1 to 8, and the results of 100 repetitions per scenario are shown in Figure 10.2.

We noticed that the number of nearest neighbours sometimes has some influence on the performance of the algorithm. Depending on the scenario, a smaller value of nn seems to be beneficial. For example for the 100 turbines scenarios, using $nn = 2$ instead of $nn = 8$ yielded in layouts with an increased energy output of 1.1% on average. For the 40 turbine scenario, however, the maximum distance in the averages is only 0.6%, and for the 400 turbines just 0.1%. A possible explanation is that with four and eight neighbours, the influence of two opposing neighbouring turbines is cancelled out: the resulting displacement vector for a selected turbine just has a ‘unspecific’ direction, instead of an ‘explorative’ direction as is the case when only one or two neighbours are considered.

10.2.3 Experimental Results

To assess the performance of our algorithm, we compare it directly with the results from Chapter 9 and the approach implemented in AWS OpenWind. In our computational study, we set up several scenarios with varying numbers of turbines, and varying farm sizes. For $n = 10, 20, \ldots, 100$ a quadratic farm of $3 \times 3km^2$ was chosen, and for $n = 200, 300, 400, 500, 1000$ rectangular farms of $8 \times 5km^2$, $10 \times 6km^2$, $12 \times 6km^2$, $14 \times 7km^2$, and $20 \times 10km^2$. Note that these wind farm dimensions differ from those listed in Table 9.2. The new dimensions are chosen such that roughly $2n$ turbines fit on the available land. With the setups of Table 9.2, a lot more land is available and thus the wake effects are a lot less dominant, which result in less challenging optimisation problems.

To allow for a better comparison with the CMA-ES algorithm presented in Chapter 9, where 10,000 generations with a total of 200,000 evaluations were performed, we ran our algorithm for 10,000 and 200,000 evaluations—we stopped the algorithms if no improvement was reached after 1,000 evaluations. In order to compare our TDA not only to an academic system, we additionally implemented the optimiser component that is part of the industrial tool AWS OpenWind, and ran it for a maximum of 200,000 evaluations. Quickly, we noticed that given the wide range of scenarios, the original CMA-ES fails to produce consistent results—in particular, it performed poorly for 70 and 80 turbines. Despite the fact that it has elaborate adjustment capabilities, it is not always capable of learning the problem-specific features of the solution space within reasonable run-times. Through the increase of CMA-ES’s initial standard deviation, the explorative phase was extended, and we were able to increase its quality performance on these two particular scenarios to a level comparable with TDA. In the following, we refer to this tuned version as CMA-ES*.

The initial layout, where turbines are placed in a maximally spaced grid, was identical for all algorithms. Each scenario was repeated 30 times, and the results are listed in Table 10.1 and in Figure 10.3.[3] As can be seen, our algorithm’s results (when using only 10,000 evaluations) are comparable to those of CMA-ES*, and in many cases have a higher solution quality. Additionally, our algorithm just uses a fraction of the evaluations and of the time to reach the results. Exemplarily, the academic CMA-ES*

[3]In the scenarios with 10 and 20 turbines, the turbines are placed very far away from each other, with just minimal wake effects occurring.

Table 10.1: Results. Reported are the predicted energy outputs in kW (average and maximum values, and the standard deviation), time in hours. The algorithm with the best average energy output has been highlighted in bold, the second-best in italics.

		CMA-ES* 200k			OpenWind 200k		
n	Initial	$\text{avg}_{\text{stdev}}$	max	time	$\text{avg}_{\text{stdev}}$	max	time
10	7.290E+4	*7.306E+4*$_{3.63E+1}$	7.315E+4	0.10	7.290E+4$_{0.00E+0}$	7.290E+4	0.01
20	1.448E+5	**1.448E+5**$_{0.00E+0}$	1.448E+5	0.05	**1.448E+5**$_{0.00E+0}$	1.448E+5	0.01
30	2.015E+5	2.096E+5$_{1.73E+2}$	2.100E+5	1.29	2.052E+5$_{1.17E+3}$	2.067E+5	0.01
40	2.630E+5	2.742E+5$_{2.07E+2}$	2.746E+5	2.01	2.671E+5$_{8.08E+2}$	2.688E+5	0.02
50	3.247E+5	3.367E+5$_{3.01E+2}$	3.372E+5	3.82	3.269E+5$_{7.28E+2}$	3.282E+5	0.03
60	3.688E+5	3.947E+5$_{3.47E+2}$	3.953E+5	5.45	3.837E+5$_{1.19E+3}$	3.858E+5	0.04
70	4.341E+5	4.435E+5$_{8.98E+3}$	4.559E+5	5.90	4.370E+5$_{9.03E+2}$	4.389E+5	0.07
80	4.707E+5	*5.077E+5*$_{1.02E+3}$	5.098E+5	9.78	4.884E+5$_{1.18E+3}$	4.906E+5	0.08
90	5.207E+5	5.541E+5$_{1.22E+3}$	5.567E+5	9.64	5.333E+5$_{1.02E+3}$	5.356E+5	0.10
100	5.535E+5	5.994E+5$_{2.18E+3}$	6.029E+5	14.0	5.762E+5$_{1.48E+3}$	5.784E+5	0.14
200	1.248E+6	1.301E+6$_{4.36E+2}$	1.302E+6	40.1	1.273E+6$_{2.63E+3}$	1.278E+6	0.39
300	1.861E+6	1.935E+6$_{7.76E+2}$	1.937E+6	111	1.893E+6$_{2.77E+3}$	1.898E+6	0.87
400	2.415E+6	2.549E+6$_{1.02E+3}$	2.550E+6	221	2.480E+6$_{2.20E+3}$	2.485E+6	1.46
500	3.062E+6	3.196E+6$_{9.63+E3}$	3.201E+6	324	3.118E+6$_{3.55E+3}$	3.124E+6	2.26
1000	6.023E+6	6.298E+6$_{2.36E+4}$	6.335E+6	327	6.202E+6$_{4.81E+3}$	6.210E+6	9.43

	TDA 10k			TDA 200k				
n	$\text{avg}_{\text{stdev}}$	max	time	$\text{avg}_{\text{stdev}}$	max	time	P_{loss}	P_{gain}
10	7.305E+4$_{1.73E+1}$	7.308E+4	0.01	**7.309E+4**$_{2.47E+1}$	7.314E+4	0.22	0.1%	100.0%
20	**1.448E+5**$_{7.05E-1}$	1.448E+5	0.02	**1.448E+5**$_{1.30E+1}$	1.449E+5	0.50	1.0%	98.1%
30	*2.123E+5*$_{3.74E+2}$	2.131E+5	0.04	**2.135E+5**$_{4.08E+2}$	2.144E+5	0.75	2.7%	92.2%
40	*2.772E+5*$_{4.46E+2}$	2.781E+5	0.05	**2.791E+5**$_{5.75E+2}$	2.806E+5	1.02	4.6%	88.7%
50	*3.392E+5*$_{4.61E+2}$	3.401E+5	0.06	**3.412E+5**$_{3.42E+2}$	3.418E+5	1.29	6.7%	84.8%
60	*3.980E+5*$_{5.25E+2}$	3.991E+5	0.08	**4.011E+5**$_{5.25E+2}$	4.022E+5	1.58	8.6%	80.4%
70	*4.512E+5*$_{1.19E+3}$	4.537E+5	0.09	**4.555E+5**$_{1.57E+3}$	4.591E+5	1.86	11.1%	72.8%
80	5.044E+5$_{1.15E+3}$	5.083E+5	0.11	**5.090E+5**$_{1.01E+3}$	5.108E+5	2.15	13.0%	72.8%
90	*5.548E+5*$_{1.34E+3}$	5.569E+5	0.12	**5.609E+5**$_{9.38E+2}$	5.625E+5	2.52	14.8%	68.9%
100	*6.015E+5*$_{1.32E+3}$	6.041E+5	0.14	**6.083E+5**$_{1.28E+3}$	6.113E+5	2.83	16.9%	63.9%
200	*1.309E+6*$_{8.99E+2}$	1.311E+6	0.33	**1.323E+6**$_{9.16E+2}$	1.325E+6	6.73	9.6%	96.8%
300	*1.949E+6*$_{1.37E+3}$	1.952E+6	0.55	**1.971E+6**$_{9.72E+2}$	1.973E+6	11.3	10.2%	87.4%
400	*2.553E+6*$_{1.52E+3}$	2.557E+6	0.84	**2.584E+6**$_{1.12E+3}$	2.586E+6	17.0	11.7%	82.6%
500	*3.211E+6*$_{1.66E+3}$	3.215E+6	1.19	**3.249E+6**$_{1.55E+3}$	3.251E+6	24.5	11.2%	90.1%
1000	*6.363E+6*$_{2.49E+3}$	6.368E+6	3.69	**6.449E**+6$_{2.04E+3}$	6.454E+6	75.0	11.9%	86.2%

needs $2h$ for the 40 turbines scenario, while even a better gain is reached within just three minutes by our algorithm. And for the 400 turbines, our algorithm needs less than an hour to achieve the results for which CMA-ES* requires $> 200h$. If given 20 times the number of evaluations, our algorithm is able to produce layouts that produce up to

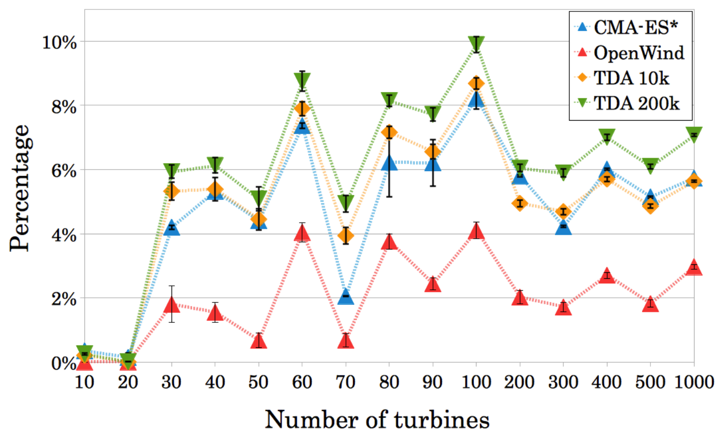


Figure 10.3: Performance comparison: our algorithm TDA (using 10,000 and 200,000 evaluations) versus the tuned academic CMA-ES* and the industry tool OpenWind (both using 200,000 evaluations). Shown is the energy gain achieved over the initial layout, and the standard deviation over the 30 repetitions.

1.4% more energy than the CMA-ES* based algorithm from Chapter 9. In addition, it is significantly faster.

Independent of the scenario, OpenWind's optimiser achieves an average improvement over the initial layouts of only 2%. Responsible for this is most likely the lack of self-adaptation, as the optimiser gets stuck in local optima. This happens very quickly: typically within the first 1,000 layout assessments, which is reflected in the short running times. Contrary to this, our TDA adapts the relocation parameters for each turbine independently of the others. The effects of this ability to change from exploration to exploitation and back is reflected in the large performance advantage of TDA over OpenWind.

To the results, we added some derived statistics. $P_{loss} = \frac{wake\ loss}{energy\ captured}$ denotes the average percentage of unused wind energy due to wake effects. As expected, this loss increases for the scenarios with 10-100 turbines, as more turbines are packed into the same area, inflicting increased wake losses. For 200-1000 turbines, this value is relatively constant, as the chosen sizes of the farms vary such that the farms could contain roughly $2n$ turbines. P_{gain} is the average percentage in energy gained over the previous scenario (i.e. after increasing the number of turbines by 10, 100, or 500). Again, the effect of the increased mutual wakes is reflected in a decreasing benefit of adding turbines, when compared to the next smaller scenario.

10.3 A Real-World Problem: Dealing With Infeasible Areas

It must be noted that the layout of wind farms are generally unable to be denoted simply by a rectangular area specified only by a width and height. Actual wind farms

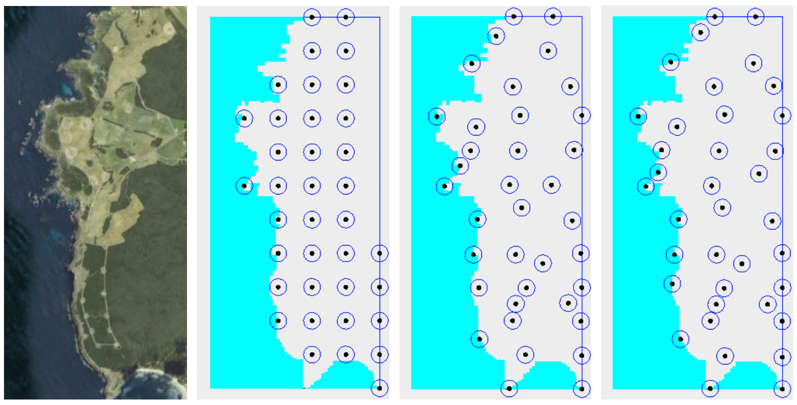


Figure 10.4: Demonstration of the TDA infeasible area modelling capability. The image on the left portrays a satellite image of the Woolnorth wind farm in Tasmania, Australia [41]. The right images are examples of the the loose adaptation used in the modelling tool, and from left to right model the scenario at 0 (252.9 MW), 5,000 (264.91 MW), and 20,000 (265.8 MW) evaluations.

may have uneven boundaries due to proximity to unstable ground, lakes, cliffs, or may simply not have authority or ownership to build in certain locations. These and other geographical constraints may additionally exist within the bounds of the wind farm. Any algorithm that attempts to realistically model a wind farm must model these constraints.

TDA has been adapted to model these infeasible areas, adding an extra constraint to those specified in Section 8.3.2. By modelling these infeasible regions as sets of shapes, any operation which would move a turbine into an infeasible region is corrected in the same manner in which the proximity and farm area constraints are handled above: when the application of the displacement vector $\vec{v}'$ (line 13 of Algorithm 10.1) would place the turbine in an infeasible zone, the distance of $\vec{v}'$ is reduced until the displacement is legal.

This extension has resulted in a capable wind farm modelling tool. In order to demonstrate the ability of the algorithm to model real-world wind farms, Figure 10.4 presents both the satellite image and a loosely modelled representation of the Woolnorth wind farm in Tasmania, Australia (40.685°S 144.717°E). The modelled scenario uses the 37 turbines contained in the north Woolnorth site. However this scenario is merely a proof of concept, as it is not using the wind characteristics of Woolnorth, nor specific internal map or terrain information of the actual site. Only the coastal details have been represented, and the above-mentioned wind resource is used (*Scenario 2* in [51]).

As can be seen, the turbines observe the constraints of the problem and are quickly distributed into a superior formation from evaluation 0 through to evaluation 20,000. As the wind is predominantly from the western direction (between 120° to 225°), the turbines tend to form in staggered north/south columns while leaving space along the east/west directions.

10.4 Conclusion

In this chapter, we have presented a fast and efficient algorithm for the layout optimisation of large wind farms. It takes problem-specific features into account, and benefits from the achieved reduced computational complexity of a layout evaluation when considering the Park wake model. As a result, our algorithm achieves higher quality results than existing approaches, while the assessment speed-up allows for an optimisation within minutes or hours instead of days or weeks (effective speed-up factors of up to 270 were observed).

Although we considered one specific wake model, namely the Park wake model, it is important to note that our optimisation algorithm can be easily applied to other wake models such as the deep array wake model [6].

Potential future work includes the following:

Multiple objectives: So far, our focus was on the optimisation of a single objective, namely the energy output. A natural next step is to extend the optimisation process by incorporating additional objectives, such as minimising the required amount of land and minimising the connecting cables' lengths. These objectives are often in conflict with each other so the goal of solving a such a multi-objective optimisation problem is usually to find a set of compromise solutions.

Parallelization: A rather natural step on the algorithmic side is the parallelisation of the fitness evaluations. As the quality assessments can be carried out in parallel, we plan to make use of current multi-core computing machines in order to further increase the algorithm's speed.

Realistic models: Furthermore, we plan to investigate other wake models, such as the computationally expensive deep array wake model. Potentially, significant speed-ups can be achieved there as well by using our approach of updates. This could then make the use of wake models computationally tractable, without the need for specialised computing servers.

This work has been published in the Renewable Energy Journal in 2013 [90].

Part IV

Summary and Future Work

Prediction is very difficult, especially if it's about the future.

Niels Bohr

Summary and Future Work

Our journey on the theory and applications of bio-inspired algorithms covered theoretical investigations in Part I, theory-motivated algorithm engineering in Part II, and the application of algorithms to real-world problems in Part III.

With Part I of this thesis, we contribute to the understanding of variable-length algorithms with our theoretical and experimental investigations. We show that parsimonious and multi-objective approaches can help algorithms to solve problems that are otherwise unsolvable. Several novel upper bounds for our studied single- and multi-objective scenario from Chapter 4 have recently been proved by Nguyen et al. [67]. The previously best bounds presented in [29, 62] typically depend on two measures, the maximum tree size and the maximum population size, that arise during the optimisation run. In several cases, Nguyen et al. [67] are able to bound both measures. In order to narrow the gap between theory and application over the next couple of years, the investigated problems need to resemble real-world problems more closely. Recently, we have taken the first steps in this direction by starting the analysis of variable-length algorithms when they are used for symbolic regression, which is one of *the* uses of genetic programming.

As we have seen in our theoretical and experimental analyses, the multi-objective optimisation approaches were amongst the most successful ones. In Part II of this thesis, we presented a multi-objective optimisation algorithm whose design is motivated by existing theoretical investigations. The resulting framework for approximation-guided evolutionary algorithms (AGEs) works with a formal notion of approximation and has the ability to work with problems of many dimensions. Our new approximation-guided algorithm called AGE-II efficiently solves problems with few and with many conflicting objectives. The experimental results show that given a fixed time budget AGE-II outperforms state-of-the-art approaches in terms of the desired additive approximation on standard benchmark functions for more than four objectives. On functions with two and three objectives, it lies level with the best approaches. As proven, its computation time increases only linearly with the number of objectives. This enables practitioners now to add objectives with only minor consequences, and to explore problems for even higher dimensions.

Real-world problems are often not only characterised by several objectives, but they also require that the solutions satisfy constraints as well. In addition, the evaluations can be computationally costly. In Part III, we have seen that the single-objective yield optimisation of wind turbine placements on a given area of land is a challenging optimisation problem. This is due to the considered constraints and due to the necessary evaluation efforts. However, with our results in mind, we can consider this problem solved. The next logical step is to tackle the multi-objective variant of this problem. Based on our knowledge gained from tackling the single-objective problem, Tran et al. [81] have already extended our work. They consider the wake effects that are produced by the different turbines on the wind farm, while optimising the energy yield, the necessary area, and the cable length needed to connect all turbines. Tran et al. [81] use amongst others our TDA and our caching-technique to speed-up the computation time. The resulting approach allows the multi-objective optimisation of large real-world scenarios within a single night on a standard computer.

References

[1] R. B. Agrawal and K. Deb. Simulated binary crossover for continuous search space. Technical report, 1994.

[2] A. Auger and B. Doerr. *Theory of Randomized Search Heuristics: Foundations and Recent Developments*. World Scientific, 2011.

[3] AWS Truepower. AWS Openwind, 2008. http://awsopenwind.org/.

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

[5] J. Bader, K. Deb, and E. Zitzler. Faster hypervolume-based search using Monte Carlo sampling. In *Proc. Multiple Criteria Decision Making for Sustainable Energy and Transportation Systems (MCDM '10)*, Vol. 636 of *Lecture Notes in Economics and Mathematical Systems*, pp. 313–326. Springer, 2010.

[6] R. Barthelmie and L. Jensen. Evaluation of wind farm efficiency and wind turbine wakes at the Nysted offshore wind farm. *Wind Energy*, 13, 2010.

[7] N. Beume, B. Naujoks, and M. Emmerich. SMS-EMOA: Multiobjective selection based on dominated hypervolume. *European Journal of Operational Research*, 181: 1653–1669, 2007.

[8] P. Briest, D. Brockhoff, B. Degener, M. Englert, C. Gunia, O. Heering, T. Jansen, M. Leifhelm, K. Plociennik, H. Röglin, A. Schweer, D. Sudholt, S. Tannenbaum, and I. Wegener. Experimental supplements to the theoretical analysis of EAs on problems from combinatorial optimization. In *Proc. 8th International Conference on Parallel Problem Solving from Nature (PPSN VIII)*, Vol. 3242 of *Lecture Notes in Computer Science*, pp. 21–30. Springer, 2004.

[9] K. Bringmann and T. Friedrich. Approximating the volume of unions and intersections of high-dimensional geometric objects. *Computational Geometry: Theory and Applications*, 43:601–610, 2010.

[10] K. Bringmann and T. Friedrich. Approximating the least hypervolume contributor: NP-hard in general, but fast in practice. In *Proc. 5th International Conference on Evolutionary Multi-Criterion Optimization (EMO '09)*, pp. 6–20. Springer, 2009.

[11] K. Bringmann and T. Friedrich. Tight bounds for the approximation ratio of the hypervolume indicator. In *Proc. 11th International Conference Parallel Problem Solving from Nature (PPSN XI)*, Vol. 6238 of *Lecture Notes in Computer Science*, pp. 607–616. Springer, 2010.

[12] K. Bringmann and T. Friedrich. The maximum hypervolume set yields near-optimal approximation. In *Proc. 12th Conference on Genetic and Evolutionary Computation (GECCO '10)*, pp. 511–518. ACM, 2010.

[13] K. Bringmann, T. Friedrich, F. Neumann, and M. Wagner. Approximation-guided evolutionary multi-objective optimization. In *Proc. 21nd International Joint Conference on Artificial Intelligence (IJCAI '11)*, pp. 1198–1203. IJCAI/AAAI, 2011.

[14] S. Cathabard, P. K. Lehre, and X. Yao. Non-uniform mutation rates for problems with unknown solution lengths. In *Proc. 11th Workshop on Foundations of Genetic Algorithms (FOGA 'XI)*, pp. 173–180. ACM, 2011.

[15] C. A. Coello Coello. Evolutionary multi-objective optimization: a historical view of the field. *Computational Intelligence Magazine, IEEE*, 1:28–36, 2006.

[16] C. A. Coello Coello and G. B. Lamont. *Applications of Multi-Objective Evolutionary Algorithms.* World Scientific, 2004.

[17] D. Dasgupta and Z. Michalewicz. *Evolutionary algorithms in engineering applications.* Springer, 1997.

[18] C. Daskalakis, I. Diakonikolas, and M. Yannakakis. How good is the Chord algorithm? In *Proc. 21st ACM-SIAM Symposium on Discrete Algorithms (SODA '10)*, pp. 978–991. SIAM, 2010.

[19] K. Deb. *Multi-objective optimization using evolutionary algorithms.* Wiley, 2001.

[20] K. Deb, A. Pratap, S. Agrawal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions Evolutionary Computation*, 6: 182–197, 2002.

[21] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler. Scalable test problems for evolutionary multiobjective optimization. In *Proc. Evolutionary Multiobjective Optimization (EMO '05)*, Advanced Information and Knowledge Processing, pp. 105–145. Springer, 2005.

[22] I. Diakonikolas and M. Yannakakis. Small approximate Pareto sets for biobjective shortest paths and other problems. *SIAM Journal on Computing*, 39:1340–1371, 2009.

[23] B. Doerr and L. A. Goldberg. Drift analysis with tail bounds. In *Proc. 11th International Conference on Parallel Problem Solving from Nature (PPSN XI)*, Vol. 6238 of *Lecture Notes in Computer Science*, pp. 174–183. Springer, 2010.

[24] B. Doerr and L. A. Goldberg. Adaptive drift analysis. *Algorithmica*, 65:224–250, 2013.

[25] B. Doerr and E. Happ. Directed trees: A powerful representation for sorting and ordering problems. In *IEEE World Congress on Computational Intelligence (WCCI '08)*, pp. 3606–3613. IEEE Computational Intelligence Society, IEEE, 2008.

[26] M. Dorigo and T. Stützle. *Ant Colony Optimization.* MIT Press, 2004.

[27] S. Droste, T. Jansen, and I. Wegener. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science*, 276:51–81, 2002.

[28] J. J. Durillo, A. J. Nebro, and E. Alba. The jMetal framework for multi-objective optimization: Design and architecture. In *Proc. Congress on Evolutionary Computation (CEC '10)*, pp. 4138–4325. IEEE, 2010.

[29] G. Durrett, F. Neumann, and U.-M. O'Reilly. Computational complexity analysis of simple genetic programing on two problems modeling isolated program semantics. In *Proc. 11th Workshop on Foundations of Genetic Algorithms (FOGA XI)*, pp. 69–80. ACM, 2011.

[30] M. Ehrgott. *Multicriteria optimization.* Springer, 2nd edition, 2005.

[31] A. Eiben and J. Smith. *Introduction to Evolutionary Computing.* Springer, 2007.

[32] M. T. M. Emmerich, N. Beume, and B. Naujoks. An EMO algorithm using the hypervolume measure as selection criterion. In *Proc. Third International Conference on Evolutionary Multi-Criterion Optimization (EMO '05)*, pp. 62–76. Springer, 2005.

[33] Evolved Analytics LLC. *DataModeler 8.0.* Evolved Analytics LLC, 2010. URL www.evolved-analytics.com.

[34] T. Friedrich, J. He, N. Hebbinghaus, F. Neumann, and C. Witt. Approximating covering problems by randomized search heuristics using multi-objective models. *Evolutionary Computation*, 18:617–633, 2010.

[35] O. Giel. Expected runtimes of a simple multi-objective evolutionary algorithm. In *Proc. Congress on Evolutionary Computation (CEC '03)*, pp. 1918–1925. IEEE, 2003.

[36] O. Giel and P. K. Lehre. On the effect of populations in evolutionary multi-objective optimisation. *Evolutionary Computation*, 18:335–356, 2010.

[37] O. Giel and I. Wegener. Evolutionary algorithms and the maximum matching problem. In *Proc. 20th Symposium on Theoretical Aspects of Computer Science (STACS '03)*, Vol. 2607 of *Lecture Notes in Computer Science*, pp. 415–426. Springer, 2003.

[38] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* MIT Press, 1989.

[39] D. E. Goldberg and U.-M. O'Reilly. Where does the good stuff go, and why? how contextual semantics influences program structure in simple genetic programming. In *Proc. 1st European Workshop on Genetic Programming (EuroGP '98)*, Vol. 1391 of *Lecture Notes in Computer Science*, pp. 16–36, 1998.

[40] M. Gong, L. Jiao, H. Du, and L. Bo. Multiobjective immune algorithm with nondominated neighbor-based selection. *Evolutionary Computation*, 16:225–255, 2008.

[41] Google. Google Imagery, 2011. © 2011 Google - Imagery © 2011 TerraMetrics, Map data © 2011 Google, Whereis(R), Sensir Pty Ltd.

[42] R. L. Graham, B. D. Lubachevsky, K. J. Nurmela, and P. R. J. Östergård. Dense packings of congruent circles in a circle. *Discrete Mathematics*, 181:139–154, 1998.

[43] N. Hansen. The CMA evolution strategy: a comparing review. In *Towards a new evolutionary computation. Advances in estimation of distribution algorithms*, pp. 75–102. Springer, 2006.

[44] E. Happ, D. Johannsen, C. Klein, and F. Neumann. Rigorous analyses of fitness-proportional selection for optimizing linear functions. In *Proc. 10th Genetic and Evolutionary Computation Conference (GECCO '08)*, pp. 953–960. ACM, 2008.

[45] J. He and X. Yao. Drift analysis and average time complexity of evolutionary algorithms. *Artificial Intelligence*, 127:57 – 85, 2001.

[46] S. Huband, L. Barone, R. L. While, and P. Hingston. A scalable multi-objective test problem toolkit. In *Evolutionary Multi-Criterion Optimization, (EMO '05)*, Vol. 3410 of *Lecture Notes in Computer Science*, pp. 280–295. Springer, 2005.

[47] C. Igel, N. Hansen, and S. Roth. Covariance matrix adaptation for multi-objective optimization. *Evolutionary Computation*, 15:1–28, 2007.

[48] H. Ishibuchi, N. Tsukamoto, Y. Sakane, and Y. Nojima. Indicator-based evolutionary algorithm with hypervolume approximation by achievement scalarizing functions. In *Proc. 12th Conference on Genetic and Evolutionary Computation Conference (GECCO '10)*, pp. 527–534. ACM, 2010.

[49] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proc. IEEE International Conference on Neural Networks (ICNN '95)*, Vol. 4, pp. 1942–1948. IEEE, 1995.

[50] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, 1992.

[51] A. Kusiak and Z. Song. Design of wind farm layout for maximum wind energy capture. *Renewable Energy*, 35:685 – 694, 2010.

[52] W. B. Langdon and R. Poli. Fitness causes bloat. In *Proc. Soft Computing in Engineering Design and Manufacturing*, pp. 23–27. Springer, 1997.

[53] J. Lässig and D. Sudholt. Experimental supplements to the theoretical analysis of migration in the island model. In *Proc. 11th International Conference on Parallel Problem Solving from Nature (PPSN XI)*, Vol. 6238 of *Lecture Notes in Computer Science*, pp. 224–233. Springer, 2010.

[54] M. Laumanns, L. Thiele, K. Deb, and E. Zitzler. Combining convergence and diversity in evolutionary multiobjective optimization. *Evolutionary Computation*, 10(3):263–282, 2002.

[55] M. Laumanns, L. Thiele, and E. Zitzler. Running time analysis of multiobjective evolutionary algorithms on pseudo-boolean functions. *IEEE Transactions on Evolutionary Computation*, 8:170–182, 2004.

[56] H. Li and Q. Zhang. Multiobjective optimization problems with complicated pareto sets, MOEA/D and NSGA-II. *IEEE Transactions on Evolutionary Computation*, 13:284–302, 2009.

[57] R. T. Marler and J. S. Arora. Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, 26:369–395, 2004.

[58] Z. Michalewicz. Quo vadis, evolutionary computation? - on a growing gap between theory and practice. In *World Congress on Computational Intelligence (WCCI '12), Plenary/Invited Lectures*, Vol. 7311 of *Lecture Notes in Computer Science*, pp. 98–121. Springer, 2012.

[59] A. A. Montaño, C. A. C. Coello, and E. Mezura-Montes. Multiobjective evolutionary algorithms in aeronautical and aerospace engineering. *IEEE Transactions Evolutionary Computation*, 16:662–694, 2012.

[60] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[61] I. Mustakerov and D. Borissova. Wind park layout design using combinatorial optimization. In *Wind Turbines.* InTech, 2011.

[62] F. Neumann. Computational complexity analysis of multi-objective genetic programming. In *Proc. 14th International Conference on Genetic and Evolutionary Computation Conference (GECCO '12)*, pp. 799–806. ACM, 2012.

[63] F. Neumann and I. Wegener. Minimum spanning trees made easier via multi-objective optimization. In *Proc. 7th Genetic and Evolutionary Computation Conference (GECCO '05)*, pp. 763–770. ACM, 2005.

[64] F. Neumann and C. Witt. *Bioinspired Computation in Combinatorial Optimization - Algorithms and Their Computational Complexity.* Springer, 2010.

[65] F. Neumann, U.-M. O'Reilly, and M. Wagner. Computational complexity analysis of genetic programming - initial results and future directions. In *Proc. Genetic Programming Theory and Practice (GPTP IX)*, Genetic and Evolutionary Computation, pp. 113–128. Springer, 2011.

[66] H. Neustadter. Method for evaluating wind turbine wake effects on wind farm performance. *Journal of Solar Energy Engineering*, pp. 107–240, 1985.

[67] A. Nguyen, T. Urli, and M. Wagner. Single- and multi-objective genetic programming: new bounds for weighted order and majority. In *Proc. 12th Workshop on Foundations of Genetic Algorithms (FOGA XII)*, pp. 161–172. ACM, 2013.

[68] A. Q. Nguyen, T. Urli, and M. Wagner. Improved computational complexity results for weighted order and majority. In *Proc. Foundations of Genetic Algorithms (FOGA XII)*, 2013. To be published.

[69] P. S. Oliveto and C. Witt. Simplified drift analysis for proving lower bounds in evolutionary computation. *Algorithmica*, 59:369–386, 2011.

[70] U.-M. O'Reilly. *An Analysis of Genetic Programming.* PhD thesis, Carleton University, 1995.

[71] U.-M. O'Reilly and F. Oppacher. Program search with a hierarchical variable length representation: Genetic programming, simulated annealing and hill climbing. In *Proc. 3rd International Conference on Parallel Problem Solving from Nature (PPSN III)*, number 866 in Lecture Notes in Computer Science, pp. 397–406. Springer, 1994.

[72] C. H. Papadimitriou and M. Yannakakis. On the approximability of trade-offs and optimal access of web sources. In *Proc. 41st Symposium on Foundations of Computer Science (FOCS '00)*, pp. 86–92. IEEE, 2000.

[73] R. Poli, W. B. Langdon, and N. F. McPhee. *A Field Guide to Genetic Programming.* lulu.com, 2008.

[74] P.-E. Rethore. State of the art in wind farm layout optimization. *Wind Energy Research*, p. 179, 2010.

[75] G. Rudolph. *Convergence properties of evolutionary algorithms.* Kovac, 1997.

[76] J. Scharnow, K. Tinnefeld, and I. Wegener. The analysis of evolutionary algorithms on sorting and shortest paths problems. *Journal of Mathematical Modelling and Algorithms*, 3:349–366, 2004.

[77] O. Schütze, M. Laumanns, C. A. C. Coello, M. Dellnitz, and E.-G. Talbi. Convergence of stochastic search algorithms to finite size pareto set approximations. *Journal of Global Optimization*, 41:559–577, 2008.

[78] O. Schütze, M. Laumanns, E. Tantar, C. A. C. Coello, and E.-G. Talbi. Computing gap free pareto front approximations with stochastic search algorithms. *Evolutionary Computation*, 18:65–96, 2010.

[79] H.-P. P. Schwefel. *Evolution and optimum seeking: the sixth generation.* John Wiley & Sons, Inc., 1993.

[80] P. G. Szabó, M. C. Markót, T. Csendes, E. Specht, L. G. Casado, and I. Garcãa. *New Approaches to Circle Packing in a Square.* Springer, 2007.

[81] R. Tran, J. Wu, C. Denison, T. Ackling, M. Wagner, and F. Neumann. Fast and effective multi-objective optimisation of wind turbine placement. In *Proc. 15th Conference on Genetic and Evolutionary Computation (GECCO '13).* ACM, 2013.

[82] J. Tzanos, K. Margellos, and J. Lygeros. Optimal wind turbine placement via randomized optimization techniques. In *Power Systems Computation Conference*, 2011.

[83] E. Ulungu and J. Teghem. Multi-objective combinatorial optimization problems: A survey. *Journal of Multi-Criteria Decision Analysis*, 3:83–104, 1994.

[84] T. Urli, M. Wagner, and F. Neumann. Experimental supplements to the computational complexity analysis of genetic programming for problems modelling isolated program semantics. In *Proc. 12th International Conference on Parallel Problem Solving from Nature (PPSN XII)*, Vol. 7491 of *Lecture Notes in Computer Science*, pp. 102–112. Springer, 2012.

[85] D. A. Van Veldhuizen and G. B. Lamont. Multiobjective evolutionary algorithm research: A history and analysis. Technical report, Citeseer, 1998.

[86] M. Wagner and T. Friedrich. Efficient parent selection for approximation-guided evolutionary multi-objective optimization. In *Proc. Congress on Evolutionary Computation (CEC '13)*. IEEE, 2013.

[87] M. Wagner and F. Neumann. Parsimony pressure versus multi-objective optimization for variable length representations. In *Proc. 12th International Conference on Parallel Problem Solving from Nature (PPSN XII)*, Vol. 7491 of *Lecture Notes in Computer Science*, pp. 133–142. Springer, 2012.

[88] M. Wagner and F. Neumann. A fast approximation-guided evolutionary multi-objective algorithm. In *Proc. 15th Conference on Genetic and Evolutionary Computation (GECCO '13)*. ACM, 2013.

[89] M. Wagner, K. Veeramachaneni, F. Neumann, and U.-M. O'Reilly. Optimizing the layout of 1000 wind turbines. In *Proc. European Wind Energy Association Event (EWEA '11)*, 2011.

[90] M. Wagner, J. Day, and F. Neumann. A fast and effective local search algorithm for optimizing the placement of wind turbines. *Renewable Energy*, 51:64 – 70, 2013.

[91] C. Wan, J. Wang, G. Yang, X. Li, and X. Zhang. Optimal micro-siting of wind turbines by genetic algorithms based on improved wind and turbine models. In *Proc. 48th IEEE Conference on Decision and Control, held jointly with the 2009 28th Chinese Control Conference (CDC/CCC '09)*, pp. 5092–5096, 2009.

[92] C. Wan, J. Wang, G. Yang, X. Li, and X. Zhang. Optimal siting of wind turbines using real coded genetic algorithms. In *Proc. European Wind Energy Association Conference and Exhibition (EWEA '09)*, 2009.

[93] C. Wan, J. Wang, G. Yang, and X. Zhang. Optimal micro-siting of wind farms by particle swarm optimization. In *Advances in Swarm Intelligence*, Vol. 6145 of *Lecture Notes in Computer Science*, pp. 198–205. Springer, 2010.

[94] I. Wegener. Methods for the analysis of evolutionary algorithms on pseudo-Boolean functions. In *Evolutionary Optimization*, pp. 349–369. Kluwer, 2002.

[95] R. Wiser and M. Bolinger. *Annual Report on U.S. Wind Power Installation, Cost, and Performance Trends 2006.* U.S. Department of Energy, 2007.

[96] E. Zitzler and S. Künzli. Indicator-based selection in multiobjective search. In *Proc. 8th International Conference Parallel Problem Solving from Nature (PPSN VIII)*, Vol. 3242 of *Lecture Notes in Computer Science*, pp. 832–842. Springer, 2004.

[97] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. *IEEE Transactions Evolutionary Computation*, 3:257–271, 1999.

[98] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the strength Pareto evolutionary algorithm for multiobjective optimization. In *Proc. Evolutionary Methods for Design, Optimisation and Control with Application to Industrial Problems (EUROGEN '01)*, pp. 95–100. International Center for Numerical Methods in Engineering (CIMNE), 2002.